Osztályozás alapjai

Contents

1 Bevezetés

```
# Használt csomagok
library(ISLR)
library(caret)
library(RWeka)
library(e1071)
```

Az osztályozás (classification) alapvető fontosságú adatelemzési feladat, egyike a legelterjedtebben alkalmazott adatelemzési módszereknek. Az osztályozás feladata egy adathalmaz elemeihez egy-egy osztály hozzárendelése, amelyet legkönnyebben úgy tudunk elképezelni, hogy az adathalmaz elemeihez felcímkézzük. Fontos kiemelni, hogy általános osztályozási feladatnál minden adatsor egy és csak is egy címkét kaphat, tehát az egyes osztályok diszjunktak. Ezzel szemben elképzelhető olyan osztályozás is, ahol az egyes adatsorok egy vagy több címkét is kaphatnak – ezzel az esettel nem foglalkozunk (Multi-label classification).

Az osztályokat elméleti tárgyalásokban általában C_k jellel jelöljük ($k = 1, \ldots, K$), és ekkor azt mondjuk, hogy K darab osztályt ismerünk. Az osztályokkal szemben a következő követelményeket támasztjuk:

- Az osztályok előre definiáltak, tehát ismertek az osztályozás során. Az osztályok megállapítását elvégezhetjük előzetes ismeretek alapján, a szakterület ismeretében, vagy különböző csoportosítási algoritmusokkal.
- Az egyes osztályok megkülönböztethetőek (nominális adatok).
- Megszámlálhatóan sok osztály létezik.

Az osztályokkal szemben nem követeljük meg, hogy sorbarendezhetők legyenek (nem ordinálisak). Ha egy adott $\mathbf{x}$ adatsor a C_k osztályhoz tartozik, akkor azt általában $\mathbf{x} \in C_k$ jelöléssel jelöljük. Az osztályozás elvégzését az ún. osztályozó végzi, amely tulajdonképpen egy olyan függvény, melynek diszkrét az értékkészlete, és az n-dik $\mathbf{x}_n$ adatsorhoz hozzárendeli az osztály indexét:

$$y_n = f(\mathbf{x}_n) \qquad y_n \in \{1, 2, \ldots, k\}$$

Az osztályozás tehát alapvetően hasonlít a regresszióhoz, csak a kimenete nem egy valós szám, hanem egy egész szám, amely azt jelöli, az adott adatsor melyik osztályhoz tartozik (tehát pl. $y_n = 5$ esetén $x_n \in C_5$). Számos esetben az osztályok indexelése 0-tól kezdődik, de ez nem változtat az osztályozás alapelvein.

Elképzelhetjük például, hogy az adósokat egy bank három kategóriába sorol:

- alacsony kockázatú (C_1)

- közepes kockázatú (C_2)
- magas kockázatú (C_3).

Ebben az esetben három osztályról beszélhetünk, zárójelben láthatjuk az osztályokhoz rendelt jelöléseket. Ezek az osztályok jól megkülönböztethetők, ugyanakkor sorba is rendezhetők. Ez utóbbi nem kötelező, erre tekintsük például a csapadékok osztályozását (nem teljes lista):

- eső
- jégeső
- hó

Ez a három osztály általánosságban nem rendezhető sorrendbe, ugyanakkor bizonyos alkalmazási esetekben rendezhetők például károkozás szerint sorrendbe, hiszen az eső például kisebb kárt okoz (tipikusan), mint a jégeső. A sorrendezéssel a továbbiakban nem foglalkozunk.

Ha csupán két osztályunk van ($K = 2$), akkor bináris osztályozásról (binary classification) beszélünk, egyébként többosztályos osztályozásról (multiclass classification). A bináris osztályozás kiemelt jelentőségű, hiszen számos esetben a többosztályos osztályozás is megoldható bináris osztályozókkal, például tekintsük az adósbesorolást, ahol minden osztályt az összes többivel hasonlítunk össze (**one-to-rest**):

- alacsony kockázatú $\leftrightarrow$ egyéb
- közepes kockázatú $\leftrightarrow$ egyéb
- magas kockázatú $\leftrightarrow$ egyéb

Másik eshetőség, hogy az osztályozás során eldöntjük két kiválasztott osztály közül melyikhez tartozik az adott adatsor, és így K osztály esetén $\binom{K}{2}$ osztályozást végzünk (**one-to-one**):

- alacsony kockázatú $\leftrightarrow$ magas kockázatú
- alacsony kockázatú $\leftrightarrow$ közepes kockázatú
- közepes kockázatú $\leftrightarrow$ magas kockázatú

Persze nem árt az óvatosság a többosztályos osztályozás bináris osztályozással történő megoldása során, hiszen például a második esetben (one-to-one), ha minden osztályozó hozzárendel egy címkét az adatsorhoz, akkor minimum két címkét fogunk kapni az osztályozás végén. Erre a problémára számos megoldás létezik, azonban itt ezeket nem tárgyaljuk.

Osztályozást számos területen használnak, például:

- **Hitelbírálat**: adott ügyfél adatai (életkor, munkakör, fizetés, stb.) és hitel (összeg) esetén tudja majd fizetni a hitelt?
- **SPAM detekció**: Egy email szövegéből megállapítani, hogy SPAM, vagy sem (bináris osztályozás)
- **Vásárlási szokások**: megállapítani, hogy egy adott reklámra fogékony-e valaki, vagy sem
- **Objektum detekció**: képi jellemzőkből megállapítani, hogy mi látható a képen
- **Beszéd felismerés**: melyik karaktert vagy szót mondtuk

1.1 Osztályok ábrázolása és kódolása

Osztályozási algoritmusok vizsgálata során rendszeresen felmerül a kérdés, hogy miként célszerű az egyes osztályokat reprezentálni, hogy azokkal az egyes algoritmusok is számolni tudjanak. Emberi ábrázolásra legcélszerűbb persze az egyes osztályok szöveges ábrázolása, például az előbbi adós besorolásnál maradva az egyes osztályok:

- $C_1 \rightarrow$ `'alacsony kockázatú'`
- $C_2 \rightarrow$ `'közepes kockázatú'`
- $C_3 \rightarrow$ `'magas kockázatú'`

Nem nehéz belátni, hogy ilyen ábrázolással a számítógép nem sokat tud kezdeni, ezért célszerű az egyes osztályokat egész számoknak megfeleltetni. Így például a következő ábrázolást kapjuk:

- $C_1 \rightarrow 1$
- $C_2 \rightarrow 2$

- $C_3 \rightarrow 3$

Szerencsénkre R nyelv alatt a kettő megoldás összepárosítható faktorok segítségével, hiszen azok egyszerre képesek a két reprezentációra. A faktorok esetén az osztályok a faktor szintjeinek (level) felelnek meg. Például:

```
class<-factor(c("alacsony kockázatú","alacsony kockázatú","magas kockázatú","közepes kockázatú"))
print(as.numeric(class))
```

```
## [1] 1 1 3 2
```

```
print(levels(class))
```

```
## [1] "alacsony kockázatú" "közepes kockázatú"  "magas kockázatú"
```

Speciális ábrázolási módszer az ún. one-hot kódolás, amikor bináris számként ábrázoljuk az egyes osztályokat, ahol a k-dik osztály kódolása során a k-dik helyiértéken szerepel 1-es, mindenhol máshol 0. A kódszó hossza az osztályok K számával egyezik meg:

- $C_1 \rightarrow 001$
- $C_2 \rightarrow 010$
- $C_3 \rightarrow 100$

Ennek a kódolásnak az alternatívája, amikor nem bináris számokat, hanem vektorokat használunk:

- $C_1 \rightarrow [0, 0, 1]$
- $C_2 \rightarrow [0, 1, 0]$
- $C_3 \rightarrow [1, 0, 0]$

Ez a kódolás valószínűségi modelleknél fog nagy hasznunkra válni, amikor látni fogjuk, hogy $p_k = P(C_k \mid \mathbf{x})$ valószínűségekkel jelölhetjük azt, hogy az $\mathbf{x}$ adatsor a k-dik osztályhoz tartozik, és az egyes valószínűségeket vektorokban adhatjuk meg, pl.:

$$[p_1, p_2, p_3] = [0.02, 0.96, 0.02]$$

A példában, az adatsor 96% valószínűség szerint a 2-es osztályba tartozik és 2-2% szerint az 1-es és 3-as osztályokba.

Érdemes megemlíteni, hogy a one-hot kódolásnak bizonyos esetekben azt a változatát használják, amikor a K darab osztályt $K-1$ hosszúságú kóddal írnak le, hiszen az utolsó osztályra gondolhatunk úgy is, mint az „egyik másik osztály sem”. Ebben az esetben:

- $C_1 \rightarrow 01$
- $C_2 \rightarrow 10$
- $C_3 \rightarrow 00$

1.2 Osztályozás célja, módszertana

Az osztályozás elsődleges célja, ahogy már említettük, adatsorok előre definiált osztályokba sorolása. Ez persze feltételezi egy olyan osztályozási módszer, modell vagy függvény ismeretét, amely az osztályozást lehetővé teszi. Ezeket a modelleket kétféleképpen tudjuk előállítani.

1. A szakterület vagy tématerület ismeretében **közvetlenül felírhatjuk az osztályozást elvégző algoritmust vagy szabályrendszert**. Ilyen szabályrendszer lehet például egyes betegségek ismeretében a tünetek alapján történő döntés, amelyek alapján például az orvosok dolgoznak. Ilyen esetben konkrét döntések segítségével osztályozhatjuk az adatsorokat, például a pácienseket, hogy betegek-e vagy sem. Természetesen számos esetben ilyen előre definiálható szabályrendszer nem létezik, vagy a szabályok nagyon nehezen állapíthatók meg, esetleg a cél éppen e szabályrendszer megismerése.
2. A másik, gyakrabban használt esetben a döntéshez szükséges szabályrendszereket, függvényeket egy ún. **tanuló algoritmus segítségével** állapítjuk meg. A tanuló algoritmusok egy előre osztályozási modellek vagy függvények paramétereit, vagy éppen a teljes modellt automatikusan képesek meghatározni

tanuló adatok ismeretében. A tanuló adatok olyan adatsorok, melyekre az adatsorok címkézése már megtörtént, például kézi erővel, vagy ismert példák alapján. Ilyen lehet, amikor múltbeli adatok alapján szeretnénk olyan osztályozó algoritmust alkotni, amely jövőbeli adatsorok osztályozását képes elvégezni. A következőkben nemsokára látni fogunk tanuló adatokra példát.

2 Osztályozás kiértékelése

Amennyiben egy osztályozó algoritmust megalkottunk, és tanítási eljárást végrehajtottuk, az eredmények ismeretében meg kell állapítanunk, mennyire működik jól az osztályozó algoritmusunk. Arra gondolnánk intuitív gondolatmenet alapján, hogy a helyes osztályozások számának aránya jól leírja az osztályozó működését, de mint látni fogjuk, nem mindig ez a helyzet. Nehezíti a helyzetünket, hogy bizonyos algoritmusok olyan mértékben idomulni tudnak a tanuló adatokhoz, hogy azokra ugyan közel hibátlan osztályozást képesek végezni, de a tanuló adatokon kívüli adatsorokra (pl. jövőbeli adatokra) nagyon rossz teljesítményt nyújtanak. Ezzel az egyébként kulcsfontosságú problémával – az ún. túlillesztéssel (overfit) – vázlatosan foglalkozunk, de mélyebb tárgyalása nem célunk, csupán az alapelvek megismerését és a probléma felvetését tűzzük ki célul.

2.1 Motiváció, bevezető példa

A következőkben egy egyszerű adathalmazt fogunk használni, amely hitelkártya tartozások késedelmes törlesztéséről nyújt képet pár egyszerű attribútum segítségével. Az adathalmaz ugyan szimulált, de nagyon jó szolgálatot tesz az alapelvek megismerésére. Töltsük be az adathalmazt és tekintsük át:

```
df<-Default
summary(df)
```

```
##  default    student       balance           income
##  No :9667   No :7056   Min.   :   0.0   Min.   :  772
##  Yes: 333   Yes:2944   1st Qu.: 481.7   1st Qu.:21340
##                        Median : 823.6   Median :34553
##                        Mean   : 835.4   Mean   :33517
##                        3rd Qu.:1166.3   3rd Qu.:43808
##                        Max.   :2654.3   Max.   :73554
```

Az adathalmaz négy oszlopot tartalmaz, melyek:

- *income*: az ügyfél bevétele
- *balance*: az ügyfél számlájának átlagos egyenlege a törlesztés után
- *student*: faktor, amely mutatja, hogy az ügyfél hallgató-e? (Yes → hallgató)
- *default*: faktor, amely mutatja, hogy történt-e késedelmes törlesztés? (Yes → késedelmes törlesztés)

Tekintsük osztályozási célnak a *default* oszlopot, amely segítségével eldönthetjük a maradék attribútumok alapján, hogy az ügyfél késedelmes törlesztést fog-e végezni. Jól látható, hogy ez egy bináris osztályozás, mely során a *Yes* és *No* osztályok közül tudunk választani.

Készítsünk egy nagyon egyszerű, ún. **1R** (one-rule) osztályozót, melyek tulajdonsága, hogy mindig egyetlen attribútum alapján végzik el az osztályozást. Az 1R osztályozókra még visszatérünk, de jelenleg alkalmazzuk azt a hasraütés alapú szabályt, hogy: *osztályozzunk minden adatsort késedelmesnek, amely esetén az átlagos egyenleg nagyobb mint 800*! Lássuk ezt R kódban, amely mutatja, hogy az osztályozást végül faktorrá alakítjuk, az elvárt No/Yes formában, ahogy azt a `default` oszlop tartalmazza:

```
df$pred_1r<-df$balance>800
df$pred_1r<-factor(df$pred_1r,c(FALSE,TRUE),c('No','Yes'))
head(df)
```

```
##   default student   balance    income pred_1r
## 1      No      No  729.5265 44361.625      No
## 2      No     Yes  817.1804 12106.135     Yes
```

```
## 3      No      No 1073.5492 31767.139     Yes
## 4      No      No  529.2506 35704.494      No
## 5      No      No  785.6559 38463.496      No
## 6      No     Yes  919.5885  7491.559     Yes
```

Láthatjuk, hogy `pred` oszlop értéke nem mindig egyezik a `default` oszlop értékével, tehát számos helyen hibás az osztályozásunk. Számoljuk össze, hány esetben egyeznek:

```
print(sum(df$pred_1r == df$default))
```

```
## [1] 5146
```

Intuitív gondolatmenettel azt mondhatjuk, hogy mivel az adathalmaz 10.000 sort tartalmaz, ezért ez nagyjából 51,46% pontosság. Megállapíthatjuk, hogy nem túl pontos az osztályozónk, és valóban, 5.146 esetben hibásan döntött az osztályozó.

Tekintsük most azt az esetet, amikor nem 1R osztályozót használunk, hanem ún. konstans osztályozót, tehát minden adatsort egy kiválasztott osztályra osztályozzuk. Legyen most az osztályozási szabályunk az, hogy *osztályozzunk minden adatrekordot 'nem késelemesre' osztályozunk, függetlenül az attribútumoktól*:

```
df$pred_const<-factor(FALSE,c(FALSE,TRUE),c('No','Yes'))
head(df)
```

```
##   default student   balance    income pred_1r pred_const
## 1      No      No  729.5265 44361.625      No         No
## 2      No     Yes  817.1804 12106.135     Yes         No
## 3      No      No 1073.5492 31767.139     Yes         No
## 4      No      No  529.2506 35704.494      No         No
## 5      No      No  785.6559 38463.496      No         No
## 6      No     Yes  919.5885  7491.559     Yes         No
```

Láthatjuk, hogy jelenleg az osztályozónk döntései általában megegyeznek az eredeti késedelmességi osztállyal. Számoljuk össze a helyes döntéseinket:

```
print(sum(df$pred_const == df$default))
```

```
## [1] 9667
```

Ez az eredmény azt mondja, hogy az osztályozásunk 96.67% pontosságú, ami nagyon szép eredmény. Ne felejtsük, hogy ez egy konstans osztályozó, tehát egyszerűen minden eseteben „Nem"-re döntöttünk. *Hol a hiba?*

Tekintsük mégegyszer az adathalmaz összesítését:

```
summary(df)
```

```
##  default    student       balance           income      pred_1r    pred_const
##  No :9667   No :7056   Min.   :   0.0   Min.   :  772   No :4819   No :10000
##  Yes: 333   Yes:2944   1st Qu.: 481.7   1st Qu.:21340   Yes:5181   Yes:    0
##                        Median : 823.6   Median :34553
##                        Mean   : 835.4   Mean   :33517
##                        3rd Qu.:1166.3   3rd Qu.:43808
##                        Max.   :2654.3   Max.   :73554
```

Azt látjuk, hogy összesen 9.667 nem késedelmes eset fordult elő, míg összesen 333 késedelmes. Ez utóbbi az adathalmaz 3.33%-a, így nem csoda, hogy fix döntés esetén csak nagyjából 3%-ot hibázunk.

A példa jól rámutat a jelenségre, amit **kiegyensúlyozatlan osztályeloszlásnak** (imbalanced data) hívunk. Amennyiben egy adathalmazban egy osztály jelentősen nagyobb számban fordul elő másik osztálynál, akkor azt az adathalmazt kiegyensúlyozatlan adathalmaznak nevezzük. Ennek tipikusan triviális okai vannak:

bizonyos események ritkábban fordulnak elő a valóságban. Ilyen például az, hogy valaki késedelmes fizetésbe esik, de ilyen például a rákos megbetegedések statisztikája is.

2.2 Keveredési mátrix

Hogy a bemutatott értékelési problémát elkerüljük, érdemes egy jól bevált eszközhöz, az ún. **keveredési mátrixhoz** (confusion matrix) folyamodnunk. A keveredési mátrixot a következő ábrán láthatjuk (forrás):

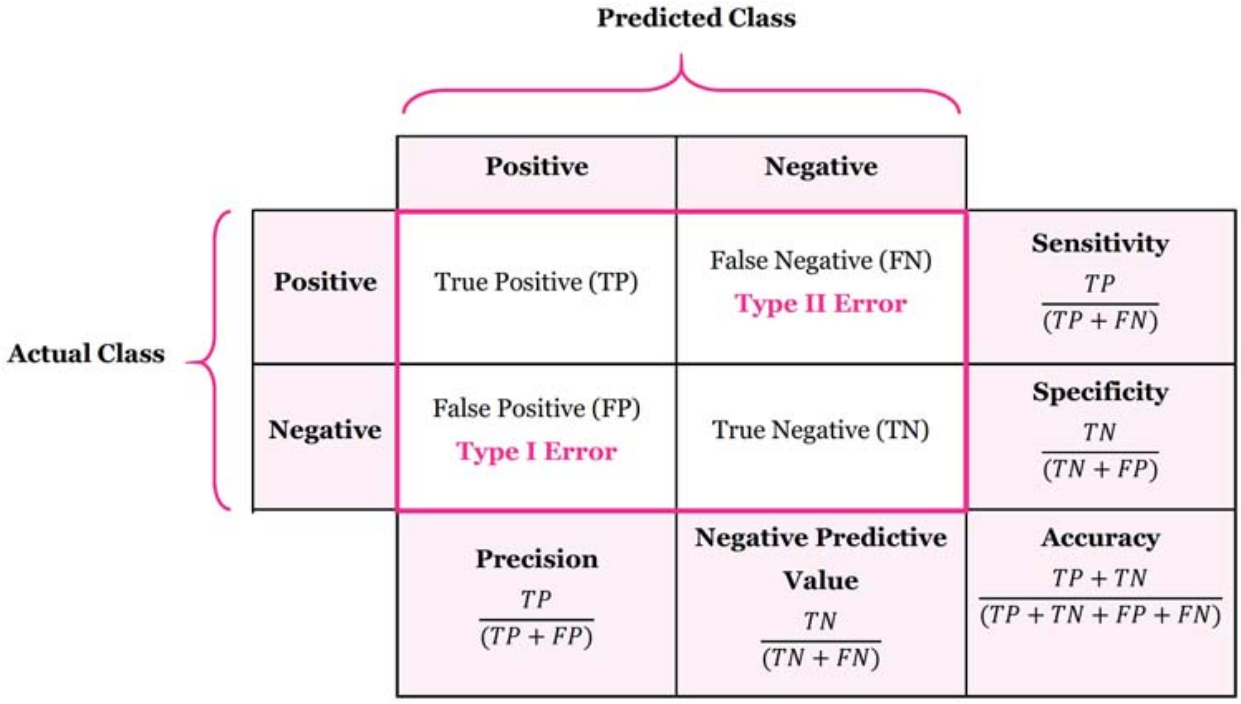


Figure 1: Keveredési mátrix

A keveredési mátrixnak saját terminológiája van, amellyel röviden megismerkedünk. A mátrix egy hipotézis kimeneteleinek a számát ábrázolja. A hipotézis lehet bármilyen feltevés, például esetünkben „késedelmes törlesztés történik". Ez lehet pozitív, tehát késedelmes törlesztés történt, vagy negatív, azaz nem történt késedelmes törlesztés. A mátrix egyik tengelyén a valóságot láthatjuk, tehát hogy a késedelmes törlesztés megtörtént-e vagy sem (itt gondolhatunk az adathalmazunk `default` oszlopára). A másik tengelyen a predikált eseményt láthatjuk, azaz, hogy az osztályozó szerint a hipotézis pozitív vagy sem (az adathalmazunkban a `pred` oszlop). Néha a mátrixot felcserélt tengelyekkel ábrázolják.

Az egyes cellák ezek kombinációinak a számát tartalmazzák. *True Positive (TP)* kimenetnek hívjuk, ha a hipotézis a valóságban megtörtént, és pozitívra osztályoztuk, tehát a valóságban késedelmes törlesztés történt, és arra is prediktáltuk. A másik sikeres osztályozási eset a *True Negative (TN)*, amikor a hipotézis a valóságban negatív és negatívra osztályoztuk, tehát például nem volt késedelmes törlesztés, és arra is osztályoztuk.

Láthatjuk, hogy hibából viszont kétfajta is van. Amennyiben a valóságban a hipotézis negatív, de pozitívra osztályoztuk, akkor elsőfajú hibáról beszélünk (*False Positive (FP)*), esetünkben, nem történt késedelmes törlesztés, de késedelmes törlesztésre osztályozuk. A másik hiba lehetőség, a másodfajú hiba (*False Negative (FN)*), amikor a hipotézis a valóságban pozitív, tehát fennáll, viszont negatívra osztályozzuk, példánkban késedelmes törlesztés történt, de nem arra osztályoztuk.

A mátrixon különböző mutatókat definiálhatunk, ezek láthatók az ábrán.

- **Pontosság** (accuracy). Az egyik kiemelten fontos mérőszám, hiszen megmutatja a helyesen osztályozott

esetek arányát.

- **Hiba arány** (error rate). A pontosság ellentéte, a hibás döntések aránya a teljes populációhoz mérten.
- **Szenzitivitás** (sensitivity, recall, true positive rate). Megadja, hogy a valóságban pozitív eseteket mekkora hányadban tudjuk helyesen osztályozni.
- **Specifitás** (specificity, selectivity, true negative rate). Megadja, hogy a valóságban negatív eseteket mekkora hányadban tudjuk helyesen osztályozni.
- **Precizitás** (precision). Megadja, hogy a pozitívra osztályozott adatsorok mekkora hányada került helyesen osztályozásra.
- **Negatív prediktív érték** (negative predictive value). Megadja, hogy a negatívra osztályozott adatsorok mekkora hányada került helyesen osztályozásra.

Rengeteg egyéb mutatót ismerünk, ezek megismerhetők például a Wikipedia oldalán. Ezek a fogalmak elsősorban statisztikából származnak, és jelentésük változhat a hipotézis megfogalmazásától függően, tehát érdemes odafigyelni a hipotézis helyes megválasztására.

A keveredési mátrixot az adatokból a `caret` csomagban található `confusionMatrix` függvénnyel tudjuk meghatározni, amely a keveredési mátrixon túl a leggyakrabban használt mutatókat is kiszámítja. Határozzuk tehát meg a keveredési mátrixot a *konstans osztályzónkra*!

```
print(confusionMatrix(df$pred_const,df$default,positive='Yes'))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  9667  333
##        Yes    0    0
##
##                Accuracy : 0.9667
##                  95% CI : (0.963, 0.9701)
##     No Information Rate : 0.9667
##     P-Value [Acc > NIR] : 0.5146
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.9667
##              Prevalence : 0.0333
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : Yes
##
```

Láthatjuk, hogy a pontosság 96.67%, ugyanakkor érdemes megfigyelni, hogy míg a specifitás 100%, azaz minden negatív esetet (amikor nem történt késedelmes fizetés) jól tudunk előre jelezni (ez természetes, hiszen mindig 'Nem'-re döntünk), addig a szenzitivitás 0, tehát egyetlen esetben sem sikerült eltalálni a késedelmes fizetéseket. Persze ezeket az információkat egyenesen látjuk a keveredési mátrixból is. Hasznos mérőszám a **kiegyensúlyozott pontosság** (balanced accuracy), amely a szenzitivitás és a specifitás átlaga, azaz

$$\text{kiegyensúlyozott pontosság} = \frac{\text{szenzitivitás} + \text{specifitás}}{2}$$

Ez a mutató jól érzékelteti, hogy az osztályozási döntések mindig kétoldalúak, tehát a negatív és pozitív esetek arányainak kiértékelése is fontos tényező, hiszen az adathalmaz *nem feltétlenül kiegyensúlyozott*.

Tekintsük meg a keveredési mátrixot az 1R döntésünkhöz is:

```
confusionMatrix(df$pred_1r,df$default,positive = 'Yes')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  4816    3
##        Yes 4851  330
##
##                Accuracy : 0.5146
##                  95% CI : (0.5048, 0.5244)
##     No Information Rate : 0.9667
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0609
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99099
##             Specificity : 0.49819
##          Pos Pred Value : 0.06369
##          Neg Pred Value : 0.99938
##              Prevalence : 0.03330
##          Detection Rate : 0.03300
##    Detection Prevalence : 0.51810
##       Balanced Accuracy : 0.74459
##
##        'Positive' Class : Yes
##
```

Jól látszik, hogy ebben az esetben a szenzitivitás magas, tehát meglehetősen pontosan jelzi, amikor késedelmes fizetés áll elő, ugyanakkor a specifitás, azaz a nem késedelmes fizetések becslésében elmarad, mindössze 49.8%-al. A pontossága ugyanakkor az osztályozásnak csak 51.4%, a kiegyensúlyozott pontossága 74.4%.

Fontos kiemelni, hogy egy modell mutatók alapján történő egyszerű kiértékelése nem mindig vezet jó eredményre. Vegyük észre, hogy az egyes mértékek értelmezése erősen függ a osztályozási problémától. Esetünkben egy bank szempontjából az a hasznos, ha magas a szenzitivitás, és jól jelezzük előre a késedelmes fizetéseket, ezzel szemben a specifitás kevésbé kritikus, hiszen kisebb kockázatot jelent a banknak egy jól fizető ügyfélnek nem adni hitelt, mint fordítva. Ezzel szemben persze az ügyfelek szempontjából ennek pont fordítottja igaz, hiszen jó adósként alacsony specifitással nehezen jutnak hitelhez.

A keveredési mátrix természetesen nem csak bináris osztályozás esetén értelmezhető, hanem tetszőleges mennyiségű osztály esetén. Ebben az esetben, ha K darab osztályt definiálunk, a keveredési mátrix $K \times K$ méretű lesz. Az egyes mértékek általánosítása már nehezebb, hiszen míg például a pontosság (accuracy) értelemszerűen számítható nagyobb dimenziókra, addig például a szenzitivitás csak osztályonként értelmezhető, nem a teljes mátrixra. A keveredési mátrix általánosítását itt nem mutatjuk be részletesen.

3 Referencia osztályozók

Bizonyos triviális osztályozókat, ún. referencia osztályozókat használhatunk annak megállapítására, milyen minimális viselkedést várunk el egy osztályozótól. A két legismertebb ilyen osztályozó a *konstans* osztályozó és a *véletlen* osztályozó. A célunk általában egy osztályozó algoritmus megalkotásával, hogy ezeket az osztályozókat valamely paraméterben felülmúljuk.

A konstans osztályozót már láttuk az adathalmazunkra, most tekintsük a véletlen osztályozót! Ebben az esetben a kimeneti osztályt véletlenszerűen választjuk meg:

```
df<-Default
df$pred<-as.factor(sample(c('Yes','No'),nrow(df),replace = T))
confusionMatrix(df$pred,df$default,positive = 'Yes')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  4835  149
##        Yes 4832  184
##
##                Accuracy : 0.5019
##                  95% CI : (0.4921, 0.5117)
##     No Information Rate : 0.9667
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0068
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.55255
##             Specificity : 0.50016
##          Pos Pred Value : 0.03668
##          Neg Pred Value : 0.97010
##              Prevalence : 0.03330
##          Detection Rate : 0.01840
##    Detection Prevalence : 0.50160
##       Balanced Accuracy : 0.52635
##
##        'Positive' Class : Yes
##
```

Természetesen a véletlenszerű osztályozó pontossága tipikusan 50% körül van, ahogy az eredményeken is tisztán látszik. A véletlenszerű osztályozásnak speciális esete a súlyozott véletlen osztályozás, melyben az egyes kimeneteket súlyozzuk azok előfordulási gyakoriságával. Az ilyen osztályozó szenzitivitása persze nagyon alacsony lesz kiegyensúlyozatlan adathalmaz esetében, például:

```
yes.count<-sum(df$default == 'Yes')
no.count<-sum(df$default == 'No')
df$pred<-as.factor(sample(c('Yes','No'),nrow(df),prob=c(yes.count,no.count),replace = T))
confusionMatrix(df$pred,df$default,positive = 'Yes')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  9325  327
```

```
##        Yes  342    6
##
##                Accuracy : 0.9331
##                  95% CI : (0.928, 0.9379)
##     No Information Rate : 0.9667
##     P-Value [Acc > NIR] : 1.0000
##
##                   Kappa : -0.017
##
##  Mcnemar's Test P-Value : 0.5883
##
##             Sensitivity : 0.01802
##             Specificity : 0.96462
##          Pos Pred Value : 0.01724
##          Neg Pred Value : 0.96612
##              Prevalence : 0.03330
##          Detection Rate : 0.00060
##    Detection Prevalence : 0.03480
##       Balanced Accuracy : 0.49132
##
##        'Positive' Class : Yes
##
```

4 Kiegyensúlyozás

Az adathalmaz kiegyensúlyozásának és az osztályozási problémák során felmerülő túlillesztési problémák megismerése kulcsfontosságú. Elmondhatjuk általánosságban, hogy e problémák felismerése és megfelelő kezelése sok esetben jobb osztályozási eredményt nyújthat, mint maga az osztályozási algoritmus megválasztása. A problémakör tárgyalásához áttekintjük az egyik legegyszerűbb osztályozási algoritmust, és példákon keresztül elemezzük a túlillesztés problémáját.

4.1 OneR algoritmus

A OneR (1R, one-rule) osztályozó – nevével ellentétben – nem egyetlen szabály, hanem egyetlen attribútum alapján végzi el az adatsorok osztályozását, és egyike a legegyszerűbb osztályozó algoritmusoknak. A OneR osztályozó az adathalmaz attribútumait végignézve minden attribútumra meghatározza az optimális dönti szabályokat különböző `if...then...` szabályok alkalmazásával, és kiválasztja azt az attribútumot, amely a legkisebb *hibaarányhoz* vezet.

A döntés diszkrét attribútomok esetén egyszerű szabályrendszerekhez vezet. Folytonos értékkészlet esetén az osztályozó a folytonos értékkészletet diszkretizálja (lásd. például `cut` függvény) különböző intervallumokra (`bin`), és ezeken az intervallumokon végzi el az osztályozást. A OneR algoritmus részletes működéséről, például az intervallumok kiszámításának módjáról itt olvashatunk.

Tekintsük meg a OneR osztályozót a példa adathalmazunkra! A OneR osztályozót egy formulával kell meghívni, amelyben kijelöljük a célváltozót és a vizsgálni kívánt attribútumokat. Jelen esetben az összes attribútumot megadjuk:

```
model<-OneR(default~.,Default)
print(model)
```

```
## balance:
##  < 1797.017329379948 -> No
##  < 1804.3019366823862    -> Yes
##  < 1806.9191881494771    -> No
```

```
##  < 1813.0928233701538    -> Yes
##  < 1888.6069130479348    -> No
##  < 1900.6002982884304    -> Yes
##  < 1918.1654198104468    -> No
##  < 1928.4362014925025    -> Yes
##  < 1950.6971840320266    -> No
##  < 2012.0916955956534    -> Yes
##  < 2034.1074763686904    -> No
##  < 2087.1074526760112    -> Yes
##  < 2121.0848955975766    -> No
##  >= 2121.0848955975766   -> Yes
## (9746/10000 instances correct)
```

```
summary(model)
```

```
##
## === Summary ===
##
## Correctly Classified Instances        9746               97.46   %
## Incorrectly Classified Instances       254                2.54   %
## Kappa statistic                          0.4724
## Mean absolute error                      0.0254
## Root mean squared error                  0.1594
## Relative absolute error                 39.3986 %
## Root relative squared error             88.8278 %
## Total Number of Instances            10000
##
## === Confusion Matrix ===
##
##     a    b   <-- classified as
##  9627   40 |    a = No
##   214  119 |    b = Yes
```

A módszer alapesetben a folytonos változókat annyi intervallumra bontja, hogy mindegyikben minimum 6 darab adatsor férjen el, majd az intervallumokra vizsgálja meg az osztályozás pontosságát. Láthatjuk a model leírásában, hogy az algoritmus a legjobb összefüggést a `balance` változóra találta meg, az egyenlegeket intervallumokba sorolta és azokra alkotta meg a döntési szabályt. A modell pontossága 97.46%. A kontingenciatáblázatból egyből láthatjuk, hogy az algoritmus a `Yes` célértékekkel bajban van, hiszen összesen 119-et tudott hatékonyan osztályozni a 333-ból. Tekintsük ezek alapján a keveredési mátrixot!

```
pred<-predict(model,Default)
confusionMatrix(pred,Default$default,positive='Yes')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  9627  214
##        Yes   40  119
##
##                Accuracy : 0.9746
##                  95% CI : (0.9713, 0.9776)
##     No Information Rate : 0.9667
##     P-Value [Acc > NIR] : 2.668e-06
##
##                   Kappa : 0.4724
```

```
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.3574
##             Specificity : 0.9959
##          Pos Pred Value : 0.7484
##          Neg Pred Value : 0.9783
##              Prevalence : 0.0333
##          Detection Rate : 0.0119
##    Detection Prevalence : 0.0159
##       Balanced Accuracy : 0.6766
##
##        'Positive' Class : Yes
##
```

A keveredési mátrix nyilvánvalóvá teszi, hogy a szenzitivitás elég alacsony, tehát a pozitív kimenetelt (`Yes`) kis hatékonysággal képes osztályozni a kiegyensúlyozatlan adathalmaz miatt. Ez persze nyilvánvaló az alapján, hogy az osztályozó elsősorban a pontosságot (accuracy) kívánja maximalizálni.

Fontos kiemelni, hogy a OneR osztályozó esetén kiemelt fontosságú a folytonos adatok diszkretizálásának módszere. Erre számos módszer létezik:

- az adatok által felölelt értékkészlet felbontása egyenletes hosszúságú intervallumokra
- az adatok által felölelt értékkészlet felbontása csoportok keresése által
- információelméleti módszerek, az entrópia vizsgálatával
- stb.

Az `RWeka` csomag `OneR` megvalósításában az intervallumfelbontást tudjuk szabályozni annak megadásával, hogy hány darab adatsor kerülhessen minimum egy intervallumba. Ezt az értéket a következőképpen tudjuk beállítani:

```
model<-OneR(default~.,Default,control=Weka_control(B=100))
print(model)
```

```
## balance:
##  < 2023.2041230146033    -> No
##  >= 2023.2041230146033   -> Yes
## (9721/10000 instances correct)
```

```
summary(model)
```

```
##
## === Summary ===
##
## Correctly Classified Instances        9721               97.21   %
## Incorrectly Classified Instances       279                2.79   %
## Kappa statistic                          0.3398
## Mean absolute error                      0.0279
## Root mean squared error                  0.167
## Relative absolute error                 43.2764 %
## Root relative squared error             93.0967 %
## Total Number of Instances            10000
##
## === Confusion Matrix ===
##
##     a    b   <-- classified as
##  9646   21 |    a = No
```

```
##   258   75 |    b = Yes
```

```
pred<-predict(model,Default)
confusionMatrix(pred,Default$default,positive='Yes')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  9646  258
##        Yes   21   75
##
##                Accuracy : 0.9721
##                  95% CI : (0.9687, 0.9752)
##     No Information Rate : 0.9667
##     P-Value [Acc > NIR] : 0.001123
##
##                   Kappa : 0.3398
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.2252
##             Specificity : 0.9978
##          Pos Pred Value : 0.7813
##          Neg Pred Value : 0.9739
##              Prevalence : 0.0333
##          Detection Rate : 0.0075
##    Detection Prevalence : 0.0096
##       Balanced Accuracy : 0.6115
##
##        'Positive' Class : Yes
##
```

Láthatjuk, hogy 100 pont esetén sokkal kevesebb döntési szabályt alkot az algoritmus, és a szenzitivitás is visszaesett.

4.2 Adathalmaz kiegyensúlyozása

Láttuk az előbbi példa során, hogy a kiegyensúlyozatlan adathalmaz meglehetősen rossz szenzitivitást eredményez az osztályozás során. Ennek elkerülése érdekében elvégezhetjük az adathalmaz kiegyensúlyozását. Tekintsük újra az adathalmaz felépítését:

```
summary(Default)
```

```
##  default    student       balance           income
##  No :9667   No :7056   Min.   :   0.0   Min.   :  772
##  Yes: 333   Yes:2944   1st Qu.: 481.7   1st Qu.:21340
##                        Median : 823.6   Median :34553
##                        Mean   : 835.4   Mean   :33517
##                        3rd Qu.:1166.3   3rd Qu.:43808
##                        Max.   :2654.3   Max.   :73554
```

Láthatjuk, hogy 333 `Yes` osztállyal szemben áll 9667 `No` osztály. Készítsünk egy adathalmazt, amelyben 100-100 adatsor szerepel mindegyik kimenetből, az egyes sorokat véletlenszerűen válasszuk ki!

```
set.seed(1)
df.split<-split(Default,Default$default)
```

```
df<-rbind(df.split$Yes[sample(nrow(df.split$Yes),100),],df.split$No[sample(nrow(df.split$No),100),])
summary(df)
```

```
##  default   student      balance           income
##  No :100   No :137   Min.   :   0.0   Min.   : 8638
##  Yes:100   Yes: 63   1st Qu.: 861.6   1st Qu.:20993
##                      Median :1323.2   Median :31573
##                      Mean   :1257.7   Mean   :32666
##                      3rd Qu.:1799.3   3rd Qu.:43386
##                      Max.   :2578.5   Max.   :66466
```

Ezzel a lépéssel tulajdonképpen az adathalmazt kiegyensúlyoztuk, hiszen a célváltozó egyenletesen fordul elő az adathalmazban. Végezzük el a kiegyensúlyozott adathalmazra az osztályozást OneR osztályozóval!

```
model<-OneR(default~.,df)
summary(model)
```

```
##
## === Summary ===
##
## Correctly Classified Instances         184               92      %
## Incorrectly Classified Instances        16                8      %
## Kappa statistic                          0.84
## Mean absolute error                      0.08
## Root mean squared error                  0.2828
## Relative absolute error                 16      %
## Root relative squared error             56.5685 %
## Total Number of Instances              200
##
## === Confusion Matrix ===
##
##   a  b   <-- classified as
##  93  7 |  a = No
##   9 91 |  b = Yes
```

```
pred<-predict(model,df)
confusionMatrix(pred,df$default,positive='Yes')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  93   9
##        Yes  7  91
##
##                Accuracy : 0.92
##                  95% CI : (0.8733, 0.9536)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.84
##
##  Mcnemar's Test P-Value : 0.8026
##
##             Sensitivity : 0.9100
##             Specificity : 0.9300
```

```
##          Pos Pred Value : 0.9286
##          Neg Pred Value : 0.9118
##              Prevalence : 0.5000
##          Detection Rate : 0.4550
##    Detection Prevalence : 0.4900
##       Balanced Accuracy : 0.9200
##
##        'Positive' Class : Yes
##
```

Láthatjuk, hogy az adathalmazra a pontosság (accuracy) összességében csökkent, viszont a szenzitivitás jelentősen nőtt. Ezzel egyetemben a kiegyensúlyozott pontosság is emelkedett, 92%-ra.

Érdemes a kiegyensúlyozás bemutatott módját azonban fenntartásokkal kezelni. Kiegyensúlyozatlan adathalmazok osztályozása általánosságban nagy kihívás, éppen ezért számos trükk és megoldási metodika jött létre a kiegyensúlyozatlanság kezelésére. A megfelelő módszer kiválasztása kreativitást és jártasságot kíván, ugyanakkor tipikusan az adott probléma szabja meg a hatékony módszert.

5 A `caret` csomag

A `caret` csomag egy átfogó eszköz regressziós és osztályozási feladatok megoldására. Több, mint 200 algoritmust ismer, és minden algoritmushoz egységes interfészt nyújt. Fontos kiemelni, hogy az egyes algoritmusok megvalósítását jól ismert csomagok segítségével végzi, mint például:

- **rpart**: döntési fák, pl. CART
- **kknn**: legközelebbi szomszéd megoldások
- **kernlab**: tartóvektor gépek
- stb.

A `caret` csomaghoz számos kisegítő függvény is tartozik, láttuk például korábban a `confusionMatrix()` függvényt. Ezen kívül hasznos szolgáltatásokat nyújt, mint például:

- adathalmaz feldarabolása, szétosztása, keverése
- jellemzők és prediktorok ábrázolása
- adatok előkészítése

A `caret` csomag legfontosabb feladata azonban az egyes adatleíró modellek tanítása, a hiperparaméterek meghatározása és a modell kiértékelésének megvalósítása. A következőkben az egyes osztályozási modellek bemutatása során rendszeresen a `caret` csomagot fogjuk használni, így jól elsajátíthatjuk az interfészek kezelését. Ne feledjük azonban, hogy minden modell esetén találunk speciális paramétereket, hiszen azok algoritmus függőek. Ezeket a paramétereket a használati utasításban felsorolt modellek mellett találjuk felsorolva itt.

5.1 Jellemzők ábrázolása

Gyakori probléma osztályozási feladatok során, hogy az egyes folytonos – esetleg diszkrét – jellemzők eloszlását ábrázoljuk a célosztály függvényében. Az egyes diszkrét változók esetében célszerűen inkább kontingencia táblázatot szoktunk használni:

```
df<-Default
table(df$default,df$student,dnn=c('default','student'))
```

```
##         student
## default   No  Yes
##     No  6850 2817
##     Yes  206  127
```

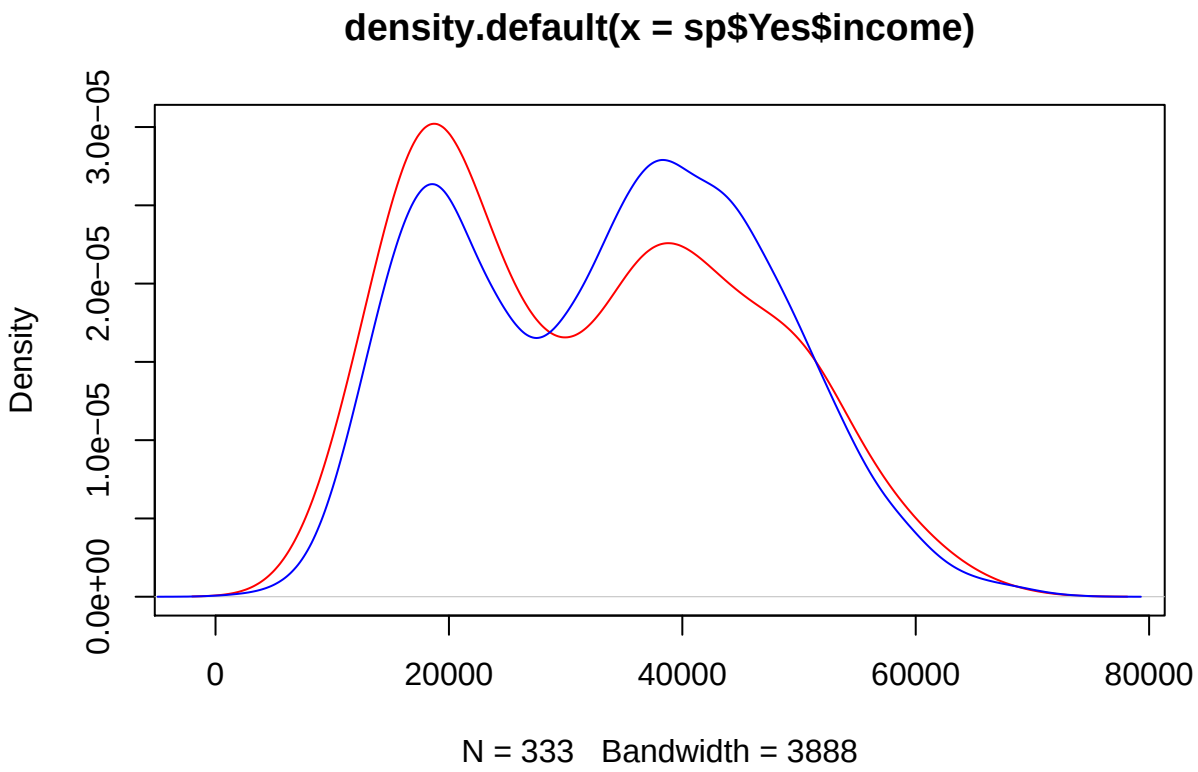
A táblázatból jól láthatjuk, hogy a `default` esetében nincs erős összefüggés aközött, hogy az ügyfél tanuló vagy sem, hiszen arányaiban a tanulók között ugyanannyi a késedelmes fizető, mint a nem tanulók között. Ezt jól láthatjuk az arányos táblázaton is (2.91 százalék és 4.3 százalék az arány):

```
prop.table(table(df$default,df$student,dnn=c('default','student')), margin=2)
```

```
##         student
## default         No        Yes
##     No  0.97080499 0.95686141
##     Yes 0.02919501 0.04313859
```

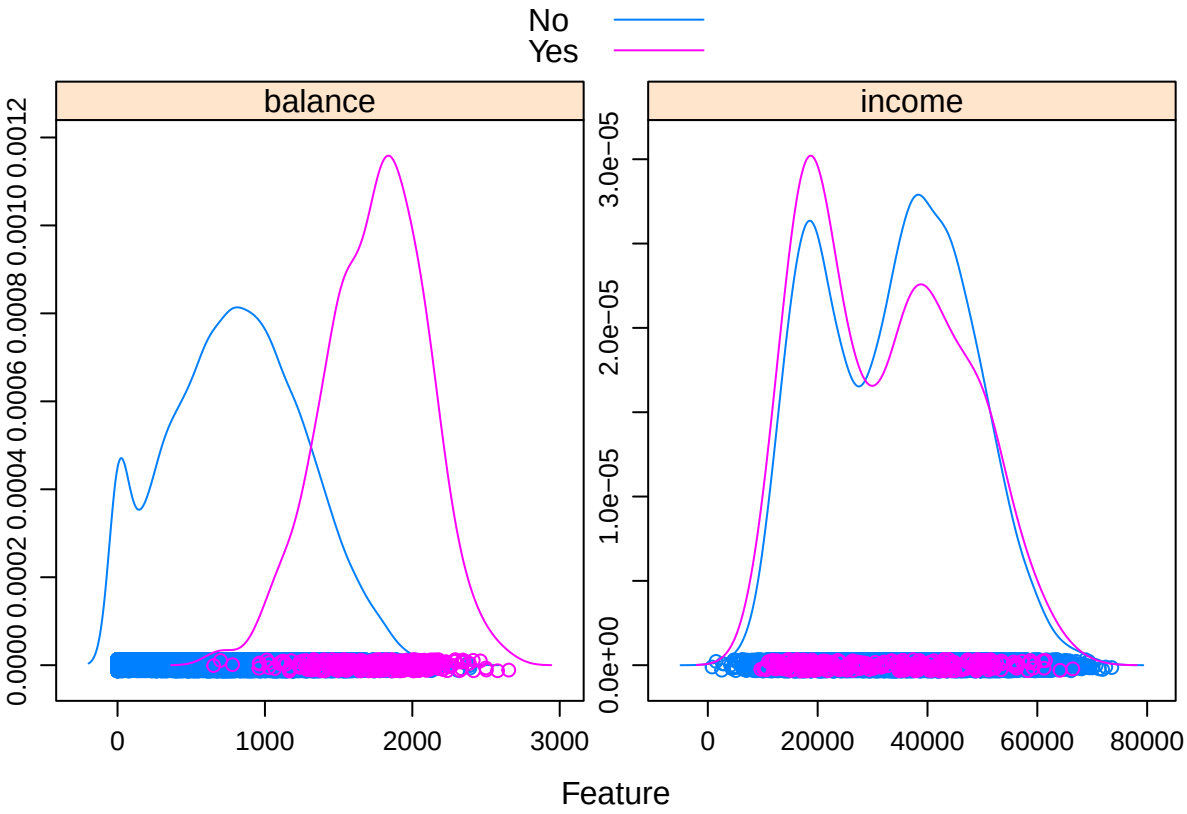
Folytonos változókra hasonló megfontolással az egyes prediktorok hatását eloszlás (sűrűség) diagrammal tudjuk reprezentálni. Gyakorlásképpen tekintsük meg ezt az `income` változóra egyszerű R eszközökkel:

```
sp<-split(Default, Default$default)
dyes<-density(sp$Yes$income)
dno<-density(sp$No$income)

plot(dyes,col='red')
lines(dno,col='blue')
```



Láthatjuk, hogy az eloszlásban nincs nagy különbség, az `income` prediktor nem különbözteti meg a késedelmes és nem késedelmes ügyfeleket. Szerencsére nem kell minden változóra ezt az ábrázolást elvégezni, a `caret` csomag lehetővé teszi *folytonos* változókra a jellemzők ábrázolását az osztály függvényében:

```
featurePlot(x=Default[,3:4],
            y=Default$default,
            plot='density',
            scales = list(x = list(relation="free"),
                          y = list(relation="free")),
            auto.key=T
            )
```



Láthatjuk, hogy az `income` változó esetén ugyanazt az ábrát kaptuk, de megjelent az eloszlás a `balance` változóra is, amely esetén tisztán kivehető, hogy a késedelmes és nem késedelmes fizetések eloszlása jelentősen eltér, tehát a `balance` változó egy *jó* prediktor lehet a késedelmes fizetések megállapítására.

5.2 Adat kiegyensúlyozás

Láttuk a korábbiakban, hogy a kiegyensúlyozatlan adathalmaz komoly tanítási problémákat tud okozni, és a szenzitivitást nagyon elrontja a legtöbb algoritmus esetén. Éppen ezért, a `caret` csomag tartalmaz számos megoldást az adathalmaz kiegyensúlyozása érdekében. Itt csak két alapvető megoldát tárgyalunk, az alulmintavételezést és a túlmintavételezést.

Az alulmintavételezés esetén a többségi osztályokból véletlenszerűen kiválasztunk annyit, amennyi a legkisebb méretű osztályhalmaz mérete:

```
balanced<-downSample(x=Default,y=Default$default)
summary(balanced)
```

```
##  default   student      balance           income      Class
##  No :333   No :438   Min.   :   0.0   Min.   : 4665   No :333
##  Yes:333   Yes:228   1st Qu.: 768.1   1st Qu.:19906   Yes:333
##                      Median :1349.0   Median :33449
##                      Mean   :1259.7   Mean   :32623
##                      3rd Qu.:1790.6   3rd Qu.:42801
##                      Max.   :2654.3   Max.   :66466
```

A túlmintavételezés esetén a kisebb méretű osztályt visszatevéses mintavételezéssel megnöveljük a nagyobb osztályok méretére:

```
balanced<-upSample(x=Default,y=Default$default)
summary(balanced)
```

```
##  default    student       balance         income      Class
##  No :9667   No :12801   Min.   :   0   Min.   :  772   No :9667
##  Yes:9667   Yes: 6533   1st Qu.: 793   1st Qu.:20030   Yes:9667
##                         Median :1340   Median :33575
##                         Mean   :1277   Mean   :32744
##                         3rd Qu.:1798   3rd Qu.:43439
##                         Max.   :2654   Max.   :73554
```

5.3 Egyszerű tanítási probléma

A `caret` csomag megismeréséhez tekintsük megint az egyszerű, egy szabályon alapuló tanítási problémánkat. Korábban ezt a problémát a `OneR` csomag segítségével oldottuk meg. Szerencsére a `caret` csomag is támogatja ezt az algoritmust, a `RWeka` csomagon keresztül. Készítsük elő az adathalmazunkat, azaz válasszuk szét tanuló (train) és teszt (test) adatahalmazra!

Mivel láttuk, hogy az adathalmaz kiegyensúlyozatlan a `default` változóra, ezért oda kell figyelnünk, hogy egyenletes arányban válogassunk be a tanuló és teszt adathalmazba `Yes` és `No` osztályokat. Szerencsére a `caret` csomag tartalmaz metódust az adathalmaz véletlenszerű szétválasztására, úgy, hogy az egyes osztályokat egyenlő arányban, az eredeti előfordulásuk szerint ossza szét:

```
set.seed(1)
trainIdx<-createDataPartition(Default$default, p = .75)
defaultTrain<-Default[trainIdx$Resample1,]
defaultTest<-Default[-trainIdx$Resample1,]
summary(defaultTrain)
```

```
##  default    student       balance           income
##  No :7251   No :5321   Min.   :   0.0   Min.   :  772
##  Yes: 250   Yes:2180   1st Qu.: 480.8   1st Qu.:21391
##                        Median : 817.4   Median :34579
##                        Mean   : 832.8   Mean   :33526
##                        3rd Qu.:1165.8   3rd Qu.:43744
##                        Max.   :2578.5   Max.   :73554
```

```
summary(defaultTest)
```

```
##  default    student       balance           income
##  No :2416   No :1735   Min.   :   0.0   Min.   : 2981
##  Yes:  83   Yes: 764   1st Qu.: 487.1   1st Qu.:21111
##                        Median : 839.9   Median :34412
##                        Mean   : 843.1   Mean   :33490
##                        3rd Qu.:1167.2   3rd Qu.:43894
##                        Max.   :2654.3   Max.   :69547
```

Ezután végezzük el a 1R modell tanítását a `train` függvény segítségével:

```
model<-train(default~., data=defaultTrain,
             method='OneR')
print(model)
```

```
## Single Rule Classification
##
## 7501 samples
##    3 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
```

```
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 7501, 7501, 7501, 7501, 7501, 7501, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.969185  0.3910287
```

```
summary(model)
```

```
##
## === Summary ===
##
## Correctly Classified Instances        7311               97.467  %
## Incorrectly Classified Instances       190                2.533  %
## Kappa statistic                          0.5009
## Mean absolute error                      0.0253
## Root mean squared error                  0.1592
## Relative absolute error                 39.2395 %
## Root relative squared error             88.6681 %
## Total Number of Instances             7501
##
## === Confusion Matrix ===
##
##     a    b   <-- classified as
##  7211   40 |    a = No
##   150  100 |    b = Yes
```

```
print(model$finalModel)
```

```
## balance:
##  < 1800.0018044692124    -> No
##  < 1815.3096713557939    -> Yes
##  < 1857.717695273143 -> No
##  < 1881.1532367079415    -> Yes
##  < 1890.6385128908769    -> No
##  < 1907.3190408021756    -> Yes
##  < 1940.0419901748405    -> No
##  < 2012.0916955956534    -> Yes
##  < 2034.1074763686904    -> No
##  >= 2034.1074763686904   -> Yes
## (7311/7501 instances correct)
```

Láthatjuk, hogy létrejött a tanított modellünk a `model` változóban. A `model$finalModel` változó tárolja az eredeti algoritmus által előállított modelt, jelen esetben az `RWeka` csomagban található `OneR` algoritmus modelljét. Vegyük észre, hogy a `caret` csomag a tanítás során alapértelmezésben a *boostrap* módszert alkalmaz, erre a későbbiekben visszatérünk.

Amennyiben a `train` függvénynek egyéb paramétert adunk, akkor azokat átadja a háttérben futtatott algoritmusnak. Például, a OneR algoritmus esetén láttuk, hogy az intervallumfelbontás szabályozható a `B` paraméterrel. Ebben az esetben a következőt tudjuk tenni:

```
model<-train(default~., data=defaultTrain,
             method='OneR',
             control=Weka_control(B=100))
print(model)
```

```
## Single Rule Classification
```

```
##
## 7501 samples
##    3 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 7501, 7501, 7501, 7501, 7501, 7501, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9694405  0.401864
```

```
summary(model)
```

```
##
## === Summary ===
##
## Correctly Classified Instances        7275               96.9871 %
## Incorrectly Classified Instances       226                3.0129 %
## Kappa statistic                          0.1978
## Mean absolute error                      0.0301
## Root mean squared error                  0.1736
## Relative absolute error                 46.6743 %
## Root relative squared error             96.7041 %
## Total Number of Instances             7501
##
## === Confusion Matrix ===
##
##     a    b   <-- classified as
##  7246    5 |    a = No
##   221   29 |    b = Yes
```

```
print(model$finalModel)
```

```
## balance:
##  < 2152.3128988786157    -> No
##  >= 2152.3128988786157   -> Yes
## (7275/7501 instances correct)
```

5.4 Predikció, kiértékelés

A betanított modellek természetesen a szokásos módon, a `predict` függvénnyel kiértékelhetők tetszőleges adathalmazra. Például, határozzuk meg a szenzitivitást és specifitást a OneR algoritmusra:

```
model<-train(default~., data=defaultTrain,
             method='OneR')
pred<-predict(model, defaultTrain)
predProb<-predict(model, defaultTrain, type='prob')
print(confusionMatrix(pred,defaultTrain$default,positive='Yes'))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  7211  150
```

```
##        Yes   40  100
##
##               Accuracy : 0.9747
##                 95% CI : (0.9709, 0.9781)
##    No Information Rate : 0.9667
##    P-Value [Acc > NIR] : 3.439e-05
##
##                  Kappa : 0.5009
##
##  Mcnemar's Test P-Value : 2.622e-15
##
##             Sensitivity : 0.40000
##             Specificity : 0.99448
##          Pos Pred Value : 0.71429
##          Neg Pred Value : 0.97962
##              Prevalence : 0.03333
##          Detection Rate : 0.01333
##    Detection Prevalence : 0.01866
##       Balanced Accuracy : 0.69724
##
##        'Positive' Class : Yes
##
```

Értékeljük ki egyúttal a teszt adathalmazra is a módszerünket!

```
pred<-predict(model, defaultTest)
print(confusionMatrix(pred,defaultTest$default,positive='Yes'))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  2395   51
##        Yes   21   32
##
##               Accuracy : 0.9712
##                 95% CI : (0.9639, 0.9774)
##    No Information Rate : 0.9668
##    P-Value [Acc > NIR] : 0.1190858
##
##                  Kappa : 0.4565
##
##  Mcnemar's Test P-Value : 0.0006316
##
##             Sensitivity : 0.38554
##             Specificity : 0.99131
##          Pos Pred Value : 0.60377
##          Neg Pred Value : 0.97915
##              Prevalence : 0.03321
##          Detection Rate : 0.01281
##    Detection Prevalence : 0.02121
##       Balanced Accuracy : 0.68843
##
##        'Positive' Class : Yes
##
```

Láthatjuk, hogy a szenzitivitás és specifitás értékek nagyjából megegyeznek a tanuló adathalmazokon mérttel, tehát a modell jól generalizálható. Ne felejtsük el azonban, hogy alapvetően a modell alulillesztett, hiszen a kisebbségi osztályokra nem illeszkedik kielégítően!

5.5 Kiegyensúlyozott tanítás

Végezzük el a végül a példa adathalmazunk osztályozását 1R algoritmus segítségével, kiegyensúlyozott adathalmazon!

```
set.seed(1)
trainIdx<-createDataPartition(Default$default, p = .75)
defaultTrain<-Default[trainIdx$Resample1,]
defaultTest<-Default[-trainIdx$Resample1,]

defaultTrainDS<-downSample(x=defaultTrain, y=defaultTrain$default, list=T)$x
summary(defaultTrainDS)
```

```
##  default   student      balance           income
##  No :250   No :330   Min.   :   0.0   Min.   : 8504
##  Yes:250   Yes:170   1st Qu.: 761.2   1st Qu.:20184
##                      Median :1336.2   Median :33563
##                      Mean   :1271.1   Mean   :32705
##                      3rd Qu.:1793.2   3rd Qu.:42786
##                      Max.   :2578.5   Max.   :66466
```

```
model<-train(default~.,defaultTrainDS,method='OneR')
print(model)
```

```
## Single Rule Classification
##
## 500 samples
##   3 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 500, 500, 500, 500, 500, 500, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8593344  0.718202
```

```
pred<-predict(model,defaultTrainDS)
print(confusionMatrix(pred,defaultTrainDS$default, positive = 'Yes'))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  230  32
##        Yes  20 218
##
##                Accuracy : 0.896
##                  95% CI : (0.8659, 0.9213)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
```

```
##
##                  Kappa : 0.792
##
##  Mcnemar's Test P-Value : 0.1272
##
##             Sensitivity : 0.8720
##             Specificity : 0.9200
##          Pos Pred Value : 0.9160
##          Neg Pred Value : 0.8779
##              Prevalence : 0.5000
##          Detection Rate : 0.4360
##    Detection Prevalence : 0.4760
##       Balanced Accuracy : 0.8960
##
##        'Positive' Class : Yes
##
```

```
pred<-predict(model,defaultTest)
print(confusionMatrix(pred,defaultTest$default, positive = 'Yes'))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  2153   17
##        Yes  263   66
##
##                Accuracy : 0.888
##                  95% CI : (0.8749, 0.9001)
##     No Information Rate : 0.9668
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2823
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.79518
##             Specificity : 0.89114
##          Pos Pred Value : 0.20061
##          Neg Pred Value : 0.99217
##              Prevalence : 0.03321
##          Detection Rate : 0.02641
##    Detection Prevalence : 0.13165
##       Balanced Accuracy : 0.84316
##
##        'Positive' Class : Yes
##
```

Láthatjuk, hogy ugyan a modell némileg alulillesztett (89.6% pontosság), de jól generalizálható, hiszen a metrikákat a teszt adathalmazon is.

5.6 Dummy változók

Tekintsük az alábbi egyszerű adathalmazt, amely megadja az időjárástól függően, hogy menjünk-e teniszezni:

```
df<-read.csv(text='Outlook,Temperature,Humidity,Wind,Play Tennis
Sunny,Hot,High,Weak,No
Sunny,Hot,High,Strong,No
Overcast,Hot,High,Weak,Yes
Rain,Mild,High,Weak,Yes
Rain,Cool,Normal,Weak,Yes
Rain,Cool,Normal,Strong,No
Overcast,Cool,Normal,Strong,Yes
Sunny,Mild,High,Weak,No
Sunny,Cool,Normal,Weak,Yes
Rain,Mild,Normal,Weak,Yes
Sunny,Mild,Normal,Strong,Yes
Overcast,Mild,High,Strong,Yes
Overcast,Hot,Normal,Weak,Yes
Rain,Mild,High,Strong,No',stringsAsFactor=T)
```

Ha erre lefuttatjuk a OneR osztályozót, érdekes eredményeket kapunk:

```
m<-train(Play.Tennis~.,df,method='OneR')
m$finalModel
```

```
## OutlookSunny:
##  < 0.5   -> Yes
##  >= 0.5  -> No
## (10/14 instances correct)
```

Vajon mi az a változó, amely szerint az osztályozás történt (OutlookSunny)? Láthatjuk, hogy a döntési szabály ennek a változónak a 0.5 értéktől teszi függővé a döntést. Ez nem teljesen az, amire számítottunk. És valóban, ami történt, hogy a caret létrehozott ún. dummy változókat. Ezeke kézzel is létrehozhatjuk:

```
dummy<-dummyVars(~.,df)
predict(dummy,df)
```

```
##    Outlook.Overcast Outlook.Rain Outlook.Sunny Temperature.Cool Temperature.Hot
## 1                 0            0             1                0               1
## 2                 0            0             1                0               1
## 3                 1            0             0                0               1
## 4                 0            1             0                0               0
## 5                 0            1             0                1               0
## 6                 0            1             0                1               0
## 7                 1            0             0                1               0
## 8                 0            0             1                0               0
## 9                 0            0             1                1               0
## 10                0            1             0                0               0
## 11                0            0             1                0               0
## 12                1            0             0                0               0
## 13                1            0             0                0               1
## 14                0            1             0                0               0
##    Temperature.Mild Humidity.High Humidity.Normal Wind.Strong Wind.Weak
## 1                 0             1               0           0         1
## 2                 0             1               0           1         0
## 3                 0             1               0           0         1
## 4                 1             1               0           0         1
## 5                 0             0               1           0         1
## 6                 0             0               1           1         0
## 7                 0             0               1           1         0
```

```
## 8                1             1               0           0         1
## 9                0             0               1           0         1
## 10               1             0               1           0         1
## 11               1             0               1           1         0
## 12               1             1               0           1         0
## 13               0             0               1           0         1
## 14               1             1               0           1         0
##    Play.Tennis.No Play.Tennis.Yes
## 1               1                0
## 2               1                0
## 3               0                1
## 4               0                1
## 5               0                1
## 6               1                0
## 7               0                1
## 8               1                0
## 9               0                1
## 10              0                1
## 11              0                1
## 12              0                1
## 13              0                1
## 14              1                0
```

A valóságban ez még csak az ún. one-hot coding. Valódi dummy változókhoz meg kell adnunk a `fullRank` paramétert:

```
dummy<-dummyVars(~.,df,fullRank=T)
predict(dummy,df)
```

```
##    Outlook.Rain Outlook.Sunny Temperature.Hot Temperature.Mild Humidity.Normal
## 1             0             1               1                 0                0
## 2             0             1               1                 0                0
## 3             0             0               1                 0                0
## 4             1             0               0                 1                0
## 5             1             0               0                 0                1
## 6             1             0               0                 0                1
## 7             0             0               0                 0                1
## 8             0             1               0                 1                0
## 9             0             1               0                 0                1
## 10            1             0               0                 1                1
## 11            0             1               0                 1                1
## 12            0             0               0                 1                0
## 13            0             0               1                 0                1
## 14            1             0               0                 1                0
##    Wind.Weak Play.Tennis.Yes
## 1          1               0
## 2          0               0
## 3          1               1
## 4          1               1
## 5          1               1
## 6          0               0
## 7          0               1
## 8          1               0
## 9          1               1
## 10         1               1
```

```
## 11         0               1
## 12         0               1
## 13         1               1
## 14         0               0
```

Ekkor láthatjuk, hogy minden változóból az egyik értéknek az oszlopa hiányzik. Ennek köszönthetően a kimeneti adathalmaz oszlopai nem lesznek összefüggőek.

A caret `train` függvény a kategorikus változókat (faktorokat) automatikusan dummy változókká alakítja, ha a formula interfészt használjuk (lásd. előbb). Ezzel szemben a klasszikus x,y interfész az adatokat az eredeti formájában továbbítja az algoritmusok részére:

```
m<-train(y=df$Play.Tennis,x=df[,-ncol(df)],method='OneR')
m$finalModel
```

```
## Outlook:
##  Overcast    -> Yes
##  Rain    -> Yes
##  Sunny   -> No
## (10/14 instances correct)
```

5.7 Model paraméterek

A támogatott modellekről információt itt találunk, vagy használhatjuk a `getModelInfo` vagy `modelLookup` függvényt. Például:

```
getModelInfo('OneR')
```

```
## $OneR
## $OneR$label
## [1] "Single Rule Classification"
##
## $OneR$library
## [1] "RWeka"
##
## $OneR$loop
## NULL
##
## $OneR$type
## [1] "Classification"
##
## $OneR$parameters
##   parameter     class label
## 1 parameter character  none
##
## $OneR$grid
## function(x, y, len = NULL, search = "grid")
##                     data.frame(parameter = "none")
##
## $OneR$fit
## function(x, y, wts, param, lev, last, classProbs, ...) {
##                     dat <- if(is.data.frame(x)) x else as.data.frame(x, stringsAsFactors = TRUE)
##                     dat$.outcome <- y
##                     theDots <- list(...)
##
##                     modelArgs <- c(list(formula = as.formula(".outcome ~ ."),
##                                         data = dat),
```

```
##                                     theDots)
##
##                     out <- do.call(RWeka::OneR, modelArgs)
##                     out
##                     }
##
## $OneR$predict
## function(modelFit, newdata, submodels = NULL) {
##                     if(!is.data.frame(newdata)) newdata <- as.data.frame(newdata, stringsAsFactors =
##                     predict(modelFit, newdata)
##                     }
##
## $OneR$prob
## function(modelFit, newdata, submodels = NULL) {
##                     if(!is.data.frame(newdata)) newdata <- as.data.frame(newdata, stringsAsFactors =
##                     predict(modelFit, newdata, type = "probability")
##                     }
##
## $OneR$levels
## function(x) x$obsLevels
##
## $OneR$predictors
## function(x, ...) predictors(x$terms)
##
## $OneR$tags
## [1] "Rule-Based Model"           "Implicit Feature Selection"
##
## $OneR$sort
## function(x) x
```

```
modelLookup('OneR')
```

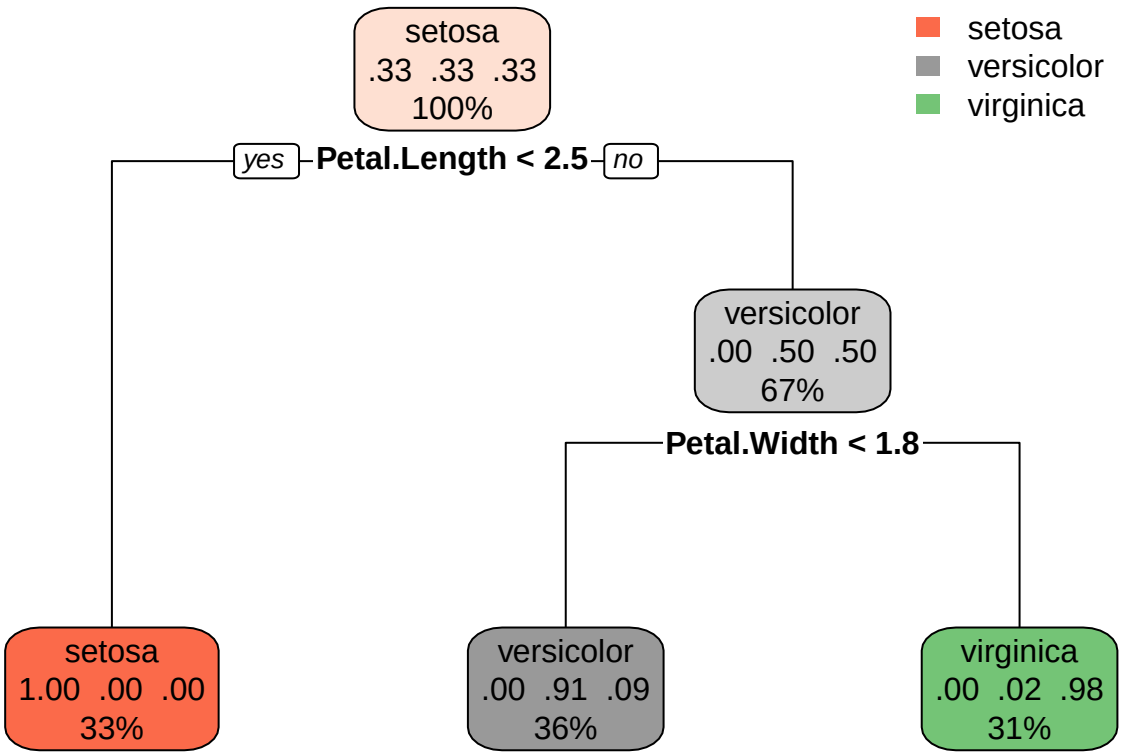
```
##   model parameter label forReg forClass probModel
## 1  OneR parameter  none  FALSE     TRUE      TRUE
```

6 Döntési fák

```
# Használt csomagok
library(rpart)
library(caret)
library(rpart.plot)
```

A döntési fák nagyon egyszerű struktúrák adatelemzési feladatok megoldására. A döntési fák tekinthetőek tulajdonképpen a OneR osztályozók rekurzív alkalmazásának is, hiszen egy gyökérelemtől lefelé haladva az adathalmazt két részre osztják, majd a részeket további feltételek mentén újra két részre osztják és így tovább. Tekintsük például az alábbi döntési fát az iris adathalmazra:

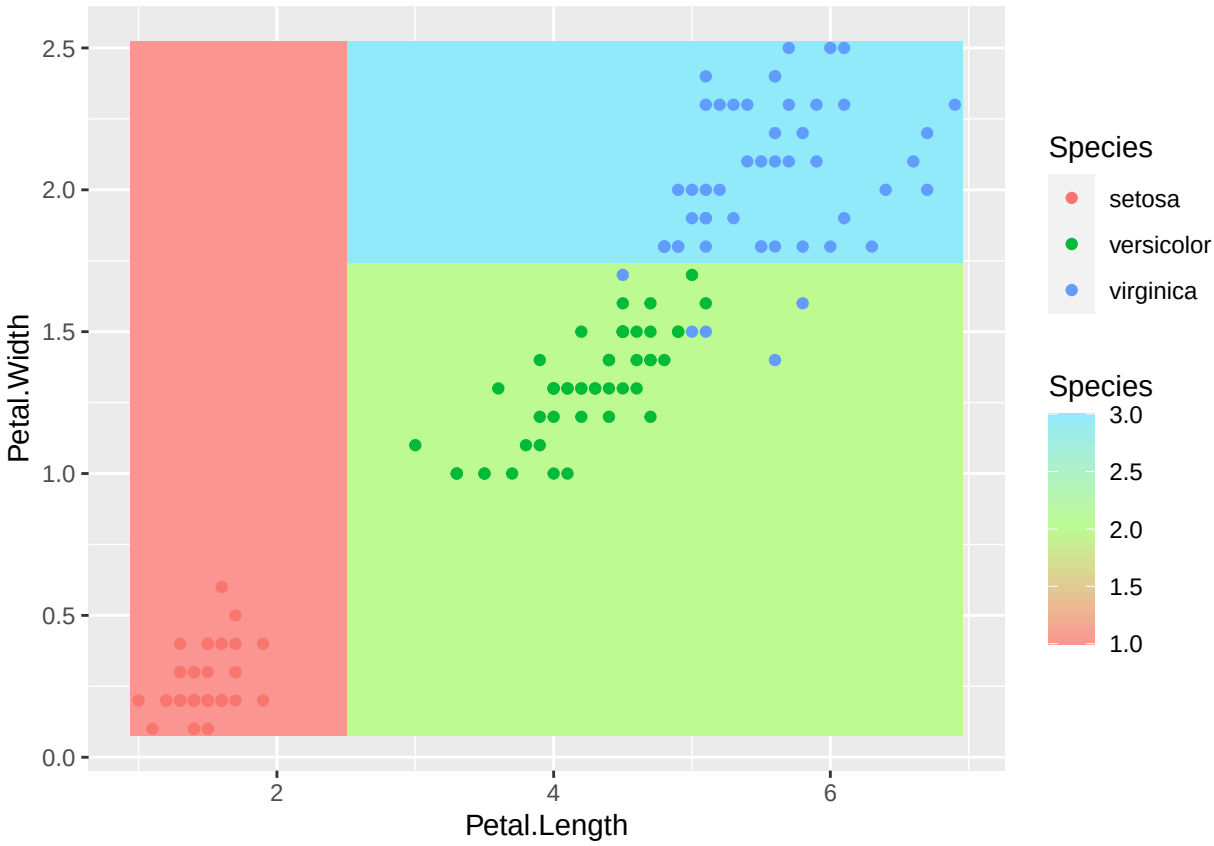
```
df<-iris
model<-rpart(Species~Petal.Length+Petal.Width,df)
rpart.plot(model)
```



A döntési fák legfontosabb részei a következők:

- gyökér csomópont
- levél csomópont
- szülő- és gyermek csomópontok

A fák nagyon fontos paramétere az ún. *mélység*. A fa mélysége azt jelöli, hogy a levél csomópontok a fa hányadik szintjén helyezkednek el. A szintek számozása a gyökér csomóponttól, nullával kezdődik. A fent látható fa 2 szintes. A döntési fa döntési határai párhuzamosak a tengelyekkel:



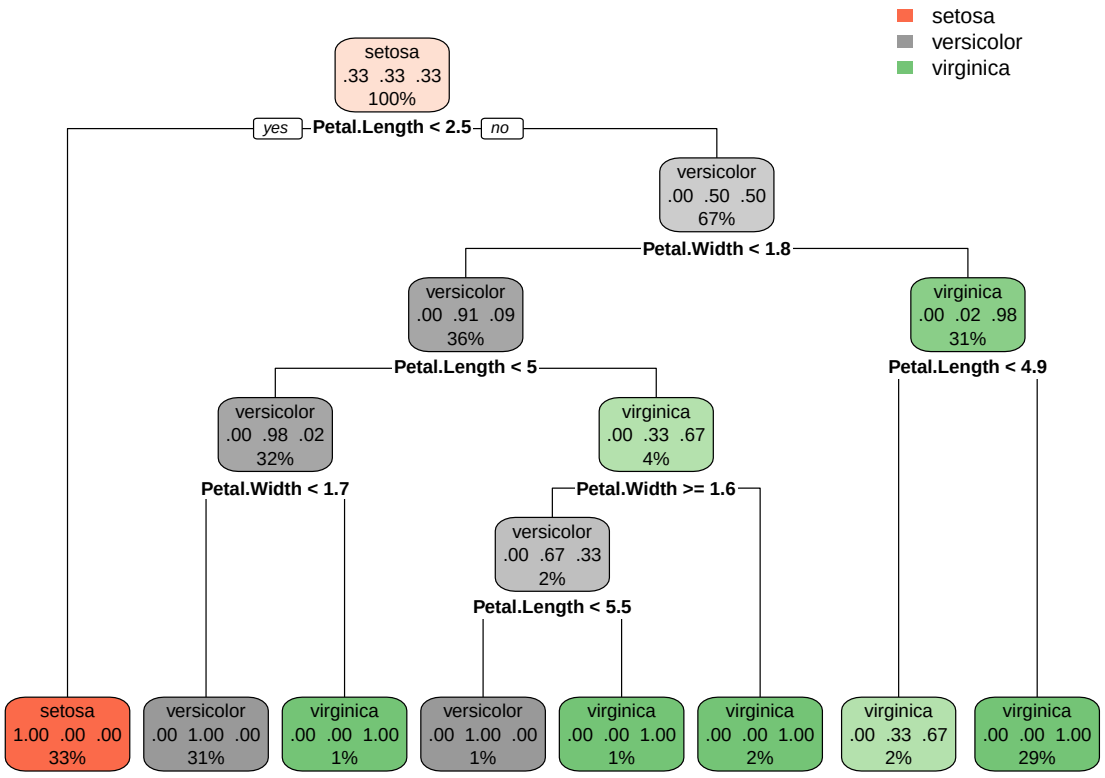
6.1 A döntési fa felépítése

A döntési fákat tipikusan mohó algoritmussal építjük, azaz valamilyen megállási feltételig (például a fa egy adott mélységet elért, vagy már túl kevés attribútum, esetleg adatsor maradt) az egyes csomópontokat valamilyen feltétel mentén kettéosztjuk. Az, hogy a kettosztáshoz milyen attribútumot és értéket válasszunk, számos megoldás ismert, a két legismertebb:

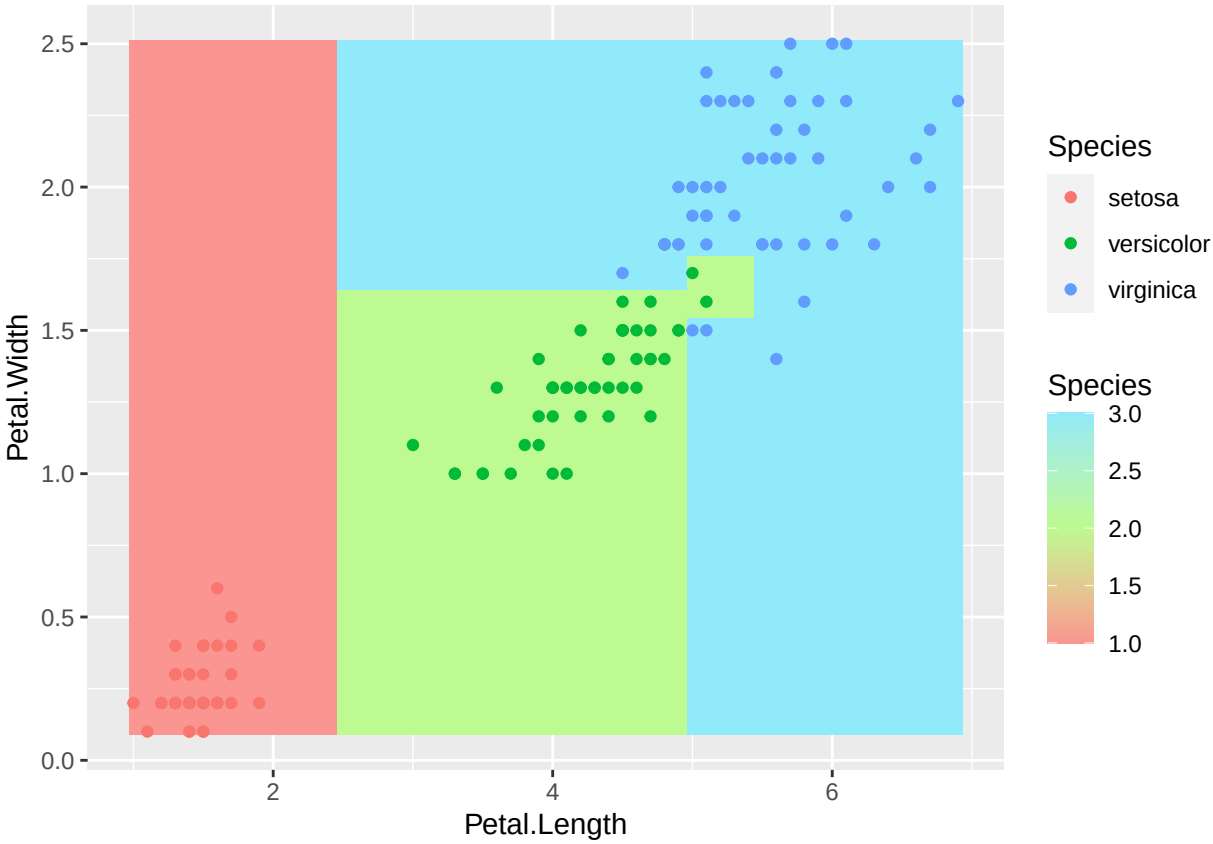
- Gini-index (CART),
- információ nyereség (ID3, C4.5).

6.2 Túlillesztés, regularizáció

A döntési fák esetében könnyen megjelenhet a túlillesztés jelensége, hiszen képesek vagyunk mindaddig növelni a fa mélységét, amíg minden egyes pontra külön szabályt tudunk alkotni. Például, a következő ábrán látható az `iris` adathalmazra illesztett 5 mélységű fa, és a hozzá tartozó döntési határok ábrázolása.



Az ehhez tartozó döntési határokat a következő ábrán láthatjuk:



A túlillesztés megelőlzésére számos módszert ismerünk, ezeket általában kombinálva használják az egyes

megvalósítások. Ilyen megoldások lehetnek:

- **korai leállás**: A fa építése során nem fejezzük be a fa építését, hanem idő előtt leáll az algoritmus. Ez kombinálható valamilyen metrika figyelésével is, amely jelzi a túlillesztettség mértékét.
- **ágak levágása (pruning)**: Egy teljesen felépített fa ágainak a levágása utólag a túlillesztettség csökkentésére.
- **validáció**: Validációs módszerek alkalmazása a megfelelő mélység és komplexitás meghatározása érdekében.
- **faépítési szabályok**: Bizonyos korlátok bevezetése a fa építése során. Ilyen lehet például a levél vagy egyéb csomópontok minimális mérete (minimum hány adatsort kell tartalmaznia), a komplexitási együttható megszabása, maximális mélység megszabása, stb.

6.3 Egyszerű tanítási példa

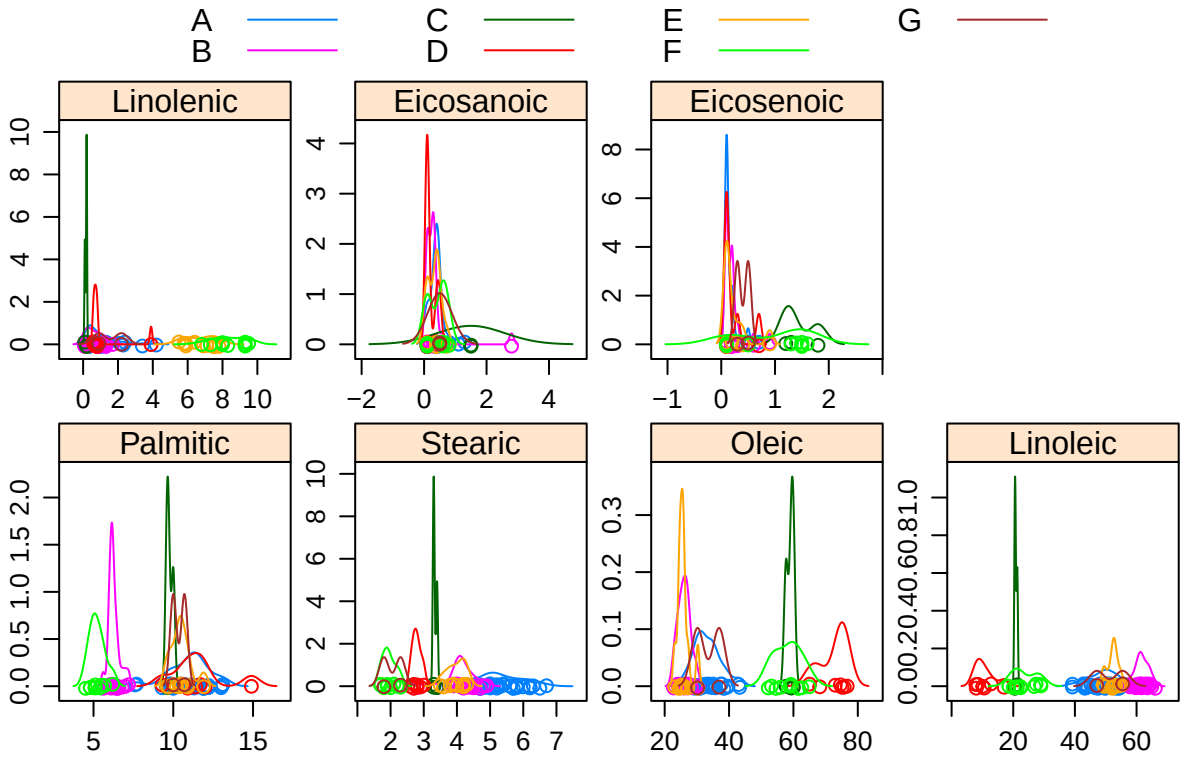
Tekintsük az `oil` adathalmazt, amely különböző étkezési olajok zsírsavtartalmát tartalmazza. Az egyes étkezési olajok fajtáját a `?oil` oldalon találjuk meg.

```
set.seed(1)

data(oil)
df<-cbind(fattyAcids,oilType)
summary(df)
```

```
##     Palmitic        Stearic          Oleic          Linoleic
##  Min.   : 4.50   Min.   :1.700   Min.   :22.80   Min.   : 7.90
##  1st Qu.: 6.20   1st Qu.:3.475   1st Qu.:26.30   1st Qu.:43.10
##  Median : 9.85   Median :4.200   Median :30.70   Median :50.80
##  Mean   : 9.04   Mean   :4.200   Mean   :36.73   Mean   :46.49
##  3rd Qu.:11.12   3rd Qu.:5.000   3rd Qu.:38.62   3rd Qu.:58.08
##  Max.   :14.90   Max.   :6.700   Max.   :76.70   Max.   :66.10
##
##    Linolenic       Eicosanoic      Eicosenoic     oilType
##  Min.   :0.100   Min.   :0.100   Min.   :0.1000   A:37
##  1st Qu.:0.375   1st Qu.:0.100   1st Qu.:0.1000   B:26
##  Median :0.800   Median :0.400   Median :0.1000   C: 3
##  Mean   :2.272   Mean   :0.399   Mean   :0.3115   D: 7
##  3rd Qu.:2.650   3rd Qu.:0.400   3rd Qu.:0.3000   E:11
##  Max.   :9.500   Max.   :2.800   Max.   :1.8000   F:10
##                                                   G: 2
```

Ábrázoljuk az adathalmazhoz a jellemzők eloszlását!

```
p<-featurePlot(x = df[,-ncol(df)], y = df$oilType, plot='density',
               scales = list(x = list(relation="free"),
                             y = list(relation="free")),
               auto.key=list(columns=4))
plot(p)
```



Válogassuk szét az adatokat tanító és teszt adathalmazra, hogy a tanítási folyamat eredményét ki tudjuk értékelni.

```
trainIdx<-createDataPartition(df$oilType, p=0.5, list=F)
trainDf<-df[trainIdx,]
testDf<-df[-trainIdx,]

summary(trainDf)
```

```
##     Palmitic         Stearic          Oleic          Linoleic
##  Min.   : 4.500   Min.   :1.700   Min.   :23.10   Min.   : 8.00
##  1st Qu.: 6.275   1st Qu.:3.325   1st Qu.:26.43   1st Qu.:43.15
##  Median :10.350   Median :4.200   Median :30.05   Median :50.70
##  Mean   : 9.362   Mean   :4.240   Mean   :36.65   Mean   :46.31
##  3rd Qu.:11.475   3rd Qu.:5.175   3rd Qu.:38.15   3rd Qu.:57.23
##  Max.   :14.900   Max.   :6.700   Max.   :75.80   Max.   :66.10
##
##    Linolenic       Eicosanoic      Eicosenoic    oilType
##  Min.   :0.100   Min.   :0.100   Min.   :0.100   A:19
##  1st Qu.:0.300   1st Qu.:0.100   1st Qu.:0.100   B:13
##  Median :0.800   Median :0.400   Median :0.100   C: 2
##  Mean   :2.166   Mean   :0.378   Mean   :0.286   D: 4
##  3rd Qu.:3.075   3rd Qu.:0.400   3rd Qu.:0.200   E: 6
##  Max.   :8.300   Max.   :1.500   Max.   :1.800   F: 5
##                                                  G: 1
```

```
summary(testDf)
```

```
##     Palmitic         Stearic          Oleic          Linoleic
##  Min.   : 4.800   Min.   :1.700   Min.   :22.80   Min.   : 7.90
```

```
##  1st Qu.: 6.200   1st Qu.:3.800   1st Qu.:26.18   1st Qu.:43.30
##  Median : 9.700   Median :4.200   Median :31.40   Median :51.05
##  Mean   : 8.689   Mean   :4.157   Mean   :36.81   Mean   :46.68
##  3rd Qu.:10.650   3rd Qu.:5.000   3rd Qu.:38.90   3rd Qu.:58.60
##  Max.   :12.200   Max.   :6.200   Max.   :76.70   Max.   :64.40
##
##    Linolenic       Eicosanoic       Eicosenoic     oilType
##  Min.   :0.100   Min.   :0.1000   Min.   :0.1000   A:18
##  1st Qu.:0.400   1st Qu.:0.1000   1st Qu.:0.1000   B:13
##  Median :0.800   Median :0.4000   Median :0.1000   C: 1
##  Mean   :2.387   Mean   :0.4217   Mean   :0.3391   D: 3
##  3rd Qu.:2.375   3rd Qu.:0.4000   3rd Qu.:0.3000   E: 5
##  Max.   :9.500   Max.   :2.8000   Max.   :1.6000   F: 5
##                                                    G: 1
```

Illesszünk egy egyszerű döntési fát az adathalmazra, amely során a `caret` csomagra bízzuk a paraméterek megválasztását:

```
model<-train(oilType~., trainDf, method='rpart2')
```

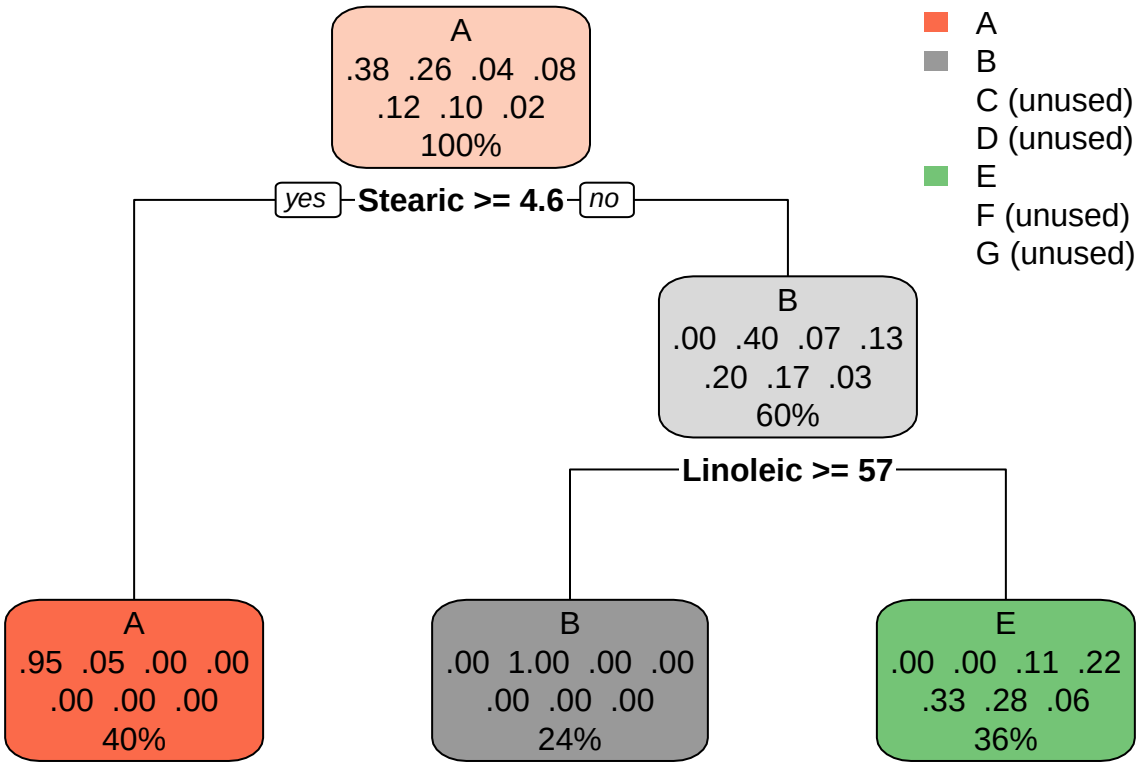
```
## note: only 2 possible values of the max tree depth from the initial fit.
##  Truncating the grid to 2 .
```

```
print(model)
```

```
## CART
##
## 50 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 50, 50, 50, 50, 50, 50, ...
## Resampling results across tuning parameters:
##
##   maxdepth  Accuracy   Kappa
##   1         0.5505482  0.3675399
##   2         0.6367998  0.5213543
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was maxdepth = 2.
```

Ábrázoljuk a döntési fát!

```
rpart.plot(model$finalModel)
```




```
print(model$finalModel)
```

```
## n= 50
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 50 31 A (0.38 0.26 0.04 0.08 0.12 0.1 0.02)
##   2) Stearic>=4.55 20  1 A (0.95 0.05 0 0 0 0 0) *
##   3) Stearic< 4.55 30 18 B (0 0.4 0.067 0.13 0.2 0.17 0.033)
##     6) Linoleic>=56.65 12  0 B (0 1 0 0 0 0 0) *
##     7) Linoleic< 56.65 18 12 E (0 0 0.11 0.22 0.33 0.28 0.056) *
```

Célszerű áttekinteni a tanító és teszt adathalmaz esetében a tévesztési mátrixot:

```
pred<-predict(model,trainDf)
print(confusionMatrix(pred,trainDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 19  1  0  0  0  0  0
##          B  0 12  0  0  0  0  0
##          C  0  0  0  0  0  0  0
##          D  0  0  0  0  0  0  0
##          E  0  0  2  4  6  5  1
##          F  0  0  0  0  0  0  0
##          G  0  0  0  0  0  0  0
##
## Overall Statistics
```

```
##
##                Accuracy : 0.74
##                  95% CI : (0.5966, 0.8537)
##     No Information Rate : 0.38
##     P-Value [Acc > NIR] : 2.526e-07
##
##                   Kappa : 0.6498
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            1.0000   0.9231     0.00     0.00   1.0000      0.0
## Specificity            0.9677   1.0000     1.00     1.00   0.7273      1.0
## Pos Pred Value         0.9500   1.0000      NaN      NaN   0.3333      NaN
## Neg Pred Value         1.0000   0.9737     0.96     0.92   1.0000      0.9
## Prevalence             0.3800   0.2600     0.04     0.08   0.1200      0.1
## Detection Rate         0.3800   0.2400     0.00     0.00   0.1200      0.0
## Detection Prevalence   0.4000   0.2400     0.00     0.00   0.3600      0.0
## Balanced Accuracy      0.9839   0.9615     0.50     0.50   0.8636      0.5
##                      Class: G
## Sensitivity              0.00
## Specificity              1.00
## Pos Pred Value            NaN
## Neg Pred Value           0.98
## Prevalence               0.02
## Detection Rate           0.00
## Detection Prevalence     0.00
## Balanced Accuracy        0.50
```

```
print('--------------')
```

```
## [1] "--------------"
```

```
pred<-predict(model,testDf)
print(confusionMatrix(pred,testDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 16  2  0  0  0  0  0
##          B  0 11  0  0  0  0  0
##          C  0  0  0  0  0  0  0
##          D  0  0  0  0  0  0  0
##          E  2  0  1  3  5  5  1
##          F  0  0  0  0  0  0  0
##          G  0  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.6957
##                  95% CI : (0.5425, 0.8226)
##     No Information Rate : 0.3913
```

```
##     P-Value [Acc > NIR] : 2.851e-05
##
##                   Kappa : 0.5882
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.8889   0.8462  0.00000  0.00000   1.0000   0.0000
## Specificity            0.9286   1.0000  1.00000  1.00000   0.7073   1.0000
## Pos Pred Value         0.8889   1.0000      NaN      NaN   0.2941      NaN
## Neg Pred Value         0.9286   0.9429  0.97826  0.93478   1.0000   0.8913
## Prevalence             0.3913   0.2826  0.02174  0.06522   0.1087   0.1087
## Detection Rate         0.3478   0.2391  0.00000  0.00000   0.1087   0.0000
## Detection Prevalence   0.3913   0.2391  0.00000  0.00000   0.3696   0.0000
## Balanced Accuracy      0.9087   0.9231  0.50000  0.50000   0.8537   0.5000
##                      Class: G
## Sensitivity           0.00000
## Specificity           1.00000
## Pos Pred Value            NaN
## Neg Pred Value        0.97826
## Prevalence            0.02174
## Detection Rate        0.00000
## Detection Prevalence  0.00000
## Balanced Accuracy     0.50000
```

Láthatjuk, hogy az osztályozás eredménye nem különösebben pontos, ráadásul a teszthalmaz esetén a számos osztályra sosem szavazott a model.

6.4 Javított tanítási példa

Érdemes észrevenni, hogy az osztályozó teljesítménye alapesetben sem kielégítő, hiszen pár osztályt leszámítva a szenzitivitások ugyan magasak, de általában a pontosság nem kielégítő. Fontos kiemelni ugyanakkor, hogy számos eseteben elvárt, hogy bizonyos osztályok szenzitivitásának csökkentésével más osztályok szenzitivitását javítsuk. Az első lépésünk tehát, hogy az eredetileg kiegyensúlyozatlan adathalmazt kiegyensúlyozzuk. Erre jelen esetben a túlmintavételezés alkalmas:

```
trainDfU<-upSample(x = trainDf, y=trainDf$oilType, list=T)$x
summary(trainDfU)
```

```
##     Palmitic         Stearic          Oleic          Linoleic
##  Min.   : 4.500   Min.   :1.700   Min.   :23.10   Min.   : 8.00
##  1st Qu.: 6.600   1st Qu.:2.200   1st Qu.:27.20   1st Qu.:20.90
##  Median :10.500   Median :3.300   Median :31.70   Median :49.50
##  Mean   : 9.443   Mean   :3.367   Mean   :43.49   Mean   :39.09
##  3rd Qu.:10.900   3rd Qu.:4.200   3rd Qu.:60.00   3rd Qu.:55.50
##  Max.   :14.900   Max.   :6.700   Max.   :75.80   Max.   :66.10
##
##    Linolenic       Eicosanoic       Eicosenoic     oilType
##  Min.   :0.100   Min.   :0.1000   Min.   :0.1000   A:19
##  1st Qu.:0.300   1st Qu.:0.1000   1st Qu.:0.1000   B:19
##  Median :0.900   Median :0.4000   Median :0.2000   C:19
##  Mean   :2.544   Mean   :0.5105   Mean   :0.4729   D:19
##  3rd Qu.:5.900   3rd Qu.:0.6000   3rd Qu.:0.6000   E:19
```

```
##  Max.   :8.300   Max.   :1.5000   Max.   :1.8000   F:19
##                                                    G:19
```

Futtassuk újra le az osztályozási algoritmust, és számítsuk ki egyből a keveredési mátrixot a tanító adatokra:

```
model<-train(oilType~., trainDfU, method='rpart2')
```

```
## note: only 1 possible values of the max tree depth from the initial fit.
##  Truncating the grid to 1 .
```

```
print(model)
```

```
## CART
##
## 133 samples
##   7 predictor
##   7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 133, 133, 133, 133, 133, 133, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9522379  0.9438053
##
## Tuning parameter 'maxdepth' was held constant at a value of 6
```

```
pred<-predict(model,trainDf)
print(confusionMatrix(pred,trainDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 19  0  0  0  0  0  0
##          B  0 13  0  0  0  0  0
##          C  0  0  2  0  0  0  0
##          D  0  0  0  4  0  0  0
##          E  0  0  0  0  6  0  0
##          F  0  0  0  0  0  5  0
##          G  0  0  0  0  0  0  1
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9289, 1)
##     No Information Rate : 0.38
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
```

```
## Sensitivity              1.00     1.00     1.00     1.00     1.00      1.0
## Specificity              1.00     1.00     1.00     1.00     1.00      1.0
## Pos Pred Value           1.00     1.00     1.00     1.00     1.00      1.0
## Neg Pred Value           1.00     1.00     1.00     1.00     1.00      1.0
## Prevalence               0.38     0.26     0.04     0.08     0.12      0.1
## Detection Rate           0.38     0.26     0.04     0.08     0.12      0.1
## Detection Prevalence     0.38     0.26     0.04     0.08     0.12      0.1
## Balanced Accuracy        1.00     1.00     1.00     1.00     1.00      1.0
##                       Class: G
## Sensitivity              1.00
## Specificity              1.00
## Pos Pred Value           1.00
## Neg Pred Value           1.00
## Prevalence               0.02
## Detection Rate           0.02
## Detection Prevalence     0.02
## Balanced Accuracy        1.00
```

Láthatjuk, hogy a tanító adathalmazra a pontosságot jelentősen sikerült javítani. Tekintsük a teszthalmazra a keveredési mátrixot:

```
pred<-predict(model,testDf)
print(confusionMatrix(pred,testDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 17  0  0  0  0  0  0
##          B  0 13  0  0  0  0  0
##          C  0  0  1  1  0  0  0
##          D  0  0  0  2  0  0  0
##          E  1  0  0  0  5  0  0
##          F  0  0  0  0  0  5  0
##          G  0  0  0  0  0  0  1
##
## Overall Statistics
##
##                Accuracy : 0.9565
##                  95% CI : (0.8516, 0.9947)
##     No Information Rate : 0.3913
##     P-Value [Acc > NIR] : 4.643e-16
##
##                   Kappa : 0.9417
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.9444   1.0000  1.00000  0.66667   1.0000   1.0000
## Specificity            1.0000   1.0000  0.97778  1.00000   0.9756   1.0000
## Pos Pred Value         1.0000   1.0000  0.50000  1.00000   0.8333   1.0000
## Neg Pred Value         0.9655   1.0000  1.00000  0.97727   1.0000   1.0000
## Prevalence             0.3913   0.2826  0.02174  0.06522   0.1087   0.1087
```

```
## Detection Rate         0.3696   0.2826  0.02174  0.04348   0.1087   0.1087
## Detection Prevalence   0.3696   0.2826  0.04348  0.04348   0.1304   0.1087
## Balanced Accuracy      0.9722   1.0000  0.98889  0.83333   0.9878   1.0000
##                      Class: G
## Sensitivity           1.00000
## Specificity           1.00000
## Pos Pred Value        1.00000
## Neg Pred Value        1.00000
## Prevalence            0.02174
## Detection Rate        0.02174
## Detection Prevalence  0.02174
## Balanced Accuracy     1.00000
```

Láthatjuk, hogy az osztályozó ebben az esetben is meglepően jó eredményeket szolgáltat.

Felmerül a kérdés, hogy miként tudunk új adatsorokat prediktálni. Például, a következő zsírsavtartalommal rendelkező olaj milyen típusú? * Palmitic: 5.2 * Stearic: 2.0 * Oleic: 54.2 * Linoleic: 21.0 * Linolenic: 9.0 * Eicosanoic: 0.6 * Eicosenoic: 1.4

```
newDataVector<-c(5.2,2.0,54.2,21.0,9.0,0.6,1.4)
newData<-as.data.frame(newDataVector,row.names = colnames(fattyAcids))
newData<-t(newData)
predict(model,newData,type='prob')
```

```
##                A B C D E F G
## newDataVector 0 0 0 0 0 1 0
```

6.5 A faépítés paraméterei

A `caret` csomag számos döntési fa megoldást támogat, ugyanakkor itt a CART fákkal foglalkozunk. Ennek két implementációja található meg a csomagban, `rpart` és `rpart2` néven. Míg az előbbi egy ún. komplexitást jelző paramétert használ a túlillesztés elkerülésére, addig az utóbbi közvetlenül a fa mélységét képes szabályozni. Ismételten megjegyezzük, hogy a modellek és azok paramétereit itt találjuk.

A modellválasztás során megadhatjuk a vezérelt paraméter lehetséges értékeit. Például, egy maximum 10 szintes fa építését a következőképpen tudjuk megtenni:

```
model<-train(oilType~., trainDfU,
             method='rpart2',
             tuneGrid=data.frame(maxdepth=1:10))
print(model)
```

```
## CART
##
## 133 samples
##   7 predictor
##   7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 133, 133, 133, 133, 133, 133, ...
## Resampling results across tuning parameters:
##
##   maxdepth  Accuracy   Kappa
##    1        0.2524983  0.1518144
##    2        0.3832082  0.2929629
##    3        0.5096322  0.4349230
```

```
##    4        0.6630249  0.6078617
##    5        0.8156814  0.7846015
##    6        0.9560224  0.9482919
##    7        0.9560224  0.9482919
##    8        0.9560224  0.9482919
##    9        0.9560224  0.9482919
##   10        0.9560224  0.9482919
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was maxdepth = 6.
```

Persze egyáltalán nem biztos, hogy az algoritmus létrehoz egy adott szinttel rendelkező fát, hiszen egyéb megállási feltételekkel is dolgozik az algoritmus. Ezeket a paramétereket a

```
?rpart.control
```

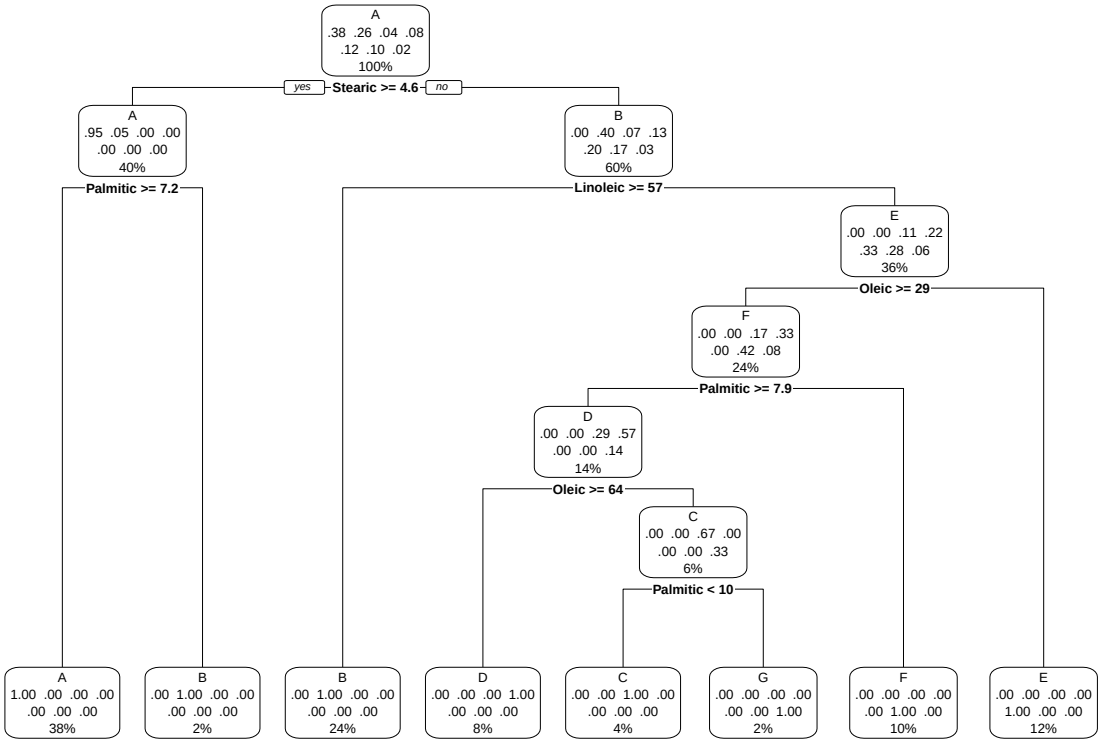
paranccsal tudjuk megtekinteni. Például:

```
model<-train(oilType~., trainDf,
             method='rpart2',
             control=rpart.control(
               minsplit=1,
               minbucket=1
             ),
             tuneGrid=data.frame(maxdepth=10))
print(model)
```

```
## CART
##
## 50 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 50, 50, 50, 50, 50, 50, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8416273  0.7880652
##
## Tuning parameter 'maxdepth' was held constant at a value of 10
```

```
rpart.plot(model$finalModel)
```

```
## Warning: All boxes will be white (the box.palette argument will be ignored) because
## the number of classes in the response 7 is greater than length(box.palette) 6.
## To silence this warning use box.palette=0 or trace=-1.
```

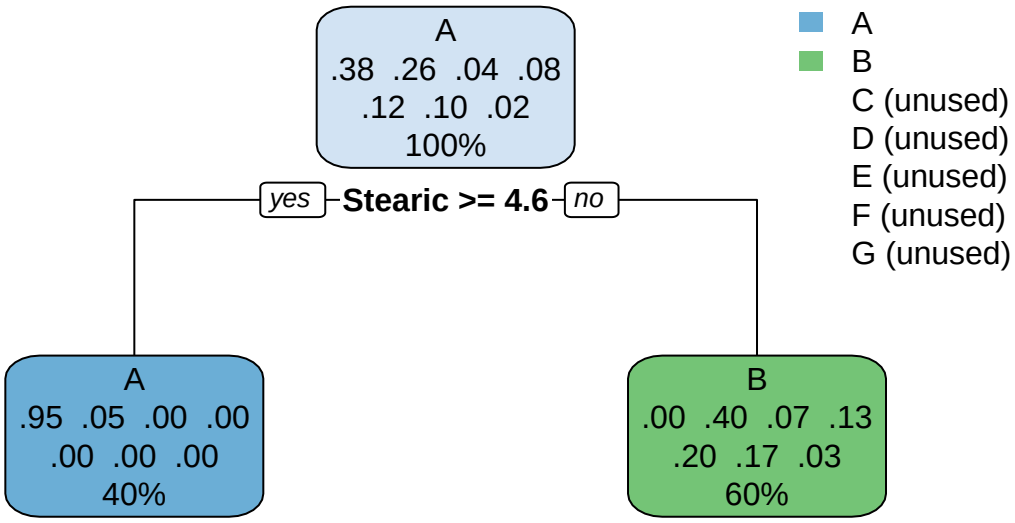


Ha esetleg tudjuk, hogy hány szintes fát szeretnénk építeni, és nem akarunk hiperparamétereket választani, akkor a validációs módszert ki is kapcsolhatjuk:

```
model<-train(oilType~., trainDf,
             method='rpart2',
             trControl=trainControl(
               method="none"
             ),
             tuneGrid=data.frame(maxdepth=1))
print(model)
```

```
## CART
##
## 50 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: None
```

```
rpart.plot(model$finalModel)
```



7 K legközelebbi szomszéd osztályozók

```
# Használt csomagok
library(rpart)
library(caret)
library(rpart.plot)
```

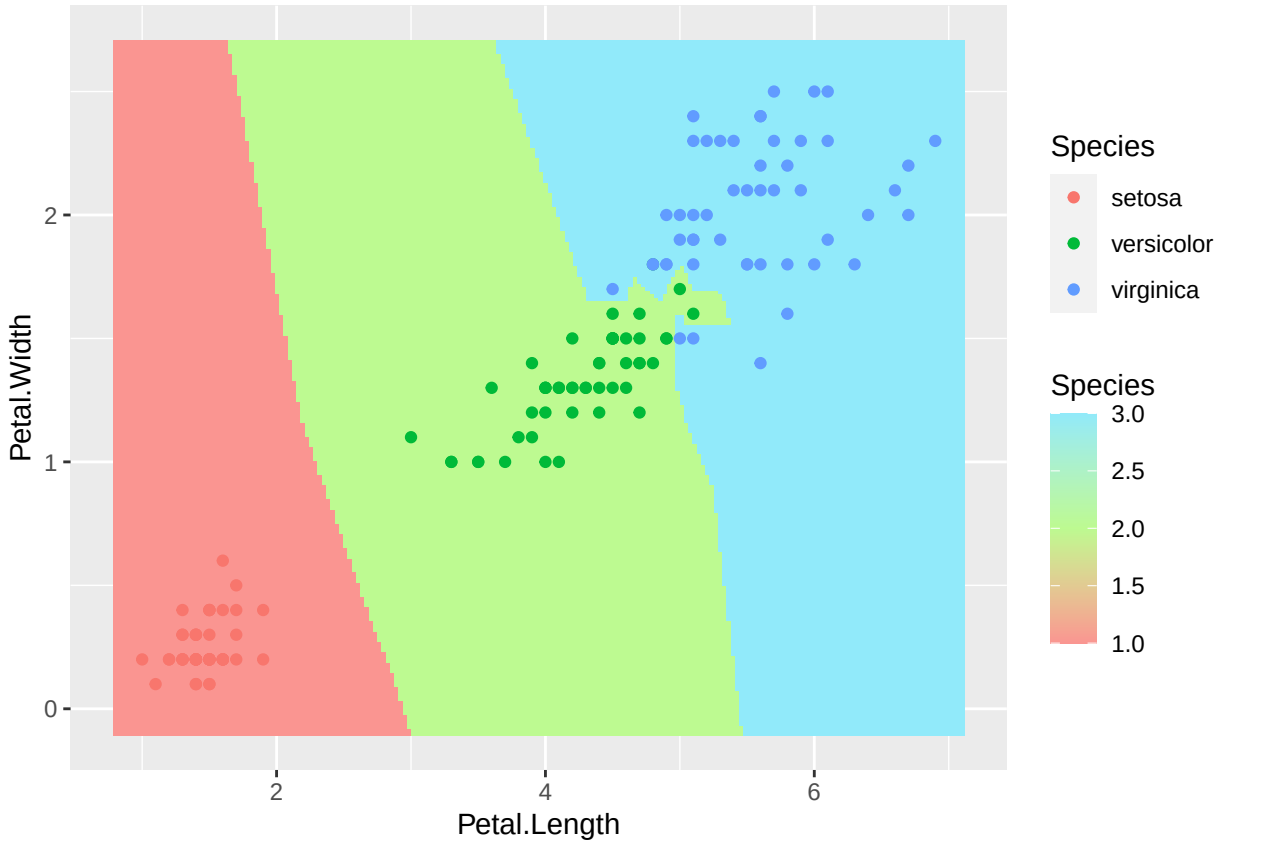
Az osztályozók egy nagy csoportját alkotják az ún. k legközelebbi szomszéd (kNN) osztályozók. Mint a nevéből is egyértelműen kikövetkeztethető, a kNN osztályozók körében a távolság fogalom kulcsfontosságú. A kNN osztályozók működési elve nagyon egyszerű: egy adatsor kiértékelése során megkeressük a tanító adathalmazban az adott adatsorhoz legközelebb elhelyezkedő k darab szomszédos elemet, majd ezek átlagát vagy többségi szavazat alapján a megfelelő osztályt adjuk vissza. Észrevehetjük, hogy a kNN osztályozók esetében *tanulási* folyamatra elvileg nincs szükség, hiszen a prediktálás során a teljes adathalmazt vizsgáljuk. A *legközelebbi* szomszéd fogalom természetesen távolság alapú – itt használhatunk az alapértelmezett euklideszi-távolságon túl tetszőleges távolság metrikát, beleértve a kategórikus adatok között értelmezett távolságmetrikákat is.

Megjegyezzük, hogy általánosságban a kNN osztályzók valószínűségi sűrűség becslők, így számos összetettebb megvalósítása létezik, például kernel alapú megoldások. Ezekkel a megoldásokkal itt nem foglalkozunk.

Tekintsük megint az `iris` adathalmazt, és a `Petal.Length` és `Petal.Width` változóit, majd ábrázoljuk az egyes fajokat két dimenzióban. Ha $k = 1$ paraméterrel futtatunk egy kNN algoritmust, a következő döntési régiókat kapjuk:

```
df<-iris
model<-train(Species~Petal.Length+Petal.Width, iris, method='knn', tuneGrid=data.frame(k=1))

padding<-0.2
petal.length<-seq(min(iris$Petal.Length)-padding,max(iris$Petal.Length)+padding,length.out = 200)
petal.width<-seq(min(iris$Petal.Width)-padding,max(iris$Petal.Width)+padding,length.out = 200)
grid<-expand.grid(petal.length,petal.width)
df<-data.frame(Petal.Length=grid[,1],Petal.Width=grid[,2])
pred<-predict(model,df)
df$Species<-as.numeric(pred)

p<-ggplot(df,aes(Petal.Length,Petal.Width, z=Species)) +
    geom_raster(aes(fill = Species)) +
    geom_point(data = iris, aes(color = Species)) +
```

```
    scale_fill_gradientn(colours=c('#fa9591','#bdfa91','#91eafa'))
plot(p)
```



Amit érdemes első lépésben észrevenni, hogy $k = 1$ érték esetén az osztályozó a tanító adathalmazra tökéletes, hiszen minden osztály legközelebbi szomszédja önmaga, így a döntés (hacsak nincs egy pontban ellentmondásos adatsorunk) bizonyosan jó lesz. Érdemes megtekinteni nagyobb k értékekre a döntési régiókat, például $k = 5$ értékre:

```
df<-iris
model<-train(Species~Petal.Length+Petal.Width, iris, method='knn', tuneGrid=data.frame(k=5))

padding<-0.2
petal.length<-seq(min(iris$Petal.Length)-padding,max(iris$Petal.Length)+padding,length.out = 200)
petal.width<-seq(min(iris$Petal.Width)-padding,max(iris$Petal.Width)+padding,length.out = 200)
grid<-expand.grid(petal.length,petal.width)
df<-data.frame(Petal.Length=grid[,1],Petal.Width=grid[,2])
pred<-predict(model,df)
df$Species<-as.numeric(pred)

p<-ggplot(df,aes(Petal.Length,Petal.Width, z=Species)) +
    geom_raster(aes(fill = Species)) +
    geom_point(data = iris, aes(color = Species)) +
    scale_fill_gradientn(colours=c('#fa9591','#bdfa91','#91eafa'))
plot(p)
```

Ugyan a döntési határ némileg "szőrös lesz", de ezzel egyetemben simul is. Ami szembetűnő, hogy megjelennek hibás osztályozások is. Általánosságban igaz, hogy nagyobb k értékek esetében a döntési határok elsimulnak, növekedik a tanuló adathalmazon a hibás osztályozás, ugyanakkor a modell generalizálhatósága javul.

7.1 Egyszerű tanítási példa

Tekintsük az `oil` adathalmazt, amely különböző étkezési olajok zsírsavtartalmát tartalmazza. Az egyes étkezési olajok fajtáját a `?oil` oldalon találjuk meg.

```
set.seed(1)

data(oil)
df<-cbind(fattyAcids,oilType)
summary(df)
```

```
##     Palmitic        Stearic          Oleic          Linoleic
##  Min.   : 4.50   Min.   :1.700   Min.   :22.80   Min.   : 7.90
##  1st Qu.: 6.20   1st Qu.:3.475   1st Qu.:26.30   1st Qu.:43.10
##  Median : 9.85   Median :4.200   Median :30.70   Median :50.80
##  Mean   : 9.04   Mean   :4.200   Mean   :36.73   Mean   :46.49
##  3rd Qu.:11.12   3rd Qu.:5.000   3rd Qu.:38.62   3rd Qu.:58.08
##  Max.   :14.90   Max.   :6.700   Max.   :76.70   Max.   :66.10
##
##    Linolenic       Eicosanoic      Eicosenoic     oilType
##  Min.   :0.100   Min.   :0.100   Min.   :0.1000   A:37
##  1st Qu.:0.375   1st Qu.:0.100   1st Qu.:0.1000   B:26
##  Median :0.800   Median :0.400   Median :0.1000   C: 3
##  Mean   :2.272   Mean   :0.399   Mean   :0.3115   D: 7
```

```
##  3rd Qu.:2.650   3rd Qu.:0.400   3rd Qu.:0.3000   E:11
##  Max.   :9.500   Max.   :2.800   Max.   :1.8000   F:10
##                                                   G: 2
```

Válogassuk szét az adatokat tanító és teszt adathalmazra, hogy a tanítási folyamat eredményét ki tudjuk értékelni.

```
trainIdx<-createDataPartition(df$oilType, p=0.5, list=F)
trainDf<-df[trainIdx,]
testDf<-df[-trainIdx,]

summary(trainDf)
```

```
##     Palmitic         Stearic          Oleic          Linoleic
##  Min.   : 4.500   Min.   :1.700   Min.   :22.80   Min.   : 8.00
##  1st Qu.: 6.200   1st Qu.:3.375   1st Qu.:26.20   1st Qu.:43.75
##  Median :10.000   Median :4.200   Median :30.70   Median :51.30
##  Mean   : 9.068   Mean   :4.204   Mean   :37.04   Mean   :46.16
##  3rd Qu.:11.175   3rd Qu.:5.075   3rd Qu.:37.62   3rd Qu.:57.02
##  Max.   :13.000   Max.   :6.700   Max.   :75.80   Max.   :66.10
##
##    Linolenic       Eicosanoic      Eicosenoic    oilType
##  Min.   :0.100   Min.   :0.100   Min.   :0.100   A:19
##  1st Qu.:0.300   1st Qu.:0.100   1st Qu.:0.100   B:13
##  Median :0.800   Median :0.400   Median :0.100   C: 2
##  Mean   :2.192   Mean   :0.378   Mean   :0.318   D: 4
##  3rd Qu.:2.175   3rd Qu.:0.400   3rd Qu.:0.200   E: 6
##  Max.   :9.500   Max.   :1.500   Max.   :1.800   F: 5
##                                                  G: 1
```

```
summary(testDf)
```

```
##     Palmitic         Stearic          Oleic          Linoleic
##  Min.   : 4.800   Min.   :1.700   Min.   :23.10   Min.   : 7.90
##  1st Qu.: 6.200   1st Qu.:3.600   1st Qu.:26.43   1st Qu.:42.75
##  Median : 9.700   Median :4.200   Median :30.70   Median :50.15
##  Mean   : 9.009   Mean   :4.196   Mean   :36.39   Mean   :46.84
##  3rd Qu.:11.100   3rd Qu.:5.000   3rd Qu.:39.60   3rd Qu.:58.62
##  Max.   :14.900   Max.   :6.300   Max.   :76.70   Max.   :64.90
##
##    Linolenic       Eicosanoic       Eicosenoic     oilType
##  Min.   :0.100   Min.   :0.1000   Min.   :0.1000   A:18
##  1st Qu.:0.500   1st Qu.:0.1000   1st Qu.:0.1000   B:13
##  Median :0.850   Median :0.4000   Median :0.1000   C: 1
##  Mean   :2.359   Mean   :0.4217   Mean   :0.3043   D: 3
##  3rd Qu.:3.150   3rd Qu.:0.4750   3rd Qu.:0.3000   E: 5
##  Max.   :9.300   Max.   :2.8000   Max.   :1.6000   F: 5
##                                                    G: 1
```

Inicializáljuk egy egyszerű kNN modellt a tanítóadathalmazra, amely során a `caret` csomagra bízzuk a k paraméter megválasztását:

```
model<-train(oilType~., trainDf, method='knn')
```

```
## Warning: predictions failed for Resample10: k=5 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent
```

```
## Warning: predictions failed for Resample10: k=7 Error in dimnames(x) <- dn :
```

```
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample10: k=9 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample11: k=5 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample11: k=7 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample11: k=9 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample17: k=5 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample17: k=7 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample17: k=9 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample19: k=5 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample19: k=7 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample19: k=9 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample20: k=5 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample20: k=7 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample20: k=9 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample21: k=5 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample21: k=7 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample21: k=9 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample23: k=5 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample23: k=7 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample23: k=9 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
print(model)
```

```
## k-Nearest Neighbors
##
## 50 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 50, 50, 50, 50, 50, 50, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.8406759  0.7850812
##   7  0.8400093  0.7807636
##   9  0.7930113  0.7143236
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

```
pred<-predict(model, trainDf)
print(confusionMatrix(pred,trainDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 18  0  0  0  0  0  1
##          B  0 13  0  0  0  0  0
##          C  0  0  0  0  0  0  0
##          D  0  0  0  4  0  0  0
##          E  1  0  0  0  6  0  0
##          F  0  0  2  0  0  5  0
##          G  0  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.92
##                  95% CI : (0.8077, 0.9778)
##     No Information Rate : 0.38
##     P-Value [Acc > NIR] : 1.678e-15
##
##                   Kappa : 0.8934
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.9474     1.00     0.00     1.00   1.0000   1.0000
## Specificity            0.9677     1.00     1.00     1.00   0.9773   0.9556
## Pos Pred Value         0.9474     1.00      NaN     1.00   0.8571   0.7143
## Neg Pred Value         0.9677     1.00     0.96     1.00   1.0000   1.0000
## Prevalence             0.3800     0.26     0.04     0.08   0.1200   0.1000
## Detection Rate         0.3600     0.26     0.00     0.08   0.1200   0.1000
## Detection Prevalence   0.3800     0.26     0.00     0.08   0.1400   0.1400
```

```
## Balanced Accuracy      0.9576     1.00     0.50     1.00   0.9886   0.9778
##                      Class: G
## Sensitivity             0.00
## Specificity             1.00
## Pos Pred Value           NaN
## Neg Pred Value          0.98
## Prevalence              0.02
## Detection Rate          0.00
## Detection Prevalence    0.00
## Balanced Accuracy       0.50
```

Láthatjuk, hogy meglepően jó pontosságot kaptunk, ugyanakkor létezik olyan osztály, amely esetében 0 a szenzitivitás.Érdemes megfigyelni, hogy sok hibaüzenetet kaptunk, amelyek arra utalnak, hogy a bootstrap algoritmus nem minden körben volt sikeres. Valóban, célszerű a helyzetet megvizsgálni részletesebben, állítsuk elő a tanító halmazokat kézzel:

```
set.seed(1)
idx<-createResample(trainDf$oilType, times=10)
model<-train(oilType~., trainDf, method='knn', trControl=trainControl(index=idx))
```

```
## Warning: predictions failed for Resample06: k=5 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample06: k=7 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample06: k=9 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample09: k=5 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample09: k=7 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample09: k=9 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample10: k=5 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample10: k=7 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning: predictions failed for Resample10: k=9 Error in dimnames(x) <- dn :
##   length of 'dimnames' [2] not equal to array extent

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
print(model)
```

```
## k-Nearest Neighbors
##
## 50 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Bootstrapped (10 reps)
```

```
## Summary of sample sizes: 50, 50, 50, 50, 50, 50, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.8402607  0.7844727
##   7  0.7561170  0.6701233
##   9  0.7146109  0.6126063
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

Figyeljük meg, hogy például milyen az osztályok eloszlása a hibás mintavételekben, valamint a modellépítés során kapott pontosságokat a végső modellben:

```
summary(trainDf[idx$Resample06,])
```

```
##     Palmitic        Stearic          Oleic          Linoleic
##  Min.   : 4.50   Min.   :1.700   Min.   :22.80   Min.   : 8.50
##  1st Qu.: 6.20   1st Qu.:3.800   1st Qu.:26.20   1st Qu.:45.20
##  Median :10.00   Median :4.200   Median :28.95   Median :51.30
##  Mean   : 9.15   Mean   :4.194   Mean   :36.42   Mean   :46.54
##  3rd Qu.:11.40   3rd Qu.:5.075   3rd Qu.:36.48   3rd Qu.:57.02
##  Max.   :13.00   Max.   :6.700   Max.   :75.10   Max.   :64.40
##
##    Linolenic       Eicosanoic      Eicosenoic    oilType
##  Min.   :0.100   Min.   :0.100   Min.   :0.100   A:16
##  1st Qu.:0.500   1st Qu.:0.100   1st Qu.:0.100   B:13
##  Median :0.800   Median :0.300   Median :0.100   C: 0
##  Mean   :2.622   Mean   :0.348   Mean   :0.284   D: 5
##  3rd Qu.:5.800   3rd Qu.:0.400   3rd Qu.:0.200   E:10
##  Max.   :9.500   Max.   :1.300   Max.   :1.400   F: 5
##                                                  G: 1
```

```
model$resample$Accuracy
```

```
##  [1] 0.8888889 0.7391304 0.8333333        NA 0.7647059 0.9375000 0.8947368
##  [8]        NA        NA 0.8235294
```

```
model$resample$Resample
```

```
##  [1] "Resample02" "Resample01" "Resample03" "Resample06" "Resample05"
##  [6] "Resample04" "Resample07" "Resample10" "Resample09" "Resample08"
```

Láthatjuk, hogy a hibát az okozta, hogy némely bootstrap mintavételben hiányzik egy-egy osztály.

7.2 Túlmintavételezés

Gondolhatnánk, hogy a kNN algoritmusnál nem számít, hogy az adott osztályokból hány példányt találunk. Ez $k = 1$ esetben így is van, hiszen mindenképpen a legközelebbi osztály határozza meg az osztályozás eredményét. Azonban, az egyes pontokat tudjuk súlyozni például túlmintavételezéssel, ezért meglehetősen különböző eredményeket kaphatunk $k > 1$ esetben:

```
trainDfU<-upSample(x = trainDf, y=trainDf$oilType, list=T)$x
summary(trainDfU)
```

```
##     Palmitic        Stearic          Oleic          Linoleic
##  Min.   : 4.500   Min.   :1.700   Min.   :22.80   Min.   : 8.00
##  1st Qu.: 6.500   1st Qu.:2.300   1st Qu.:27.20   1st Qu.:20.70
```

```
##  Median :10.000   Median :3.300   Median :36.90   Median :47.10
##  Mean   : 9.198   Mean   :3.378   Mean   :44.58   Mean   :37.96
##  3rd Qu.:10.900   3rd Qu.:4.100   3rd Qu.:59.20   3rd Qu.:52.70
##  Max.   :13.000   Max.   :6.700   Max.   :75.80   Max.   :66.10
##
##    Linolenic       Eicosanoic       Eicosenoic     oilType
##  Min.   :0.100   Min.   :0.1000   Min.   :0.1000   A:19
##  1st Qu.:0.300   1st Qu.:0.1000   1st Qu.:0.1000   B:19
##  Median :0.800   Median :0.4000   Median :0.2000   C:19
##  Mean   :2.732   Mean   :0.4865   Mean   :0.5429   D:19
##  3rd Qu.:5.800   3rd Qu.:0.5000   3rd Qu.:0.9000   E:19
##  Max.   :9.500   Max.   :1.5000   Max.   :1.8000   F:19
##                                                    G:19
```

Ebben az esetben a modellillesztés és az eredmények:

```
model<-train(oilType~., trainDfU, method='knn')
print(model)
```

```
## k-Nearest Neighbors
##
## 133 samples
##   7 predictor
##   7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 133, 133, 133, 133, 133, 133, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.9738223  0.9691233
##   7  0.9600464  0.9529207
##   9  0.9292951  0.9174089
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

```
pred<-predict(model, trainDf)
print(confusionMatrix(pred,trainDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 17  0  0  0  0  0  0
##          B  0 13  0  0  0  0  0
##          C  0  0  2  0  0  0  0
##          D  0  0  0  4  0  0  0
##          E  1  0  0  0  6  0  0
##          F  0  0  0  0  0  5  0
##          G  1  0  0  0  0  0  1
##
## Overall Statistics
##
##                Accuracy : 0.96
```

```
##                  95% CI : (0.8629, 0.9951)
##     No Information Rate : 0.38
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9479
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.8947     1.00     1.00     1.00   1.0000      1.0
## Specificity            1.0000     1.00     1.00     1.00   0.9773      1.0
## Pos Pred Value         1.0000     1.00     1.00     1.00   0.8571      1.0
## Neg Pred Value         0.9394     1.00     1.00     1.00   1.0000      1.0
## Prevalence             0.3800     0.26     0.04     0.08   0.1200      0.1
## Detection Rate         0.3400     0.26     0.04     0.08   0.1200      0.1
## Detection Prevalence   0.3400     0.26     0.04     0.08   0.1400      0.1
## Balanced Accuracy      0.9474     1.00     1.00     1.00   0.9886      1.0
##                      Class: G
## Sensitivity            1.0000
## Specificity            0.9796
## Pos Pred Value         0.5000
## Neg Pred Value         1.0000
## Prevalence             0.0200
## Detection Rate         0.0200
## Detection Prevalence   0.0400
## Balanced Accuracy      0.9898
```

8 Naiv Bayes osztályozók

```
# Használt csomagok
library(caret)
```

Az ún. Bayes osztályozók valószínűségszámításon alapuló osztályozók, melyek feladata, hogy az osztályok posterior valószínűségét megbecsüljék. A rövid bemutatásukhoz tételezzük fel, hogy létezik K darab $C_1, C_2, \ldots, C_K$ osztályunk, és szeretnénk eldönteni, hogy az $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ adatsor melyik osztályhoz tartozik. Ekkor célszerűen kiszámíthatjuk a $\mathrm{P}(C_k \mid \mathbf{x})$ valószínűségeket, amelyek megadják, hogy az $\mathbf{x}$ adatsor mekkora valószínűséggel tartozik az egyes osztályokhoz. A Bayes tétel alapján felírhatjuk, hogy

$$\mathrm{P}(C_k \mid \mathbf{x}) = \frac{\mathrm{P}(\mathbf{x} \mid C_k)\mathrm{P}(C_k)}{\mathrm{P}(\mathbf{x})} = \frac{\mathrm{P}(x_1, x_2, \ldots, x_N \mid C_k)\mathrm{P}(C_k)}{\mathrm{P}(\mathbf{x})} = \frac{\mathrm{P}(x_1 \mid C_k)\mathrm{P}(x_2 \mid C_k)\ldots\mathrm{P}(x_N \mid C_k)\mathrm{P}(C_k)}{\mathrm{P}(\mathbf{x})}$$

Az utolsó egyenlőség során felhasználtuk azt a feltételezés (ettől naiv), hogy **az egyes attribútumok (x_n) az osztályoktól feltételesen függetlenek egymástól**. Amint kiszámítjuk a valószínűségeket adott $\mathbf{x}$-hez minden C_k-ra, az adatsorhoz a legnagyobb valószínűséggel rendelkező osztályt rendelhetjük. Szerencsére, mivel a nevező minden osztályra azonos, ezért annak kiszámítására nincs szükség.

8.1 Diszkrét osztályok

Diszkrét prediktorok esetén egyszerű dolgunk van, hiszen az egyes, szorzatban szereplő tényezőket könnyen tudjuk számítani. Tekintsük az alábbi nagyon egyszerű táblázatot:

```
df<-read.csv(text='Outlook,Temperature,Humidity,Wind,Play Tennis
Sunny,Hot,High,Weak,No
Sunny,Hot,High,Strong,No
Overcast,Hot,High,Weak,Yes
Rain,Mild,High,Weak,Yes
Rain,Cool,Normal,Weak,Yes
Rain,Cool,Normal,Strong,No
Overcast,Cool,Normal,Strong,Yes
Sunny,Mild,High,Weak,No
Sunny,Cool,Normal,Weak,Yes
Rain,Mild,Normal,Weak,Yes
Sunny,Mild,Normal,Strong,Yes
Overcast,Mild,High,Strong,Yes
Overcast,Hot,Normal,Weak,Yes
Rain,Mild,High,Strong,No',stringsAsFactor=T)
print(df)
```

```
##     Outlook Temperature Humidity   Wind Play.Tennis
## 1     Sunny         Hot     High   Weak          No
## 2     Sunny         Hot     High Strong          No
## 3  Overcast         Hot     High   Weak         Yes
## 4      Rain        Mild     High   Weak         Yes
## 5      Rain        Cool   Normal   Weak         Yes
## 6      Rain        Cool   Normal Strong          No
## 7  Overcast        Cool   Normal Strong         Yes
## 8     Sunny        Mild     High   Weak          No
## 9     Sunny        Cool   Normal   Weak         Yes
## 10     Rain        Mild   Normal   Weak         Yes
## 11    Sunny        Mild   Normal Strong         Yes
## 12 Overcast        Mild     High Strong         Yes
## 13 Overcast         Hot   Normal   Weak         Yes
## 14     Rain        Mild     High Strong          No
```

```
summary(df)
```

```
##      Outlook  Temperature   Humidity     Wind   Play.Tennis
##  Overcast:4   Cool:4      High  :7   Strong:6   No :5
##  Rain    :5   Hot :4      Normal:7   Weak  :8   Yes:9
##  Sunny   :5   Mild:6
```

A számítás ebben az esetben nagyon egyszerű. Mekkora az esélye, hogy esőben, hőségben , gyenge szél mellett, normális páratartalomban teniszezni fogunk? Számítsuk ki a

$$\mathrm{P}(\mathrm{Yes} \mid \{\mathrm{Rain}, \mathrm{Hot}, \mathrm{Normal}, \mathrm{Weak}\})$$

valószínűséget! Ehhez listázzuk ki a sorokat, melyek a `Yes` kategóriához tartoznak:

```
df[df$Play.Tennis == 'Yes',]
```

```
##     Outlook Temperature Humidity   Wind Play.Tennis
## 3  Overcast         Hot     High   Weak         Yes
## 4      Rain        Mild     High   Weak         Yes
## 5      Rain        Cool   Normal   Weak         Yes
## 7  Overcast        Cool   Normal Strong         Yes
## 9     Sunny        Cool   Normal   Weak         Yes
## 10     Rain        Mild   Normal   Weak         Yes
```

```
## 11    Sunny        Mild   Normal Strong         Yes
## 12 Overcast        Mild     High Strong         Yes
## 13 Overcast         Hot   Normal   Weak         Yes
```

$$\mathrm{P}(\mathrm{Yes}) = 9/14$$
$$\mathrm{P}(\mathrm{Rain} \mid \mathrm{Yes}) = 3/9$$
$$\mathrm{P}(\mathrm{Hot} \mid \mathrm{Yes}) = 2/9$$
$$\mathrm{P}(\mathrm{Normal} \mid \mathrm{Yes}) = 6/9$$
$$\mathrm{P}(\mathrm{Weak} \mid \mathrm{Yes}) = 6/9$$

ezek szorzatából kapjuk, hogy

$$\mathrm{P}(\mathrm{Yes} \mid \{\mathrm{Rain}, \mathrm{Hot}, \mathrm{Normal}, \mathrm{Weak}\}) \propto 0.021164$$

Hasonlóan, a `No` osztályra:

$$\mathrm{P}(\mathrm{No} \mid \{\mathrm{Rain}, \mathrm{Hot}, \mathrm{Normal}, \mathrm{Weak}\}) \propto 0.004571$$

Láthatjuk, hogy a `Yes` válasz valószínűbb.

8.2 Folytonos változók

A folytonos változókat valamilyen folytonos eloszlás szerint modellezzük. Ehhez általában az adatsor várható értékét és szórását tapasztalati úton meghatározzuk, ezzel modellezve a változót. Például, ha az iris adathalmazt tekintjük, a `Petal.Width` változó eloszlását előállíthatjuk a `versicolor` fajra:

```
iris_vers<-iris[iris$Species=='versicolor',]
sigma<-sd(iris_vers$Petal.Width)
mu<-mean(iris_vers$Petal.Width)
print(sigma)
```

```
## [1] 0.1977527
```

```
print(mu)
```

```
## [1] 1.326
```

Egy új bemeneti értékre, például 1.45-re a P(Petal.Width = 1.45 | versicolor) valószínűség a következőképpen számítható:

```
p<-dnorm(1.45,mean=mu,sd=sigma)
print(p)
```

```
## [1] 1.657327
```

8.3 Döntési régiók

Tekintsük megint az `iris` adathalmazt, és a `Petal.Length` és `Petal.Width` változóit, majd ábrázoljuk az egyes fajokat két dimenzióban. Alkalmazzuk rá a naiv Bayes becslést, és rajzoljuk ki megint a döntési régiókat. Az ábrán láthatjuk, hogy az egyes régiók próbálnak hasonlítani egy Gauss-eloszláshoz, ami nem véletlen, hiszen az `iris` változói folytonos változók, így azokat normális eloszlással modellezzük.

8.4 Egyszerű példa

```
set.seed(1)

data(oil)
df<-cbind(fattyAcids,oilType)
summary(df)
```

```
##     Palmitic        Stearic          Oleic          Linoleic
##  Min.   : 4.50   Min.   :1.700   Min.   :22.80   Min.   : 7.90
##  1st Qu.: 6.20   1st Qu.:3.475   1st Qu.:26.30   1st Qu.:43.10
##  Median : 9.85   Median :4.200   Median :30.70   Median :50.80
##  Mean   : 9.04   Mean   :4.200   Mean   :36.73   Mean   :46.49
##  3rd Qu.:11.12   3rd Qu.:5.000   3rd Qu.:38.62   3rd Qu.:58.08
##  Max.   :14.90   Max.   :6.700   Max.   :76.70   Max.   :66.10
##
##    Linolenic       Eicosanoic      Eicosenoic     oilType
##  Min.   :0.100   Min.   :0.100   Min.   :0.1000   A:37
##  1st Qu.:0.375   1st Qu.:0.100   1st Qu.:0.1000   B:26
##  Median :0.800   Median :0.400   Median :0.1000   C: 3
##  Mean   :2.272   Mean   :0.399   Mean   :0.3115   D: 7
##  3rd Qu.:2.650   3rd Qu.:0.400   3rd Qu.:0.3000   E:11
##  Max.   :9.500   Max.   :2.800   Max.   :1.8000   F:10
##                                                   G: 2
```

Válogassuk szét az adatokat tanító és teszt adathalmazra, hogy a tanítási folyamat eredményét ki tudjuk értékelni.

```
trainIdx<-createDataPartition(df$oilType, p=0.5, list=F)
trainDf<-df[trainIdx,]
testDf<-df[-trainIdx,]

summary(trainDf)
```

```
##     Palmitic         Stearic          Oleic          Linoleic
##  Min.   : 4.500   Min.   :1.700   Min.   :22.80   Min.   : 8.00
##  1st Qu.: 6.200   1st Qu.:3.375   1st Qu.:26.20   1st Qu.:43.75
##  Median :10.000   Median :4.200   Median :30.70   Median :51.30
##  Mean   : 9.068   Mean   :4.204   Mean   :37.04   Mean   :46.16
##  3rd Qu.:11.175   3rd Qu.:5.075   3rd Qu.:37.62   3rd Qu.:57.02
##  Max.   :13.000   Max.   :6.700   Max.   :75.80   Max.   :66.10
##
##    Linolenic       Eicosanoic      Eicosenoic    oilType
##  Min.   :0.100   Min.   :0.100   Min.   :0.100   A:19
##  1st Qu.:0.300   1st Qu.:0.100   1st Qu.:0.100   B:13
##  Median :0.800   Median :0.400   Median :0.100   C: 2
##  Mean   :2.192   Mean   :0.378   Mean   :0.318   D: 4
##  3rd Qu.:2.175   3rd Qu.:0.400   3rd Qu.:0.200   E: 6
##  Max.   :9.500   Max.   :1.500   Max.   :1.800   F: 5
##                                                  G: 1
```

```
summary(testDf)
```

```
##     Palmitic         Stearic          Oleic          Linoleic
##  Min.   : 4.800   Min.   :1.700   Min.   :23.10   Min.   : 7.90
##  1st Qu.: 6.200   1st Qu.:3.600   1st Qu.:26.43   1st Qu.:42.75
##  Median : 9.700   Median :4.200   Median :30.70   Median :50.15
##  Mean   : 9.009   Mean   :4.196   Mean   :36.39   Mean   :46.84
##  3rd Qu.:11.100   3rd Qu.:5.000   3rd Qu.:39.60   3rd Qu.:58.62
##  Max.   :14.900   Max.   :6.300   Max.   :76.70   Max.   :64.90
##
##    Linolenic       Eicosanoic       Eicosenoic     oilType
##  Min.   :0.100   Min.   :0.1000   Min.   :0.1000   A:18
##  1st Qu.:0.500   1st Qu.:0.1000   1st Qu.:0.1000   B:13
##  Median :0.850   Median :0.4000   Median :0.1000   C: 1
##  Mean   :2.359   Mean   :0.4217   Mean   :0.3043   D: 3
##  3rd Qu.:3.150   3rd Qu.:0.4750   3rd Qu.:0.3000   E: 5
##  Max.   :9.300   Max.   :2.8000   Max.   :1.6000   F: 5
##                                                    G: 1
```

Próbáljuk meg a tanító halmazra alkalmazni a naiv Bayes megoldásunkat! Jelen esetben nem kell hiperparamétert választanunk, ezért a validációs módszert kikapcsoljuk. Klasszikus naiv Bayes osztályozáshoz kapcsoljuk ki a Laplace-szűrést, és a kernel használatát is!

```
model<-train(oilType~., trainDf, method='naive_bayes',
             trControl=trainControl(method="none"),
             tuneGrid=data.frame(laplace=F,usekernel=F,adjust=1))
print(model)
```

```
## Naive Bayes
##
## 50 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
```

```
##
## No pre-processing
## Resampling: None
```

```
pred<-predict(model, trainDf)
print(confusionMatrix(pred,trainDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction A B C D E F G
##          A 0 0 0 0 0 0 0
##          B 0 0 0 0 0 0 0
##          C 0 0 0 0 0 0 0
##          D 0 0 0 0 0 0 0
##          E 0 0 0 0 0 0 0
##          F 0 0 0 0 0 0 0
##          G 0 0 0 0 0 0 0
##
## Overall Statistics
##
##                Accuracy : NaN
##                  95% CI : (NA, NA)
##     No Information Rate : NA
##     P-Value [Acc > NIR] : NA
##
##                   Kappa : NaN
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity                NA       NA       NA       NA       NA       NA
## Specificity                NA       NA       NA       NA       NA       NA
## Pos Pred Value             NA       NA       NA       NA       NA       NA
## Neg Pred Value             NA       NA       NA       NA       NA       NA
## Prevalence                NaN      NaN      NaN      NaN      NaN      NaN
## Detection Rate            NaN      NaN      NaN      NaN      NaN      NaN
## Detection Prevalence      NaN      NaN      NaN      NaN      NaN      NaN
## Balanced Accuracy          NA       NA       NA       NA       NA       NA
##                      Class: G
## Sensitivity                NA
## Specificity                NA
## Pos Pred Value             NA
## Neg Pred Value             NA
## Prevalence                NaN
## Detection Rate            NaN
## Detection Prevalence      NaN
## Balanced Accuracy          NA
```

Láthatjuk, hogy a modell működésképtelen. A hiba megkereséséhez tekintsük a Bayes modellt:

```
print(model$finalModel)
```

```
##
```

```
## ================================== Naive Bayes ==================================
##
##  Call:
## naive_bayes.default(x = x, y = y, laplace = param$laplace, usekernel = FALSE)
##
## ---------------------------------------------------------------------------------
##
## Laplace smoothing: FALSE
##
## ---------------------------------------------------------------------------------
##
##  A priori probabilities:
##
##    A    B    C    D    E    F    G
## 0.38 0.26 0.04 0.08 0.12 0.10 0.02
##
## ---------------------------------------------------------------------------------
##
##  Tables:
##
## ---------------------------------------------------------------------------------
##  ::: Palmitic (Gaussian)
## ---------------------------------------------------------------------------------
##
## Palmitic          A          B          C          D          E          F
##     mean 10.8789474  6.2538462  9.8000000 11.4000000 10.7833333  5.1000000
##     sd    1.2955389  0.3478874  0.2828427  0.6480741  0.6400521  0.4743416
##
## Palmitic          G
##     mean 10.0000000
##     sd
##
## ---------------------------------------------------------------------------------
##  ::: Stearic (Gaussian)
## ---------------------------------------------------------------------------------
##
## Stearic         A         B         C         D         E         F         G
##    mean 5.4105263 4.1307692 3.3000000 2.8500000 3.9833333 1.9000000 2.3000000
##    sd   0.6154445 0.5137844 0.0000000 0.1290994 0.2857738 0.1581139
##
## ---------------------------------------------------------------------------------
##  ::: Oleic (Gaussian)
## ---------------------------------------------------------------------------------
##
## Oleic          A         B         C         D         E         F         G
##   mean 32.410526 25.823077 58.850000 74.625000 25.466667 58.900000 36.900000
##   sd    3.494900  2.073706  1.626346  1.189888  1.053882  5.018964
##
## ---------------------------------------------------------------------------------
##  ::: Linoleic (Gaussian)
## ---------------------------------------------------------------------------------
##
## Linoleic          A          B          C          D          E          F
##     mean 49.7894737 61.9923077 21.0000000  9.3000000 52.2000000 23.3400000
```

```
##     sd    3.8411362  2.2005535  0.4242641  1.2463279  1.3798551  4.3986362
##
## Linoleic          G
##     mean 47.1000000
##     sd
##
## ------------------------------------------------------------------------------
##  ::: Linolenic (Gaussian)
## ------------------------------------------------------------------------------
##
## Linolenic          A          B          C          D          E          F
##      mean 0.83157895 0.48461538 0.20000000 0.70000000 6.61666667 8.48000000
##      sd   0.94225063 0.45064057 0.00000000 0.08164966 0.87044050 0.91760558
##
## Linolenic          G
##      mean 2.20000000
##      sd
##
## ------------------------------------------------------------------------------
##
## # ... and 2 more tables
##
## ------------------------------------------------------------------------------
```

Láthatjuk, hogy a G osztályhoz minden változó esetén `NA` szórás tartozik, hiszen egyetlen értéket mértünk. Számos lehetőségünk van ennek kijavítására:

- A teljes becslésből eltávolíthatjuk a G osztályt, hiszen mindösszesen 2 példa alapján a naiv Bayes becslők nem hatékonyak
- Mesterségesen adunk hozzá újabb adatsorokat, melyekben a G osztály adatsorához zajt adunk.

Jelenleg az utóbbit fogjuk választani, tekintsük a tanító adathalmaz oszlopainak szórását!

```
sds<-sapply(trainDf[,-ncol(trainDf)],sd)
print(sds)
```

```
##   Palmitic    Stearic      Oleic   Linoleic  Linolenic Eicosanoic Eicosenoic
##  2.5606472  1.2592450 15.7935519 16.5668453  2.9646103  0.3202614  0.4212179
```

Adjunk hozzá egy új sort, amely a G osztály adatsorával egyezik meg, csupán némi zajt keverve hozzá:

```
noise<-sds*rnorm(length(sds)) * 0.05
row<-trainDf[trainDf$oilType == 'G',]
row[1:7]<-row[1:7] + noise
trainDf<-rbind(trainDf,row)
summary(trainDf)
```

```
##     Palmitic         Stearic          Oleic          Linoleic
##  Min.   : 4.500   Min.   :1.700   Min.   :22.80   Min.   : 8.00
##  1st Qu.: 6.200   1st Qu.:3.300   1st Qu.:26.40   1st Qu.:43.90
##  Median :10.000   Median :4.200   Median :31.00   Median :51.30
##  Mean   : 9.087   Mean   :4.166   Mean   :37.04   Mean   :46.16
##  3rd Qu.:11.150   3rd Qu.:5.050   3rd Qu.:37.55   3rd Qu.:56.35
##  Max.   :13.000   Max.   :6.700   Max.   :75.80   Max.   :66.10
##
##    Linolenic       Eicosanoic      Eicosenoic     oilType
##  Min.   :0.100   Min.   :0.100   Min.   :0.1000   A:19
```

```
##  1st Qu.:0.300   1st Qu.:0.100   1st Qu.:0.1000   B:13
##  Median :0.800   Median :0.400   Median :0.1000   C: 2
##  Mean   :2.196   Mean   :0.381   Mean   :0.3214   D: 4
##  3rd Qu.:2.306   3rd Qu.:0.400   3rd Qu.:0.2500   E: 6
##  Max.   :9.500   Max.   :1.500   Max.   :1.8000   F: 5
##                                                   G: 2
```

Végezzük el a tanítást újra!

```
model<-train(oilType~., trainDf, method='naive_bayes',
             trControl=trainControl(method="none"),
             tuneGrid=data.frame(laplace=F,usekernel=F,adjust=1))
print(model)
```

```
## Naive Bayes
##
## 51 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: None
```

```
pred<-predict(model, trainDf)
print(confusionMatrix(pred,trainDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 19  0  0  0  0  0  0
##          B  0 13  0  0  0  0  0
##          C  0  0  2  0  0  0  0
##          D  0  0  0  4  0  0  0
##          E  0  0  0  0  6  0  0
##          F  0  0  0  0  0  5  0
##          G  0  0  0  0  0  0  2
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9302, 1)
##     No Information Rate : 0.3725
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            1.0000   1.0000  1.00000  1.00000   1.0000  1.00000
## Specificity            1.0000   1.0000  1.00000  1.00000   1.0000  1.00000
## Pos Pred Value         1.0000   1.0000  1.00000  1.00000   1.0000  1.00000
## Neg Pred Value         1.0000   1.0000  1.00000  1.00000   1.0000  1.00000
```

```
## Prevalence             0.3725   0.2549  0.03922  0.07843   0.1176  0.09804
## Detection Rate         0.3725   0.2549  0.03922  0.07843   0.1176  0.09804
## Detection Prevalence   0.3725   0.2549  0.03922  0.07843   0.1176  0.09804
## Balanced Accuracy      1.0000   1.0000  1.00000  1.00000   1.0000  1.00000
##                      Class: G
## Sensitivity           1.00000
## Specificity           1.00000
## Pos Pred Value        1.00000
## Neg Pred Value        1.00000
## Prevalence            0.03922
## Detection Rate        0.03922
## Detection Prevalence  0.03922
## Balanced Accuracy     1.00000
```

```
pred<-predict(model, testDf)
print(confusionMatrix(pred,testDf$oilType))
```

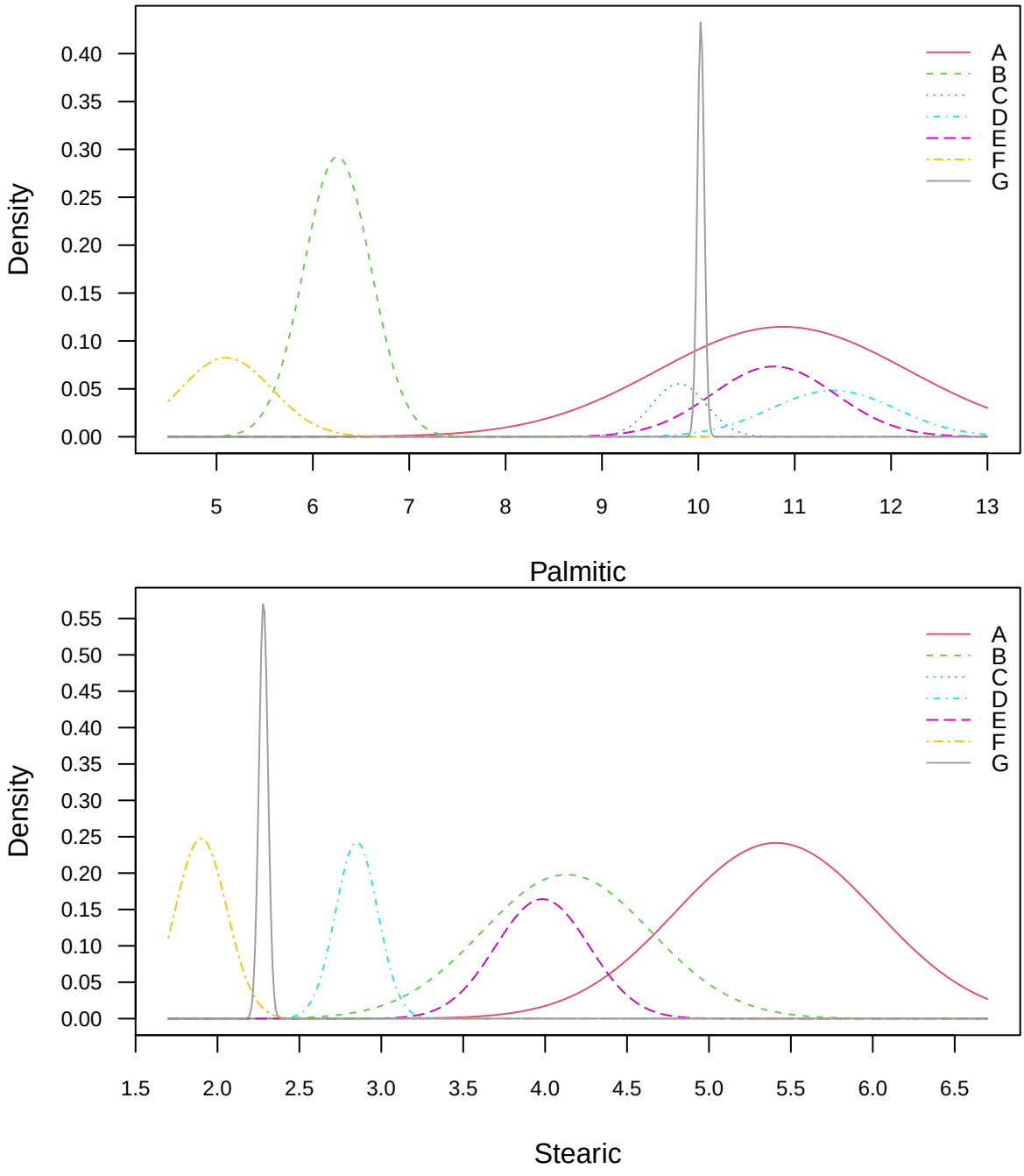
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 18  1  0  0  0  0  1
##          B  0 12  0  0  0  0  0
##          C  0  0  1  0  0  0  0
##          D  0  0  0  2  0  0  0
##          E  0  0  0  0  5  0  0
##          F  0  0  0  1  0  5  0
##          G  0  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.9348
##                  95% CI : (0.821, 0.9863)
##     No Information Rate : 0.3913
##     P-Value [Acc > NIR] : 1.076e-14
##
##                   Kappa : 0.9103
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            1.0000   0.9231  1.00000  0.66667   1.0000   1.0000
## Specificity            0.9286   1.0000  1.00000  1.00000   1.0000   0.9756
## Pos Pred Value         0.9000   1.0000  1.00000  1.00000   1.0000   0.8333
## Neg Pred Value         1.0000   0.9706  1.00000  0.97727   1.0000   1.0000
## Prevalence             0.3913   0.2826  0.02174  0.06522   0.1087   0.1087
## Detection Rate         0.3913   0.2609  0.02174  0.04348   0.1087   0.1087
## Detection Prevalence   0.4348   0.2609  0.02174  0.04348   0.1087   0.1304
## Balanced Accuracy      0.9643   0.9615  1.00000  0.83333   1.0000   0.9878
##                      Class: G
## Sensitivity           0.00000
## Specificity           1.00000
## Pos Pred Value            NaN
```

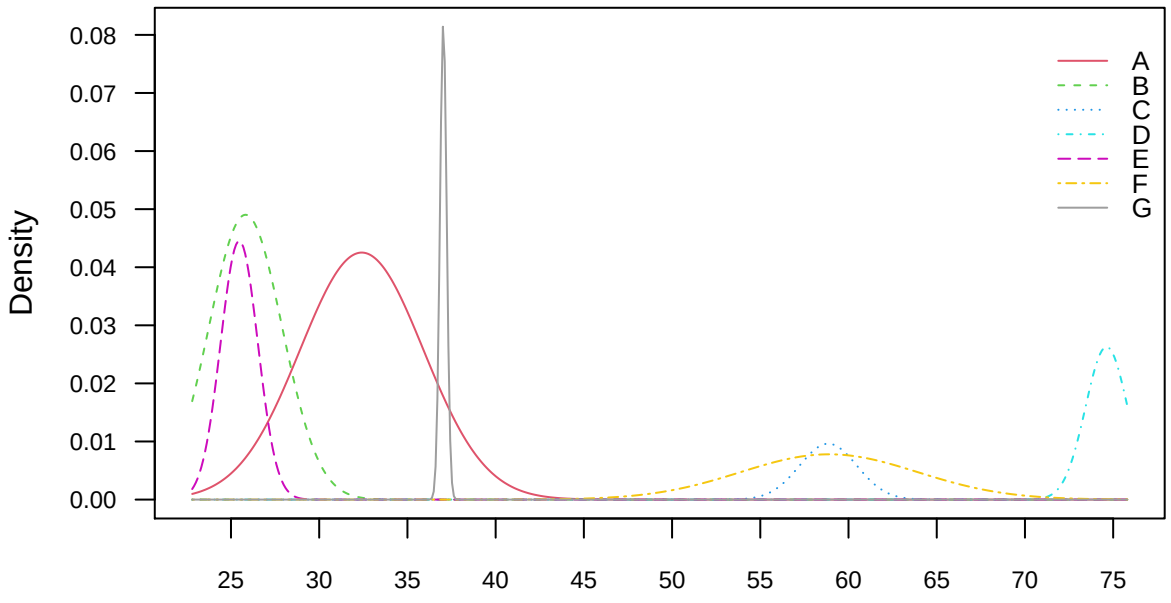
```
## Neg Pred Value         0.97826
## Prevalence             0.02174
## Detection Rate         0.00000
## Detection Prevalence   0.00000
## Balanced Accuracy      0.50000
```

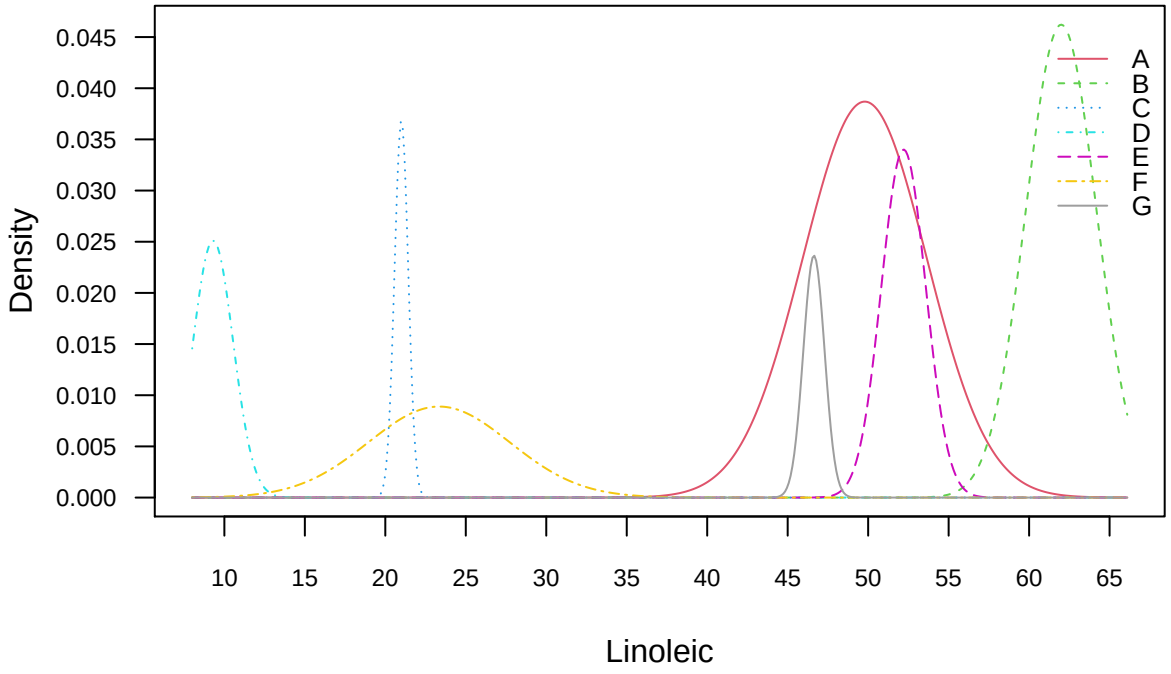
```
pred<-predict(model, testDf, type='prob')
```

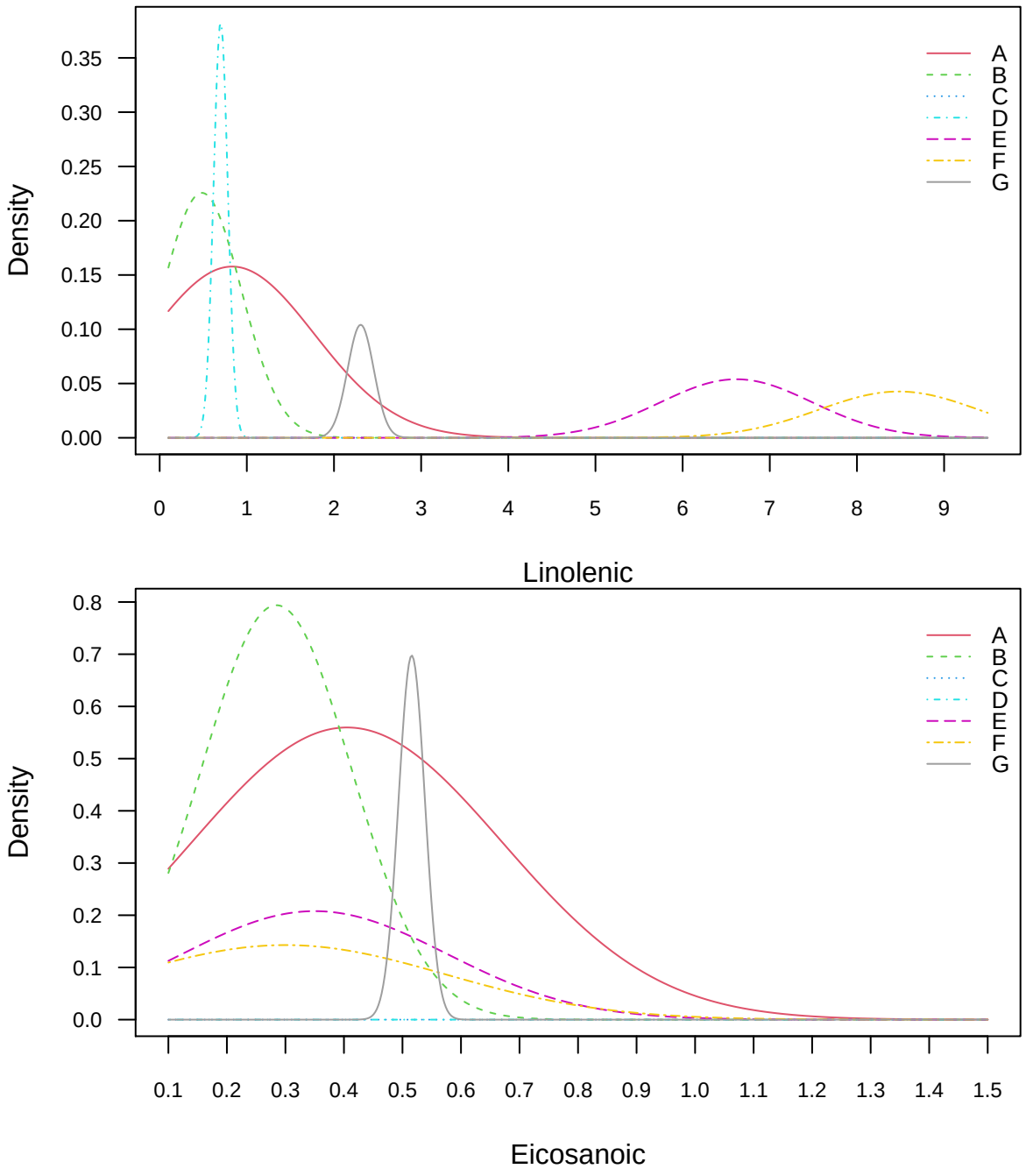
Az egyes feltételes valószínűségek egyszerűen ábrázolhatók a `plot` paranccsal:

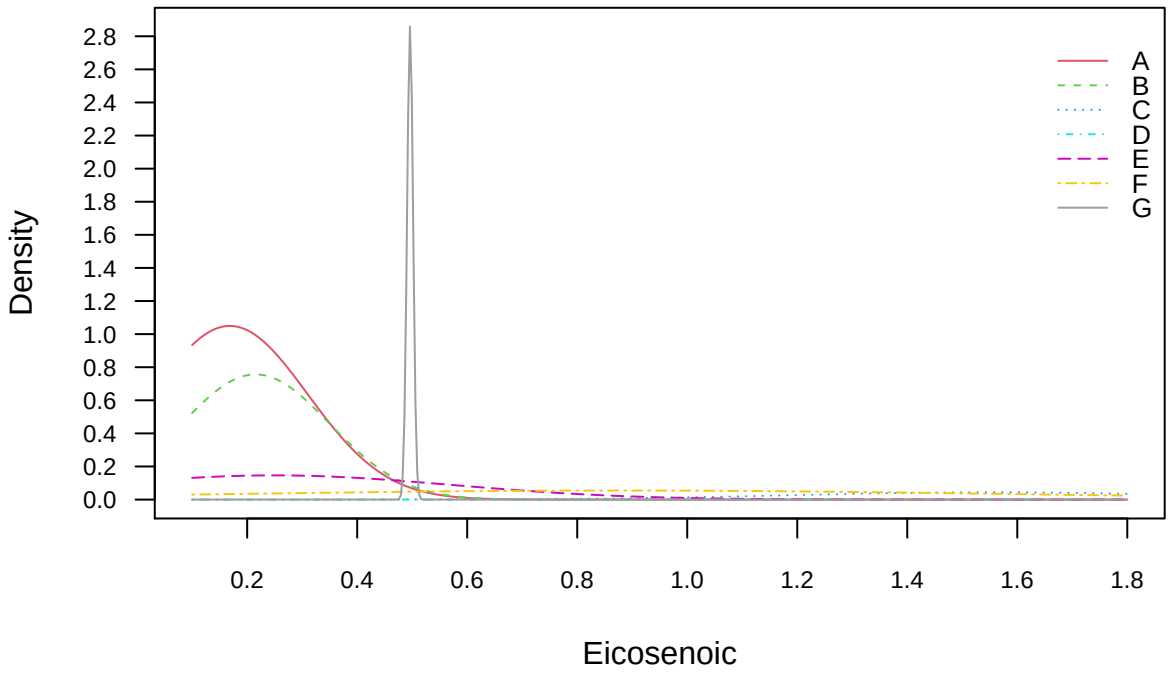
```
plot(model$finalModel)
```




Density
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
25
30
35
40
45
50
55
60
65
70
75
A
B
C
D
E
F
G
Oleic


Density
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
10
15
20
25
30
35
40
45
50
55
60
65
A
B
C
D
E
F
G
Linoleic


Density
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
1
2
3
4
5
6
7
8
9
Linolenic
A
B
C
D
E
F
G
Density
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
1.1
1.2
1.3
1.4
1.5
Eicosanoic
A
B
C
D
E
F
G



8.5 Diszkrét értékek

A tenisz példát a `caret` csomag segítségével egyszerűen tudjuk megoldani, csupán arra kell figyelnünk, hogy a formula alapú interfész a faktorokat dummy változókba alakítja, így célszerű a klasszikus interfészt használni:

```
df<-read.csv(text='Outlook,Temperature,Humidity,Wind,Play Tennis
Sunny,Hot,High,Weak,No
Sunny,Hot,High,Strong,No
Overcast,Hot,High,Weak,Yes
Rain,Mild,High,Weak,Yes
Rain,Cool,Normal,Weak,Yes
Rain,Cool,Normal,Strong,No
Overcast,Cool,Normal,Strong,Yes
Sunny,Mild,High,Weak,No
Sunny,Cool,Normal,Weak,Yes
Rain,Mild,Normal,Weak,Yes
Sunny,Mild,Normal,Strong,Yes
Overcast,Mild,High,Strong,Yes
Overcast,Hot,Normal,Weak,Yes
Rain,Mild,High,Strong,No',stringsAsFactor=T)

model<-train(x=df[,-ncol(df)],y=df$Play.Tennis, method='naive_bayes', tuneGrid=data.frame(usekernel=F, a
```

```
## Warning: naive_bayes(): Feature Outlook - zero probabilities are present.
## Consider Laplace smoothing.
```

```
print(model$finalModel)
```

```
##
## ================================== Naive Bayes ==================================
##
##  Call:
## naive_bayes.default(x = x, y = y, laplace = param$laplace, usekernel = FALSE)
##
```

```
## -------------------------------------------------------------------------------
##
## Laplace smoothing: FALSE
##
## -------------------------------------------------------------------------------
##
##  A priori probabilities:
##
##        No       Yes
## 0.3571429 0.6428571
##
## -------------------------------------------------------------------------------
##
##  Tables:
##
## -------------------------------------------------------------------------------
##  ::: Outlook (Categorical)
## -------------------------------------------------------------------------------
##
## Outlook           No       Yes
##   Overcast 0.0000000 0.4444444
##   Rain     0.4000000 0.3333333
##   Sunny    0.6000000 0.2222222
##
## -------------------------------------------------------------------------------
##  ::: Temperature (Categorical)
## -------------------------------------------------------------------------------
##
## Temperature        No       Yes
##        Cool 0.2000000 0.3333333
##        Hot  0.4000000 0.2222222
##        Mild 0.4000000 0.4444444
##
## -------------------------------------------------------------------------------
##  ::: Humidity (Bernoulli)
## -------------------------------------------------------------------------------
##
## Humidity        No       Yes
##   High   0.8000000 0.3333333
##   Normal 0.2000000 0.6666667
##
## -------------------------------------------------------------------------------
##  ::: Wind (Bernoulli)
## -------------------------------------------------------------------------------
##
## Wind            No       Yes
##   Strong 0.6000000 0.3333333
##   Weak   0.4000000 0.6666667
##
## -------------------------------------------------------------------------------
```

```r
test<-data.frame(Outlook = 'Rain', Temperature='Hot', Humidity='Normal',Wind='Weak')
predict(model,test,type='prob')
```

```
##          No       Yes
```

```
## 1 0.1776316 0.8223684
```

9 Logisztikus regresszió

```
# Használt csomagok
library(caret)
library(MASS)
library(ISLR)
```

A logisztikus regresszió - nevével ellentétben - egy bináris osztályozó megoldás, amely lineáris függvény segítségével próbál valószínűségi modellt felállítani a posterior valószínűségre. A logisztikus regressziónak létezik kategórikus változókra is megoldása, ezzel itt most nem foglalkozunk.

A logisztikus regresszió motiválásához tekintsük a posterior valószínűséget. Tételezzük fel, hogy létezik két darab C_1, C_2 osztályunk, és szeretnénk eldönteni, hogy az $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ adatsor melyik osztályhoz tartozik. Ekkor célszerűen kiszámíthatjuk a $\mathrm{P}(C_1 \mid \mathbf{x})$ valószínűséget, amely megadja, hogy az $\mathbf{x}$ adatsor mekkora valószínűséggel tartozik az egyes osztályhoz. Nyilván $\mathrm{P}(C_2 \mid \mathbf{x}) = 1 - \mathrm{P}(C_1 \mid \mathbf{x})$. Írjuk fel a Bayes-tételt!

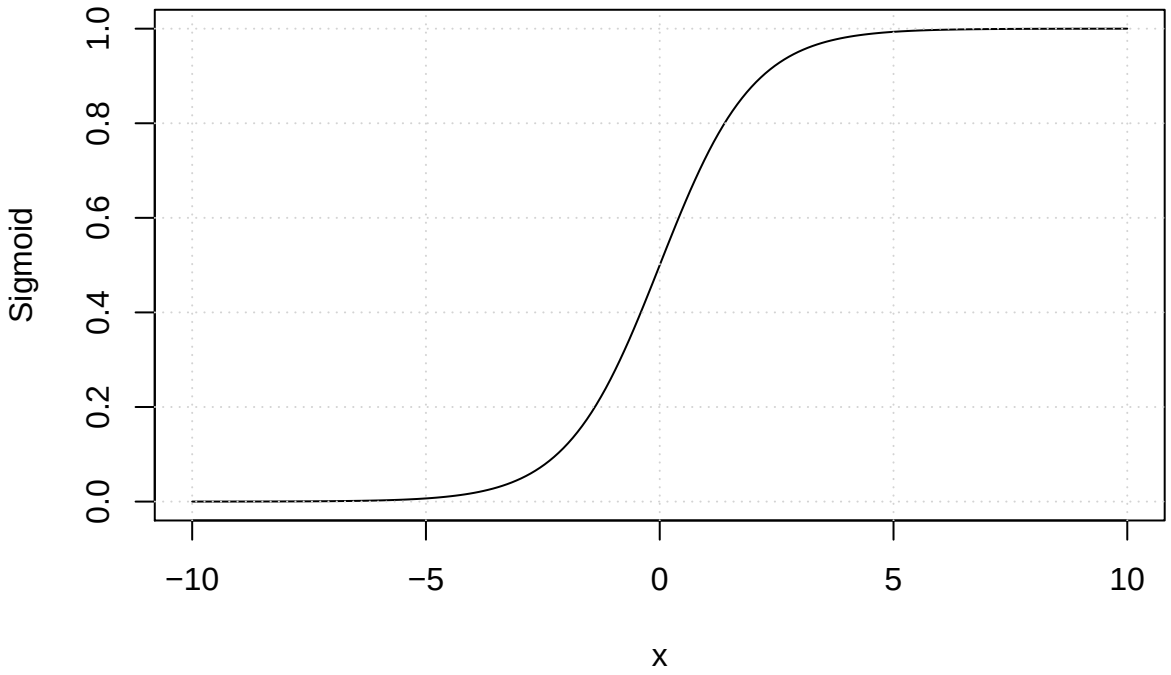
$$\mathrm{P}(C_1 \mid \mathbf{x}) = \frac{\mathrm{P}(\mathbf{x} \mid C_1)\mathrm{P}(C_1)}{\mathrm{P}(\mathbf{x} \mid C_1)\mathrm{P}(C_1) + \mathrm{P}(\mathbf{x} \mid C_2)\mathrm{P}(C_2)} = \frac{1}{1 + \frac{\mathrm{P}(\mathbf{x}|C_2)\mathrm{P}(C_2)}{\mathrm{P}(\mathbf{x}|C_1)\mathrm{P}(C_1)}}$$

Amennyiben a prior valószínűségek megegyeznek, és a likelood valószínűségek azonos szórású, normális eloszlású változók, akkor a posterior a következő formában írható fel:

$$\mathrm{P}(C_1 \mid \mathbf{x}) = \frac{1}{1 + e^{-f(x)}} = \sigma(f(x))$$

Itt a $\sigma()$ függvény az ún. szigmoid függvény, és a formáját a következő ábrán láthatjuk:

```
x<-seq(-10,10,length.out = 200)
plot(x,1/(1+exp(-x)),type='l', ylab='Sigmoid')
grid()
```



Ezt megfordítva felirhatjuk a következő formát:

$$f(x) = \log \frac{\mathrm{P}(C_1 \mid \mathbf{x})}{1 - \mathrm{P}(C_1 \mid \mathbf{x})}$$

ahol a jobboldali kifejezés az odds logaritmusa (*logit*). A trükk, hogy az $f(x)$ függvényt a prediktorok lineáris függvényeként kezeljük, azaz

$$f(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_N x_N$$

A teljes regressziós függvény tehát:

$$\mathrm{P}(C_1 \mid \mathbf{x}) = \frac{1}{1 + e^{-f(x)}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_N x_N)}}$$

9.1 Egyszerű példa, döntési határok

Mivel a logisztikus regresszió alapvetően bináris osztályozás, ezért az `iris` adathalmazból eltávolítjuk a `setosa` fajtákat, és a maradék két fajtára valósítjuk meg a regressziót. A döntési határ látható a következő ábrán. Fontos észrevenni, hogy ugyan a döntési határ egy egyenes, de pereme elmosódott, hiszen a valószínűségeket ábrázoltuk színátmenettel:

9.2 Egyszerű példa

Egyszerű bináris osztályozást végezhetünk a már megismert `Default` adathalmazon. Töltsük be az adathalmazt!

```
set.seed(1)

data(Default)
df<-Default
summary(df)
```

```
##  default    student       balance           income
##  No :9667   No :7056   Min.   :   0.0   Min.   :  772
##  Yes: 333   Yes:2944   1st Qu.: 481.7   1st Qu.:21340
##                        Median : 823.6   Median :34553
##                        Mean   : 835.4   Mean   :33517
##                        3rd Qu.:1166.3   3rd Qu.:43808
##                        Max.   :2654.3   Max.   :73554
```

Válogassuk szét az adatokat tanító és teszt adathalmazra, hogy a tanítási folyamat eredményét ki tudjuk értékelni.

```
trainIdx<-createDataPartition(df$default, p=0.5, list=F)
trainDf<-df[trainIdx,]
testDf<-df[-trainIdx,]

summary(trainDf)
```

```
##  default    student       balance           income
```

```
##  No :4834   No :3533   Min.   :   0.0   Min.   : 1498
##  Yes: 167   Yes:1468   1st Qu.: 480.1   1st Qu.:21272
##                        Median : 817.7   Median :34642
##                        Mean   : 831.3   Mean   :33538
##                        3rd Qu.:1160.8   3rd Qu.:43915
##                        Max.   :2654.3   Max.   :73554
```

```
summary(testDf)
```

```
##  default    student       balance           income
##  No :4833   No :3523   Min.   :   0.0   Min.   :  772
##  Yes: 166   Yes:1476   1st Qu.: 482.8   1st Qu.:21405
##                        Median : 828.9   Median :34500
##                        Mean   : 839.5   Mean   :33496
##                        3rd Qu.:1170.6   3rd Qu.:43691
##                        Max.   :2499.0   Max.   :72461
```

Próbáljuk meg a tanító halmazra alkalmazni a logisztikus regressziót megoldásunkat! A `caret` csomag számos implementációt támogat, az alapértelmezett az R nyelv `glm` generalizált lineáris modelljén alapul. Célszerű ugyanakkor egy regularizált modellt választani, mint például a `plr` megvalósítás.

```
model<-train(default~., trainDf, method='plr', trControl=trainControl(method="boot", number=30, samplin
```

```
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
```

```
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
```

```
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
```

```
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
```

```
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
```

```
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
```

```
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
```

```
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
##
## Convergence warning in plr: 2
```

```
print(model)
```

```
## Penalized Logistic Regression
##
## 5001 samples
##    3 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Bootstrapped (30 reps)
## Summary of sample sizes: 5001, 5001, 5001, 5001, 5001, 5001, ...
## Addtional sampling using down-sampling
##
## Resampling results across tuning parameters:
##
##   lambda  Accuracy   Kappa
##   1e-04   0.8076379  0.1950818
##   1e-02   0.8103233  0.1981884
##   1e-01   0.8288867  0.2213646
##   1e+00   0.8110024  0.2016038
##   4e+00   0.7937332  0.1894879
##   8e+00   0.7930902  0.1848979
##
## Tuning parameter 'cp' was held constant at a value of bic
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were lambda = 0.1 and cp = bic.
```

```
pred<-predict(model, trainDf)
print(confusionMatrix(pred,trainDf$default,positive='Yes'))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  3426   20
##        Yes 1408  147
##
##                Accuracy : 0.7145
##                  95% CI : (0.7017, 0.7269)
##     No Information Rate : 0.9666
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1175
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.88024
##             Specificity : 0.70873
##          Pos Pred Value : 0.09453
```

```
##          Neg Pred Value : 0.99420
##              Prevalence : 0.03339
##          Detection Rate : 0.02939
##    Detection Prevalence : 0.31094
##       Balanced Accuracy : 0.79448
##
##        'Positive' Class : Yes
##
```

```
pred<-predict(model, testDf)
print(confusionMatrix(pred,testDf$default,positive='Yes'))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  3413   13
##        Yes 1420  153
##
##                Accuracy : 0.7133
##                  95% CI : (0.7006, 0.7259)
##     No Information Rate : 0.9668
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1233
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.92169
##             Specificity : 0.70619
##          Pos Pred Value : 0.09727
##          Neg Pred Value : 0.99621
##              Prevalence : 0.03321
##          Detection Rate : 0.03061
##    Detection Prevalence : 0.31466
##       Balanced Accuracy : 0.81394
##
##        'Positive' Class : Yes
##
```

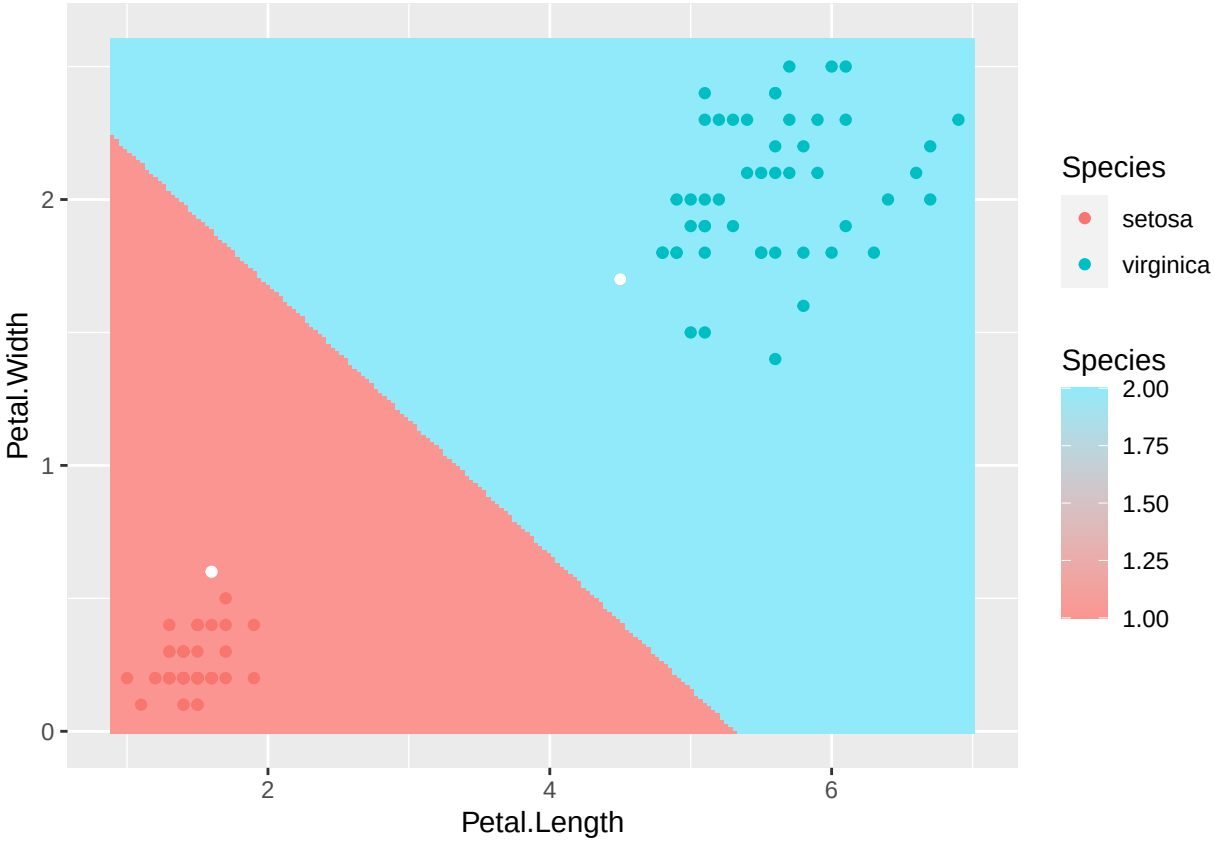
10 Tartóvektor gépek

```
# Használt csomagok
library(caret)
library(MASS)
library(plot3D)
library(plotly)
```

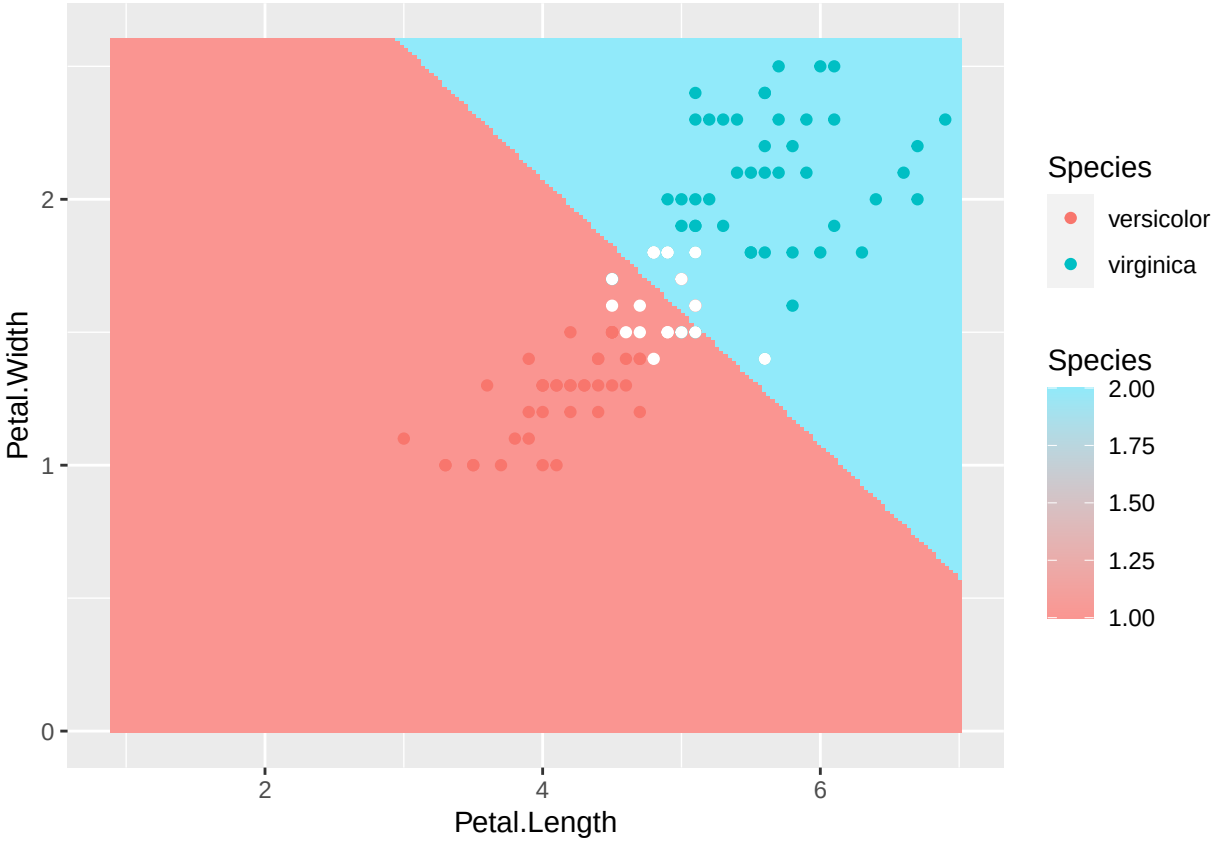
10.1 Bevezetés

A tartóvektor gépek (SVM – Support Vector Machine) bináris osztályozó megoldások, melyek célja, hogy a két osztályba tartozó adatsorok között megtalálja azt a lineáris döntési határt (ún. *hipersíkot*), melyre a legszélesebb határt adja a két ponthalmaz között. A megfogalmazásból érezhetjük, hogy a megoldás során kulcsfontosságú a távolság, mint metrika használata. Tekintsük tehát az `iris` adathalmazt, amelyből

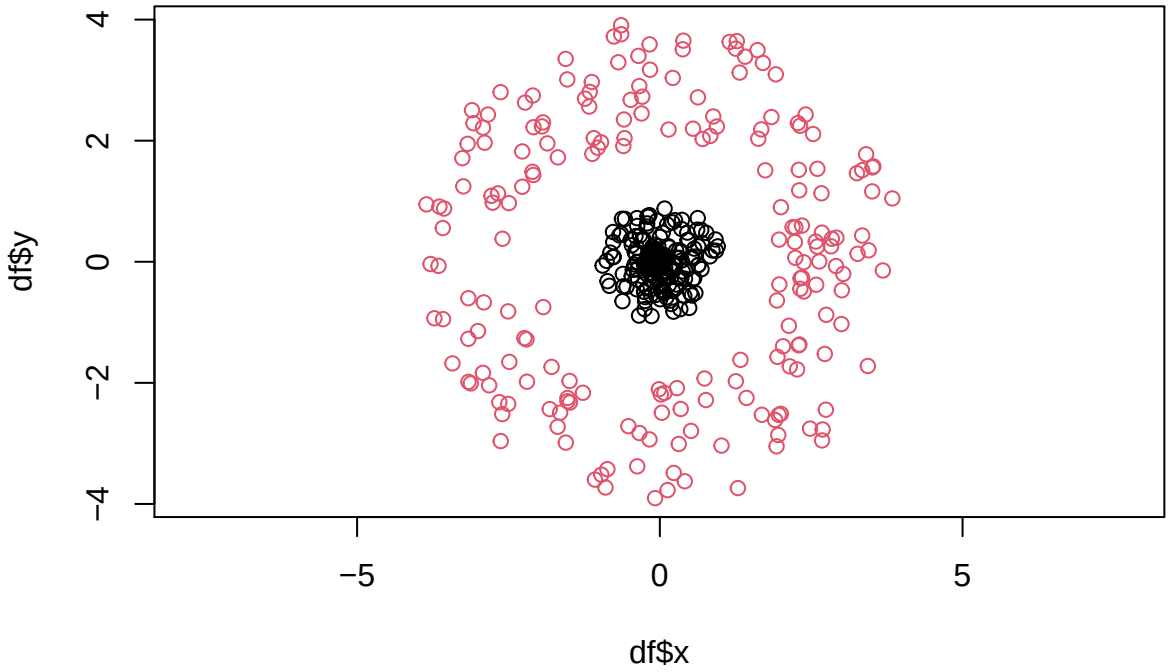
eltávolítottuk a `versicolor` fajt, és így két, egymástól távol elhelyezkedő csoportot kaptunk. Erre lefuttatva az algoritmus, a következő eredményt kapjuk:



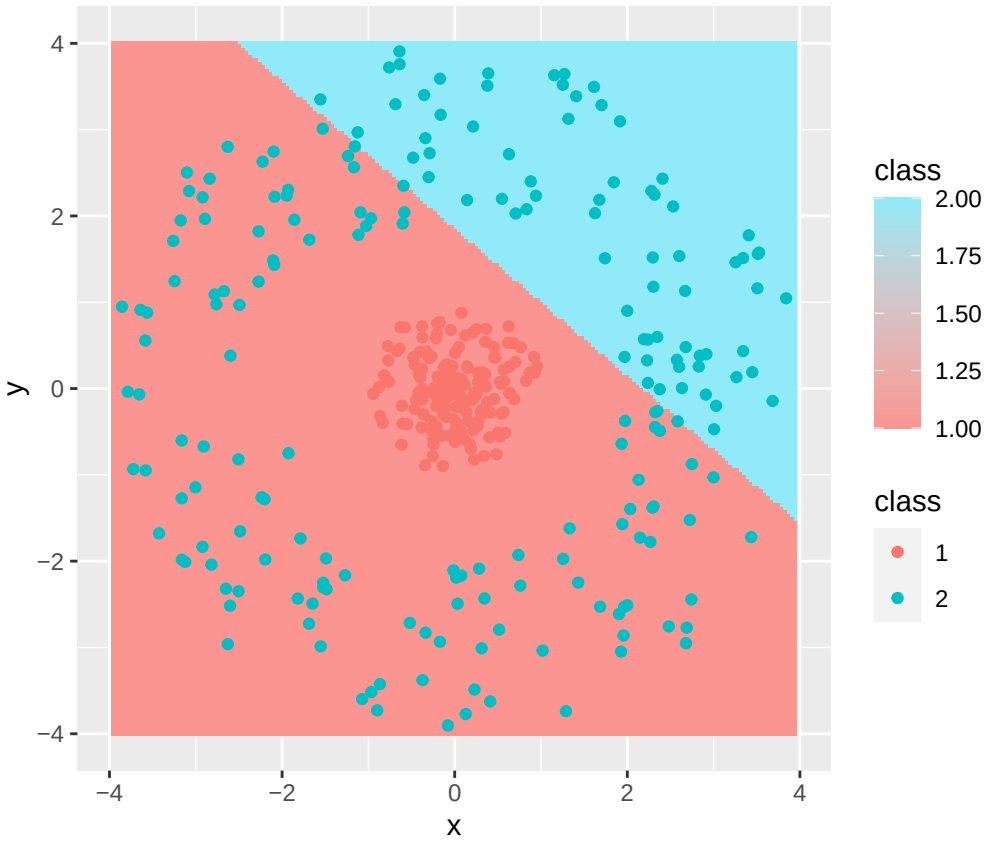
A döntési határon tisztán látszik, hogy az egy lineáris egyenes. Vegyük észre ezen kívül az ábrán a két fehérrel jelölt pontot – ezek az ún. tartóvektorok, hiszen a döntési határhoz képest ezek a legközelebbi pontok. A SVM előnye, hogy a jövőbeli pontok osztályozásához csak a tartóvektorokra van szükség, így csökkentve az osztályozás számításigényét. Persze nem minden esetben lehetséges a csoportok ilyen tisztán történő elkülönítése. Ekkor az SVM megengedi, hogy a határsávon belül helyezkedjenek el pontok, azonban ezek súlyozását egy C paraméterrel szabályozhatjuk, az alábbi ábrán például $C = 1$ értékhez látjuk a döntési határt. Nagyobb C érték kevésbbé megengedő a határszakaszon belül található pontokkal:



Persze a tartóvektor gépek nem lennének túl hasznosak, ha csak lineáris szétválasztásra lennének képesek. A SVM előnye, hogy minden számításban csak az egyes jellemzők skalárszorzata szerepel, ezért könnyen megvalósítható, hogy az SVM ne a jellemzőkre működjön közvetlenül, hanem azok valamilyen – akár nem lineáris – transzformáltjaikra. Az alapötletet jól szemléltethetjük a következő ponthalmazzal:



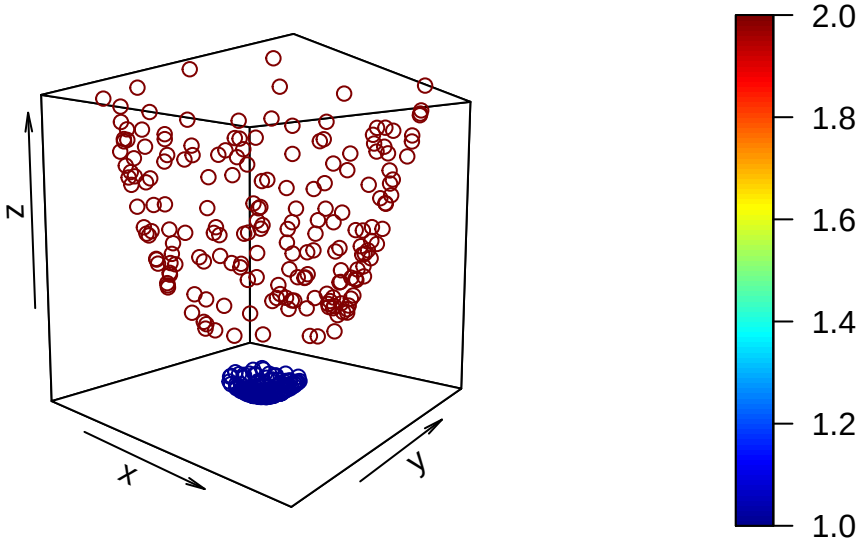
Tisztán látszik, hogy lineáris módszerekkel az adathalmaz nem választható szét:



Azonban, ha bevezetünk egy egyzserű transzformációt, amely minden pontot három dimenzióba transzformál, a transzformált ponthalmazban már lineáris módszerekkel elvégezhetjük az osztályozást. Tételezzük fel, hogy minden egyes pontot áttranszformálunk három dimenzióba a következő módon:

$$[x, y] \to [x, y, x^2 + y^2]$$

Ha ezt a három dimenziós tért ábrázoljuk, akkor a következőt láthatjuk:



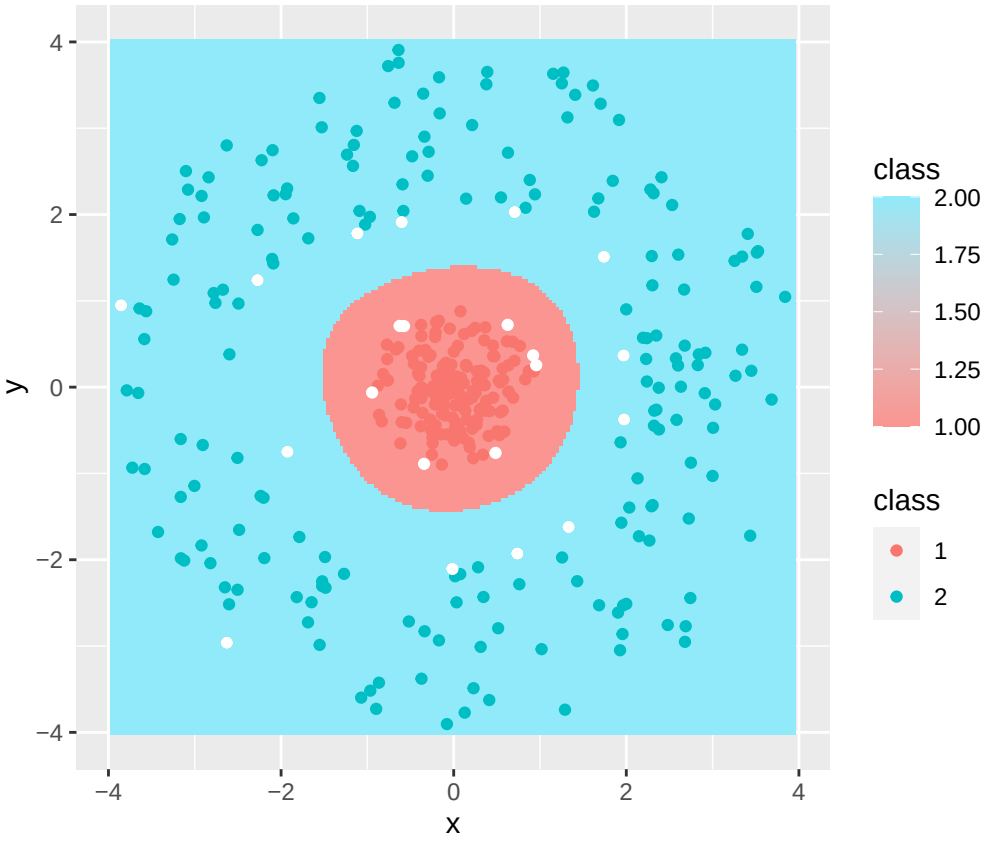
A transzformált térben már nyilvánvaló, hogy egy síkkal el tudjuk a két ponthalmazt választani egymástól. Sajnos, az egyes pontok ilyen transzformációja túlságosan számításigényes, és korlátozott dimenziókra szorít

minket. Éppen ezért, az SVM ún. *kernel* függvényeket használ, amelyek lehetővé teszik hatékonyan, hogy akár végtelen dimenziós terekbe is transzformáljuk a ponthalmazunkat, és ebben a végtelen dimenziós térben végezzük el ponthalmaz szétválasztását.

A legegyszerűbb kernel a **lineáris kernel**, amelyet már láttunk működés közben, ez az eredeti attribútumokat nem transzformálja. Számos egyéb kernelt ismerünk, például a *polinomiális kerneleket*, *Laplace kernel*, stb. A legelterjedtebb kernel az ún. Radial Basis Function Kernel, **RBF kernel** vagy Gauss kernel. Ennek a kernelnek a formája:

$$K(\mathbf{x}, \mathbf{x}') = e^{-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}}$$

Mivel a kernel körszimmetrikus, és láthatóan közeli pontokra nagyobb értéket ad, ezért jellemzője, hogy kör vagy ellipszis alakú csoportokat alkot. Az RBF kernel paramétere a σ paraméter, mely a kernel szélességét szabályozza, nagyobb értéknél a kernel szélesebb (jó analógia lehet a normális eloszlás szórása). RBF kernel használatával a következő döntési régiókat kapjuk a bemutatott ponthalmazra ($C = 1$ és $\sigma = 1$):



10.2 Kategórikus eloszlások

Láttuk, hogy az SVM bináris osztályozó, tehát a teret két részre osztja. Az SVM két osztálynál nagyobb problémák esetén a one-to-rest vagy a one-to-one módszert alkalmazza, jelen esetben a `kernlab` csomag a **one-to-one** megoldást használja.

10.3 Egyszerű példa

Vegyük az étkezési olajokat tartalmazó adathalmazt!

```
set.seed(1)
```

```
data(oil)
df<-cbind(fattyAcids,oilType)
summary(df)
```

```
##     Palmitic        Stearic          Oleic          Linoleic
##  Min.   : 4.50   Min.   :1.700   Min.   :22.80   Min.   : 7.90
##  1st Qu.: 6.20   1st Qu.:3.475   1st Qu.:26.30   1st Qu.:43.10
##  Median : 9.85   Median :4.200   Median :30.70   Median :50.80
##  Mean   : 9.04   Mean   :4.200   Mean   :36.73   Mean   :46.49
##  3rd Qu.:11.12   3rd Qu.:5.000   3rd Qu.:38.62   3rd Qu.:58.08
##  Max.   :14.90   Max.   :6.700   Max.   :76.70   Max.   :66.10
##
##    Linolenic       Eicosanoic      Eicosenoic     oilType
##  Min.   :0.100   Min.   :0.100   Min.   :0.1000   A:37
##  1st Qu.:0.375   1st Qu.:0.100   1st Qu.:0.1000   B:26
##  Median :0.800   Median :0.400   Median :0.1000   C: 3
##  Mean   :2.272   Mean   :0.399   Mean   :0.3115   D: 7
##  3rd Qu.:2.650   3rd Qu.:0.400   3rd Qu.:0.3000   E:11
##  Max.   :9.500   Max.   :2.800   Max.   :1.8000   F:10
##                                                   G: 2
```

Válasszuk szét az adathalmazt tanító és tesztelő adathalmazra!

```
trainIdx<-createDataPartition(df$oilType, p=0.5, list=F)
trainDf<-df[trainIdx,]
testDf<-df[-trainIdx,]

summary(trainDf)
```

```
##     Palmitic        Stearic          Oleic          Linoleic
##  Min.   : 4.500   Min.   :1.700   Min.   :22.80   Min.   : 8.00
##  1st Qu.: 6.200   1st Qu.:3.375   1st Qu.:26.20   1st Qu.:43.75
##  Median :10.000   Median :4.200   Median :30.70   Median :51.30
##  Mean   : 9.068   Mean   :4.204   Mean   :37.04   Mean   :46.16
##  3rd Qu.:11.175   3rd Qu.:5.075   3rd Qu.:37.62   3rd Qu.:57.02
##  Max.   :13.000   Max.   :6.700   Max.   :75.80   Max.   :66.10
##
##    Linolenic       Eicosanoic      Eicosenoic    oilType
##  Min.   :0.100   Min.   :0.100   Min.   :0.100   A:19
##  1st Qu.:0.300   1st Qu.:0.100   1st Qu.:0.100   B:13
##  Median :0.800   Median :0.400   Median :0.100   C: 2
##  Mean   :2.192   Mean   :0.378   Mean   :0.318   D: 4
##  3rd Qu.:2.175   3rd Qu.:0.400   3rd Qu.:0.200   E: 6
##  Max.   :9.500   Max.   :1.500   Max.   :1.800   F: 5
##                                                  G: 1
```

```
summary(testDf)
```

```
##     Palmitic        Stearic          Oleic          Linoleic
##  Min.   : 4.800   Min.   :1.700   Min.   :23.10   Min.   : 7.90
##  1st Qu.: 6.200   1st Qu.:3.600   1st Qu.:26.43   1st Qu.:42.75
##  Median : 9.700   Median :4.200   Median :30.70   Median :50.15
##  Mean   : 9.009   Mean   :4.196   Mean   :36.39   Mean   :46.84
##  3rd Qu.:11.100   3rd Qu.:5.000   3rd Qu.:39.60   3rd Qu.:58.62
##  Max.   :14.900   Max.   :6.300   Max.   :76.70   Max.   :64.90
##
```

```
##    Linolenic       Eicosanoic       Eicosenoic     oilType
##  Min.   :0.100   Min.   :0.1000   Min.   :0.1000   A:18
##  1st Qu.:0.500   1st Qu.:0.1000   1st Qu.:0.1000   B:13
##  Median :0.850   Median :0.4000   Median :0.1000   C: 1
##  Mean   :2.359   Mean   :0.4217   Mean   :0.3043   D: 3
##  3rd Qu.:3.150   3rd Qu.:0.4750   3rd Qu.:0.3000   E: 5
##  Max.   :9.300   Max.   :2.8000   Max.   :1.6000   F: 5
##                                                    G: 1
```

Futtassunk első lépésben egy lineáris kernelt! Ebben az esetben az `svmLinear` módszert használjuk. Minden más paramétert tartsunk alapállapotban:

```
model<-train(oilType~., trainDf, method='svmLinear')
print(model)
```

```
## Support Vector Machines with Linear Kernel
##
## 50 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 50, 50, 50, 50, 50, 50, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8305734  0.7810607
##
## Tuning parameter 'C' was held constant at a value of 1
```

```
print(model$finalModel)
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 24
##
## Objective Function Value : -2.1747 -0.2037 -0.144 -1.6301 -0.1357 -0.4662 -0.0791 -0.0947 -0.347 -0.
## Training error : 0
```

```
pred<-predict(model, trainDf)
print(confusionMatrix(pred,trainDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 19  0  0  0  0  0  0
##          B  0 13  0  0  0  0  0
##          C  0  0  2  0  0  0  0
##          D  0  0  0  4  0  0  0
##          E  0  0  0  0  6  0  0
```

```
##          F  0  0  0  0  0  5  0
##          G  0  0  0  0  0  0  1
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9289, 1)
##     No Information Rate : 0.38
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity             1.00     1.00     1.00     1.00     1.00      1.0
## Specificity             1.00     1.00     1.00     1.00     1.00      1.0
## Pos Pred Value          1.00     1.00     1.00     1.00     1.00      1.0
## Neg Pred Value          1.00     1.00     1.00     1.00     1.00      1.0
## Prevalence              0.38     0.26     0.04     0.08     0.12      0.1
## Detection Rate          0.38     0.26     0.04     0.08     0.12      0.1
## Detection Prevalence    0.38     0.26     0.04     0.08     0.12      0.1
## Balanced Accuracy       1.00     1.00     1.00     1.00     1.00      1.0
##                      Class: G
## Sensitivity             1.00
## Specificity             1.00
## Pos Pred Value          1.00
## Neg Pred Value          1.00
## Prevalence              0.02
## Detection Rate          0.02
## Detection Prevalence    0.02
## Balanced Accuracy       1.00
```

```
pred<-predict(model, testDf)
print(confusionMatrix(pred,testDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 17  1  0  0  0  0  0
##          B  1 12  0  0  0  0  0
##          C  0  0  1  0  0  0  0
##          D  0  0  0  3  0  0  0
##          E  0  0  0  0  5  0  0
##          F  0  0  0  0  0  5  0
##          G  0  0  0  0  0  0  1
##
## Overall Statistics
##
##                Accuracy : 0.9565
##                  95% CI : (0.8516, 0.9947)
##     No Information Rate : 0.3913
```

```
##     P-Value [Acc > NIR] : 4.643e-16
##
##                   Kappa : 0.9411
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.9444   0.9231  1.00000  1.00000   1.0000   1.0000
## Specificity            0.9643   0.9697  1.00000  1.00000   1.0000   1.0000
## Pos Pred Value         0.9444   0.9231  1.00000  1.00000   1.0000   1.0000
## Neg Pred Value         0.9643   0.9697  1.00000  1.00000   1.0000   1.0000
## Prevalence             0.3913   0.2826  0.02174  0.06522   0.1087   0.1087
## Detection Rate         0.3696   0.2609  0.02174  0.06522   0.1087   0.1087
## Detection Prevalence   0.3913   0.2826  0.02174  0.06522   0.1087   0.1087
## Balanced Accuracy      0.9544   0.9464  1.00000  1.00000   1.0000   1.0000
##                      Class: G
## Sensitivity           1.00000
## Specificity           1.00000
## Pos Pred Value        1.00000
## Neg Pred Value        1.00000
## Prevalence            0.02174
## Detection Rate        0.02174
## Detection Prevalence  0.02174
## Balanced Accuracy     1.00000
```

Láthatjuk, hogy a lineáris SVM nagyon jó eredményeket ad az adathalmazra, tehát az minden valószínűség szerint hatékonyan szétválasztható egyenesek mentén. Tekintsük a radiális kernelt, ahol az `svmRadial` módszert kell használnunk:

```
model<-train(oilType~., trainDf, method='svmRadial')
print(model)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 50 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 50, 50, 50, 50, 50, 50, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.6681405  0.5255735
##   0.50  0.7607570  0.6757443
##   1.00  0.7910427  0.7212218
##
## Tuning parameter 'sigma' was held constant at a value of 0.2381935
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.2381935 and C = 1.
```

```r
print(model$finalModel)
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.238193542843721
##
## Number of Support Vectors : 31
##
## Objective Function Value : -4.3173 -1.7225 -1.5352 -4.0242 -2.5821 -1.4859 -1.3434 -1.2553 -1.9573 -
## Training error : 0.06
```

```r
pred<-predict(model, trainDf)
print(confusionMatrix(pred,trainDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 17  0  0  0  0  0  0
##          B  1 13  0  0  0  0  0
##          C  0  0  2  0  0  0  0
##          D  0  0  0  4  0  0  0
##          E  1  0  0  0  6  0  1
##          F  0  0  0  0  0  5  0
##          G  0  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.94
##                  95% CI : (0.8345, 0.9875)
##     No Information Rate : 0.38
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9211
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.8947   1.0000     1.00     1.00   1.0000      1.0
## Specificity            1.0000   0.9730     1.00     1.00   0.9545      1.0
## Pos Pred Value         1.0000   0.9286     1.00     1.00   0.7500      1.0
## Neg Pred Value         0.9394   1.0000     1.00     1.00   1.0000      1.0
## Prevalence             0.3800   0.2600     0.04     0.08   0.1200      0.1
## Detection Rate         0.3400   0.2600     0.04     0.08   0.1200      0.1
## Detection Prevalence   0.3400   0.2800     0.04     0.08   0.1600      0.1
## Balanced Accuracy      0.9474   0.9865     1.00     1.00   0.9773      1.0
##                      Class: G
## Sensitivity              0.00
```

```
## Specificity              1.00
## Pos Pred Value            NaN
## Neg Pred Value           0.98
## Prevalence               0.02
## Detection Rate           0.00
## Detection Prevalence     0.00
## Balanced Accuracy        0.50
```

```
pred<-predict(model, testDf)
print(confusionMatrix(pred,testDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 17  1  0  0  0  0  1
##          B  1 12  0  0  0  0  0
##          C  0  0  1  0  0  0  0
##          D  0  0  0  2  0  0  0
##          E  0  0  0  0  5  0  0
##          F  0  0  0  1  0  5  0
##          G  0  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.913
##                  95% CI : (0.7921, 0.9758)
##     No Information Rate : 0.3913
##     P-Value [Acc > NIR] : 1.829e-13
##
##                   Kappa : 0.8808
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.9444   0.9231  1.00000  0.66667   1.0000   1.0000
## Specificity            0.9286   0.9697  1.00000  1.00000   1.0000   0.9756
## Pos Pred Value         0.8947   0.9231  1.00000  1.00000   1.0000   0.8333
## Neg Pred Value         0.9630   0.9697  1.00000  0.97727   1.0000   1.0000
## Prevalence             0.3913   0.2826  0.02174  0.06522   0.1087   0.1087
## Detection Rate         0.3696   0.2609  0.02174  0.04348   0.1087   0.1087
## Detection Prevalence   0.4130   0.2826  0.02174  0.04348   0.1087   0.1304
## Balanced Accuracy      0.9365   0.9464  1.00000  0.83333   1.0000   0.9878
##                      Class: G
## Sensitivity           0.00000
## Specificity           1.00000
## Pos Pred Value            NaN
## Neg Pred Value        0.97826
## Prevalence            0.02174
## Detection Rate        0.00000
## Detection Prevalence  0.00000
## Balanced Accuracy     0.50000
```

Láthatjuk, hogy nem olyan pontos a módszer, mint lineáris kernel esetében. Érdemes megfigyelni, hogy a módszer 3 C értéket próbált ki, ugyanakkor a sigma értéket nem változtatta. Mivel itt két paraméter változtatható, ezért az algoritmusnak értékpárokat kell megadni, ha kézzel szeretnénk vezérelni az osztályozást:

```
model<-train(oilType~., trainDf, method='svmRadial',
             tuneGrid=data.frame(C=c(0.25,0.5,1), sigma=c(1,2,3)))
print(model)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 50 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 50, 50, 50, 50, 50, 50, ...
## Resampling results across tuning parameters:
##
##   C     sigma  Accuracy   Kappa
##   0.25  1      0.6385305  0.4385328
##   0.50  2      0.6634044  0.4776308
##   1.00  3      0.6820204  0.5126083
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 3 and C = 1.
```

Ha esetleg több kominációt szeretnénk kipróbálni, akkor érdemes azokat egy paraméter rácsra elosztani:

```
pGrid<-expand.grid(
  C=c(0.25,0.5,1),
  sigma=c(.5,1,2,3)
)
print(pGrid)
```

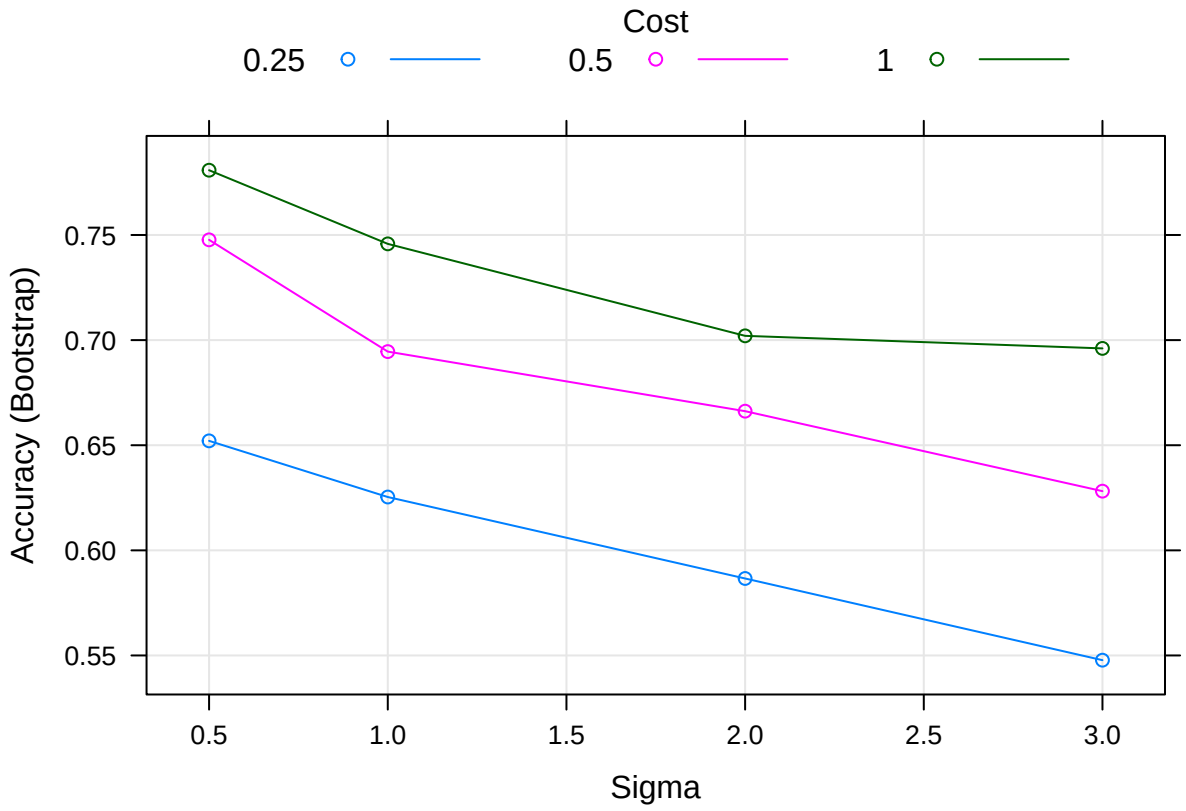
```
##       C sigma
## 1  0.25   0.5
## 2  0.50   0.5
## 3  1.00   0.5
## 4  0.25   1.0
## 5  0.50   1.0
## 6  1.00   1.0
## 7  0.25   2.0
## 8  0.50   2.0
## 9  1.00   2.0
## 10 0.25   3.0
## 11 0.50   3.0
## 12 1.00   3.0
```

```
model<-train(oilType~., trainDf, method='svmRadial',
             tuneGrid=pGrid)
print(model)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 50 samples
##  7 predictor
```

```
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 50, 50, 50, 50, 50, 50, ...
## Resampling results across tuning parameters:
##
##   C     sigma  Accuracy   Kappa
##   0.25  0.5    0.6520819  0.4799365
##   0.25  1.0    0.6253675  0.4300242
##   0.25  2.0    0.5866090  0.3538560
##   0.25  3.0    0.5477547  0.2915582
##   0.50  0.5    0.7476951  0.6457127
##   0.50  1.0    0.6945298  0.5571446
##   0.50  2.0    0.6661928  0.5007655
##   0.50  3.0    0.6281591  0.4326436
##   1.00  0.5    0.7808166  0.6959145
##   1.00  1.0    0.7457840  0.6400777
##   1.00  2.0    0.7020274  0.5598675
##   1.00  3.0    0.6960500  0.5465174
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.5 and C = 1.
```

```
plot(model)
```




```
pred<-predict(model, trainDf)
print(confusionMatrix(pred,trainDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 18  0  0  0  0  0  0
##          B  1 13  0  0  0  0  0
##          C  0  0  2  0  0  0  0
##          D  0  0  0  4  0  0  0
##          E  0  0  0  0  6  0  0
##          F  0  0  0  0  0  5  0
##          G  0  0  0  0  0  0  1
##
## Overall Statistics
##
##                Accuracy : 0.98
##                  95% CI : (0.8935, 0.9995)
##     No Information Rate : 0.38
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9736
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.9474   1.0000     1.00     1.00     1.00      1.0
## Specificity            1.0000   0.9730     1.00     1.00     1.00      1.0
## Pos Pred Value         1.0000   0.9286     1.00     1.00     1.00      1.0
## Neg Pred Value         0.9688   1.0000     1.00     1.00     1.00      1.0
## Prevalence             0.3800   0.2600     0.04     0.08     0.12      0.1
## Detection Rate         0.3600   0.2600     0.04     0.08     0.12      0.1
## Detection Prevalence   0.3600   0.2800     0.04     0.08     0.12      0.1
## Balanced Accuracy      0.9737   0.9865     1.00     1.00     1.00      1.0
##                      Class: G
## Sensitivity              1.00
## Specificity              1.00
## Pos Pred Value           1.00
## Neg Pred Value           1.00
## Prevalence               0.02
## Detection Rate           0.02
## Detection Prevalence     0.02
## Balanced Accuracy        1.00
```

```
pred<-predict(model, testDf)
print(confusionMatrix(pred,testDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 17  1  0  2  0  0  1
##          B  1 12  0  0  0  0  0
##          C  0  0  1  0  0  0  0
##          D  0  0  0  1  0  0  0
```

```
##          E  0  0  0  0  5  0  0
##          F  0  0  0  0  0  5  0
##          G  0  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.8913
##                  95% CI : (0.7643, 0.9638)
##     No Information Rate : 0.3913
##     P-Value [Acc > NIR] : 2.432e-12
##
##                   Kappa : 0.8482
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.9444   0.9231  1.00000  0.33333   1.0000   1.0000
## Specificity            0.8571   0.9697  1.00000  1.00000   1.0000   1.0000
## Pos Pred Value         0.8095   0.9231  1.00000  1.00000   1.0000   1.0000
## Neg Pred Value         0.9600   0.9697  1.00000  0.95556   1.0000   1.0000
## Prevalence             0.3913   0.2826  0.02174  0.06522   0.1087   0.1087
## Detection Rate         0.3696   0.2609  0.02174  0.02174   0.1087   0.1087
## Detection Prevalence   0.4565   0.2826  0.02174  0.02174   0.1087   0.1087
## Balanced Accuracy      0.9008   0.9464  1.00000  0.66667   1.0000   1.0000
##                      Class: G
## Sensitivity           0.00000
## Specificity           1.00000
## Pos Pred Value            NaN
## Neg Pred Value        0.97826
## Prevalence            0.02174
## Detection Rate        0.00000
## Detection Prevalence  0.00000
## Balanced Accuracy     0.50000
```

11 Neurális hálózatok

```r
# Használt csomagok
library(caret)
library(MASS)
library(NeuralNetTools)
```

A neurális hálózatok egyszerű, ún. mesterséges neuronokból képzett hálózatok. A neurális hálózatok alkalmasak többek között bináris és kategorikus osztályozási feladatok megoldására és regressziós feladatok megoldására is.

A neuronok számos bemenettel rendelkeznek, és minden bemenethez súlyokat rendelnek, majd ezeket összegzik. A kimenetükre egy – tipikusan nem lineáris – aktivációs függvényt alkalmaznak. Mesterséges neuront láthatunk az alábbi ábrán:

Matematikai formában egy neuron kimenete az alábbi formában írható fel:

$$o_j = \varphi(w_{1j}x_1 + w_{2j}x_2 + \cdots + w_{nj}x_n)$$

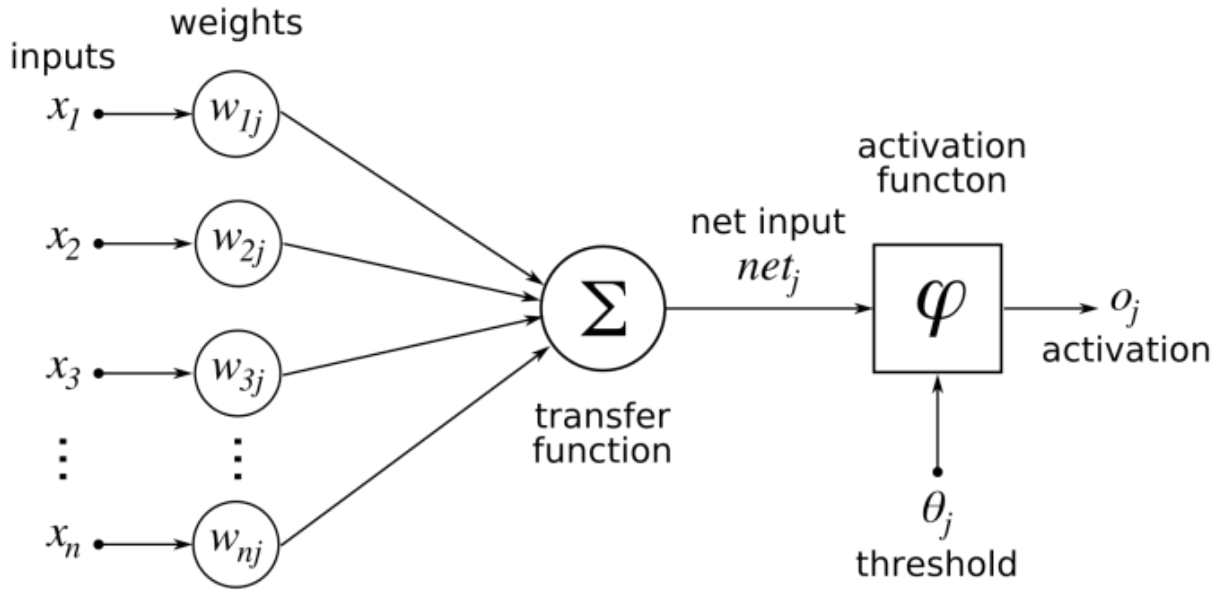


Figure 2: Mesterséges neuron

Aktivációs függvénynek leggyakrabban a következő függvényeket használják:

- szigmoid függvény
- softmax függvény
- `tanh` függvény
- RELU (Rectified Linear Unit), egyenirányított lineáris függvény
- stb.

Egy tipikus, ún. feedforward neurális háló látható a következő ábrán, ahol az egyes körök egy-egy mesterséges neuront ábrázolnak:

A hálózatban láthatjuk, hogy számos rejtett (hidden) réteg szerepelhet. Kulcsfontosságú a kimeneti réteg, hiszen a kimeneti rétegnek valamilyen értelmezhető mennyiséget kell mutatnia. Éppen ezért, alkalmazástól függ, hogy milyen kiemeneti neuronokat használunk. Például, bináris osztályozás esetén célszerűen szigmoid aktivációs függvényt használunk, mely lehetővé teszi, hogy az egyik osztály posterior valószínűségét megbecsüljük. A neurális hálózatok működésének megismeréséhez érdemes megtekinteni egy egyszerű szimulációs oldalt.

Megjegyezzük, hogy az megszokott tárgyalással szemben, ahol az egyes hálózatokat neuronokkal ábrázolják, és ezek közti kapcsolatokat elemzik, napjainkban a neurális hálózatok témakörében minden olyan hálózatot neurális hálózatnak tekintenek, amely az ún. *backpropagation* algoritmussal tanítható. A neurális hálózatok témaköre hatalmas, és egyik jellemzője a mai hálózatoknak, hogy hatalmas a paraméter terük, nem ritkák a több millió paraméterrel rendelkező hálózatok. Gyakorlati alkalmazásukban a hálózatok mélysége (azaz rejtett rétegeinek száma) kiemelkedő fontosságú.

A neurális hálózatok azonban hatékonyan használhatók egyszerű adatelemzési feladatokra is, akkor is, ha nem alkalmazunk mély hálózatokat. Éppen ezért, tárgyalásunkban csak az *egy rejtett réteggel* rendelkező hálózatok alkalmazását mutatjuk be. Az egyszerű osztályozási feladatok megoldása során nem kritikus a hálózatok pontos működésének megismerése (de természetesen hasznos).

11.1 Döntési határ, nem lineáris döntési határok

Tekintsük egy egyszerű, bináris osztályozást. Az `iris` adathalmazból eltávolítjuk a `versicolor` fajt, és egy egyszerszerű, egy rejtett mesterséges neuronból álló hálózatot illesztünk a ponthalmazra (`size=1`):

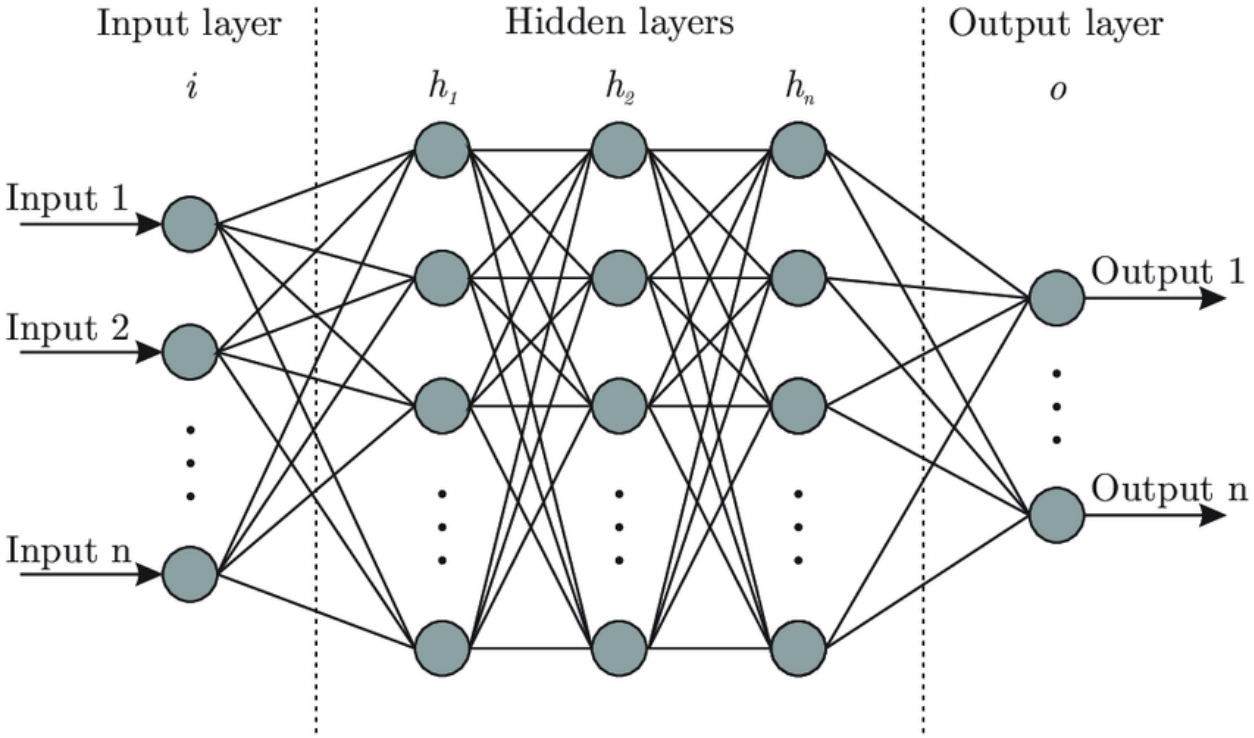
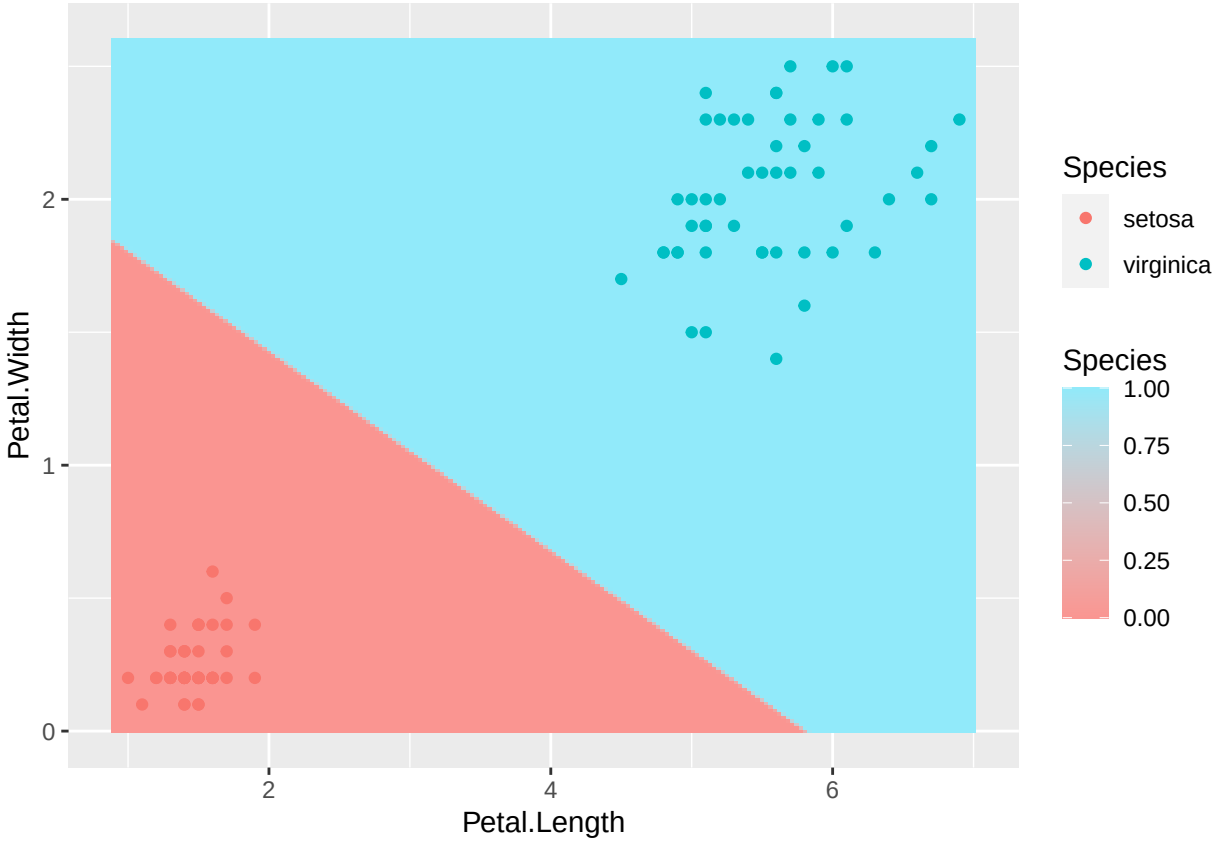
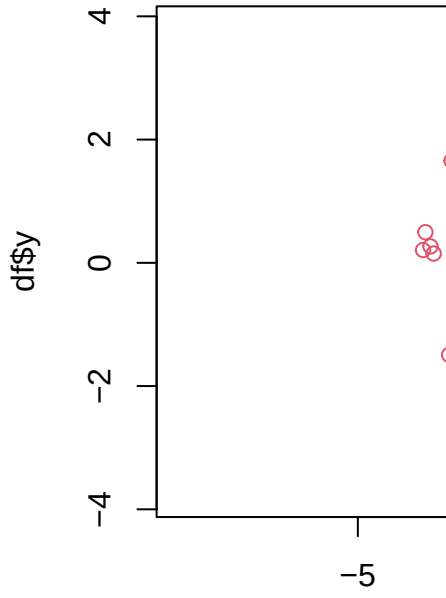


Figure 3: Feedforward Neural Network

```
## # weights:  5
## initial  value 64.070486
## final  value 0.000000
## converged
```

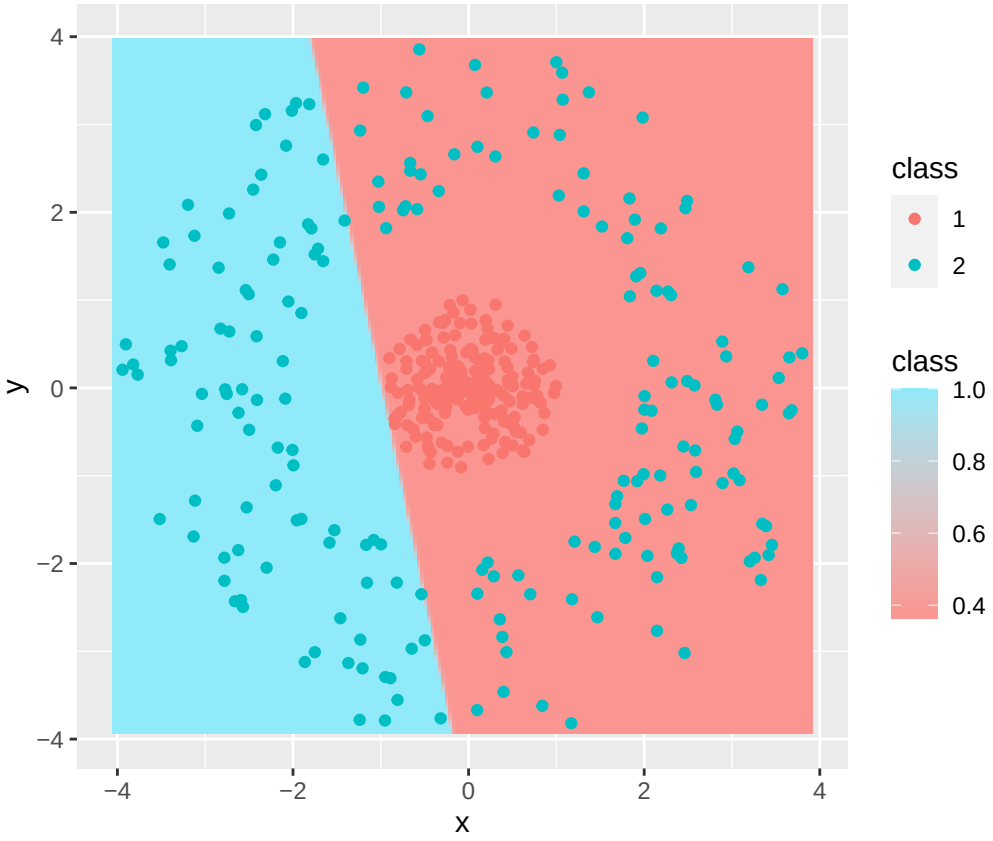


Ha az algoritmust többször lefuttatjuk, azt láthatjuk, hogy a döntési határ egy egyenes, de a helye folyamatosan változik. Ennek az az egyszerű oka – ellentétben az SVM algoritmussal –, hogy a neurális hálózatok osztályozás esetén az osztályozás pontosságát (valóságban a keresztentrópiát) maximalizálják, erre pedig természetesen több megoldás is létezhet.



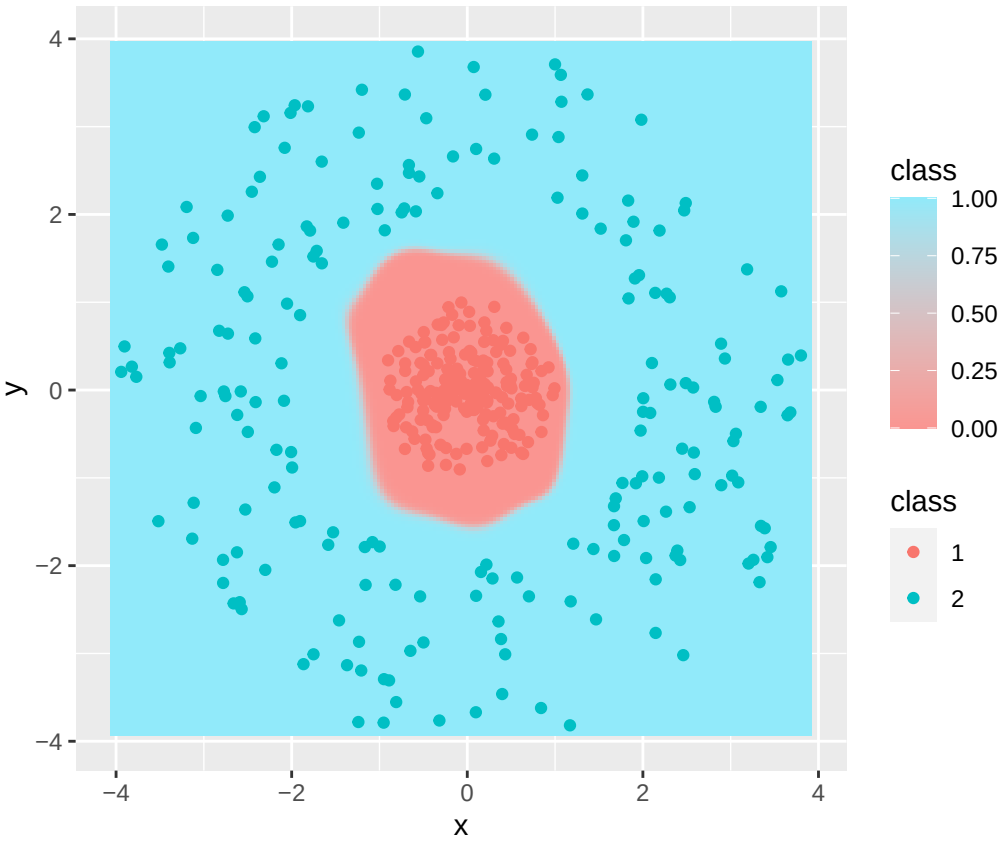
Tekintsünk egy olyan adathalmazt, amelyet lineáris döntéssel nem tudunk szétválasztani:
Ebben az esetben, láthatjuk, a egy mesterséges neuronnal rendelkező hálózatunk csődöt mond:

```
## # weights:  5
## initial  value 278.345351
## iter  10 value 212.562224
## iter  20 value 207.037295
## iter  30 value 205.934179
## iter  40 value 205.779620
## iter  50 value 205.769804
## iter  60 value 205.764937
## iter  70 value 205.742524
## final  value 205.737494
## converged
```



Ha azonban növeljük a rejtett rétegben található neuronok számát, az illesztés tetszőleges pontossággal megvalósítható (például 100 neuronra):

```
## # weights:  401
## initial  value 642.332936
## iter  10 value 0.883248
## iter  20 value 0.015457
## iter  30 value 0.004477
## iter  40 value 0.000873
## final  value 0.000067
## converged
```



11.2 Egyszerű példa

Tekintsük meg az étkezési olaj adathalmazunkra a neurális hálózatok segítségével az osztályozási feladat megoldását. Töltsük be az adathalmazt!

```
set.seed(1)

data(oil,package='caret')
df<-cbind(fattyAcids,oilType)
summary(df)
```

```
##     Palmitic        Stearic          Oleic          Linoleic
##  Min.   : 4.50   Min.   :1.700   Min.   :22.80   Min.   : 7.90
##  1st Qu.: 6.20   1st Qu.:3.475   1st Qu.:26.30   1st Qu.:43.10
##  Median : 9.85   Median :4.200   Median :30.70   Median :50.80
##  Mean   : 9.04   Mean   :4.200   Mean   :36.73   Mean   :46.49
##  3rd Qu.:11.12   3rd Qu.:5.000   3rd Qu.:38.62   3rd Qu.:58.08
##  Max.   :14.90   Max.   :6.700   Max.   :76.70   Max.   :66.10
##
##    Linolenic       Eicosanoic      Eicosenoic     oilType
##  Min.   :0.100   Min.   :0.100   Min.   :0.1000   A:37
##  1st Qu.:0.375   1st Qu.:0.100   1st Qu.:0.1000   B:26
##  Median :0.800   Median :0.400   Median :0.1000   C: 3
##  Mean   :2.272   Mean   :0.399   Mean   :0.3115   D: 7
##  3rd Qu.:2.650   3rd Qu.:0.400   3rd Qu.:0.3000   E:11
##  Max.   :9.500   Max.   :2.800   Max.   :1.8000   F:10
##                                                   G: 2
```

Válasszuk szét az adathalmazt tanító és tesztelő adathalmazra!

```
trainIdx<-createDataPartition(df$oilType, p=0.5, list=F)
trainDf<-df[trainIdx,]
testDf<-df[-trainIdx,]

summary(trainDf)
```

```
##     Palmitic         Stearic          Oleic          Linoleic
##  Min.   : 4.500   Min.   :1.700   Min.   :22.80   Min.   : 8.00
##  1st Qu.: 6.200   1st Qu.:3.375   1st Qu.:26.20   1st Qu.:43.75
##  Median :10.000   Median :4.200   Median :30.70   Median :51.30
##  Mean   : 9.068   Mean   :4.204   Mean   :37.04   Mean   :46.16
##  3rd Qu.:11.175   3rd Qu.:5.075   3rd Qu.:37.62   3rd Qu.:57.02
##  Max.   :13.000   Max.   :6.700   Max.   :75.80   Max.   :66.10
##
##    Linolenic       Eicosanoic      Eicosenoic    oilType
##  Min.   :0.100   Min.   :0.100   Min.   :0.100   A:19
##  1st Qu.:0.300   1st Qu.:0.100   1st Qu.:0.100   B:13
##  Median :0.800   Median :0.400   Median :0.100   C: 2
##  Mean   :2.192   Mean   :0.378   Mean   :0.318   D: 4
##  3rd Qu.:2.175   3rd Qu.:0.400   3rd Qu.:0.200   E: 6
##  Max.   :9.500   Max.   :1.500   Max.   :1.800   F: 5
##                                                  G: 1
```

```
summary(testDf)
```

```
##     Palmitic         Stearic          Oleic          Linoleic
##  Min.   : 4.800   Min.   :1.700   Min.   :23.10   Min.   : 7.90
##  1st Qu.: 6.200   1st Qu.:3.600   1st Qu.:26.43   1st Qu.:42.75
##  Median : 9.700   Median :4.200   Median :30.70   Median :50.15
##  Mean   : 9.009   Mean   :4.196   Mean   :36.39   Mean   :46.84
##  3rd Qu.:11.100   3rd Qu.:5.000   3rd Qu.:39.60   3rd Qu.:58.62
##  Max.   :14.900   Max.   :6.300   Max.   :76.70   Max.   :64.90
##
##    Linolenic       Eicosanoic       Eicosenoic     oilType
##  Min.   :0.100   Min.   :0.1000   Min.   :0.1000   A:18
##  1st Qu.:0.500   1st Qu.:0.1000   1st Qu.:0.1000   B:13
##  Median :0.850   Median :0.4000   Median :0.1000   C: 1
##  Mean   :2.359   Mean   :0.4217   Mean   :0.3043   D: 3
##  3rd Qu.:3.150   3rd Qu.:0.4750   3rd Qu.:0.3000   E: 5
##  Max.   :9.300   Max.   :2.8000   Max.   :1.6000   F: 5
##                                                    G: 1
```

```
pGrid<-expand.grid(size=seq(1,50,by = 3),decay=0)
model<-train(oilType~., trainDf, method='nnet', tuneGrid=pGrid, trace=F)
```

```
## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty
```

```
## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty
```

```
## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty
```

```
## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty
## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty
```

```
## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty
```

```
## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty
## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty
```

```
## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'G' is empty
## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty
```

```
## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty
```

```
## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty
```

```
## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty
```

```
## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : groups 'C' 'G' are empty
## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty
```

```
## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty

## Warning in nnet.formula(.outcome ~ ., data = dat, size = param$size, decay =
## param$decay, : group 'C' is empty
```

```
print(model)
```

```
## Neural Network
##
## 50 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 50, 50, 50, 50, 50, 50, ...
## Resampling results across tuning parameters:
##
##   size  Accuracy   Kappa
##    1    0.3626345  0.0424940
##    4    0.4999513  0.2909204
##    7    0.6081251  0.4676883
##   10    0.6178076  0.4800946
##   13    0.6934226  0.5874819
##   16    0.7929881  0.7274632
##   19    0.7677672  0.6872099
##   22    0.7814976  0.7113055
##   25    0.8097078  0.7498398
##   28    0.8041554  0.7410461
##   31    0.7958580  0.7336749
##   34    0.7884229  0.7232965
##   37    0.8369814  0.7848121
##   40    0.8162622  0.7603451
##   43    0.8309395  0.7811765
##   46    0.8216309  0.7653350
##   49    0.8149802  0.7605511
##
## Tuning parameter 'decay' was held constant at a value of 0
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 37 and decay = 0.
```

```
print(model$finalModel)
```

```
## a 7-37-7 network with 562 weights
## inputs: Palmitic Stearic Oleic Linoleic Linolenic Eicosanoic Eicosenoic
## output(s): .outcome
## options were - softmax modelling
```

```
pred<-predict(model, trainDf)
print(confusionMatrix(pred,trainDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 19  0  0  0  0  0  0
##          B  0 13  0  0  0  0  0
##          C  0  0  2  0  0  0  0
##          D  0  0  0  4  0  0  0
##          E  0  0  0  0  6  0  0
##          F  0  0  0  0  0  5  0
##          G  0  0  0  0  0  0  1
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9289, 1)
##     No Information Rate : 0.38
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity              1.00     1.00     1.00     1.00     1.00      1.0
## Specificity              1.00     1.00     1.00     1.00     1.00      1.0
## Pos Pred Value           1.00     1.00     1.00     1.00     1.00      1.0
## Neg Pred Value           1.00     1.00     1.00     1.00     1.00      1.0
## Prevalence               0.38     0.26     0.04     0.08     0.12      0.1
## Detection Rate           0.38     0.26     0.04     0.08     0.12      0.1
## Detection Prevalence     0.38     0.26     0.04     0.08     0.12      0.1
## Balanced Accuracy        1.00     1.00     1.00     1.00     1.00      1.0
##                      Class: G
## Sensitivity              1.00
## Specificity              1.00
## Pos Pred Value           1.00
## Neg Pred Value           1.00
## Prevalence               0.02
## Detection Rate           0.02
## Detection Prevalence     0.02
## Balanced Accuracy        1.00
```

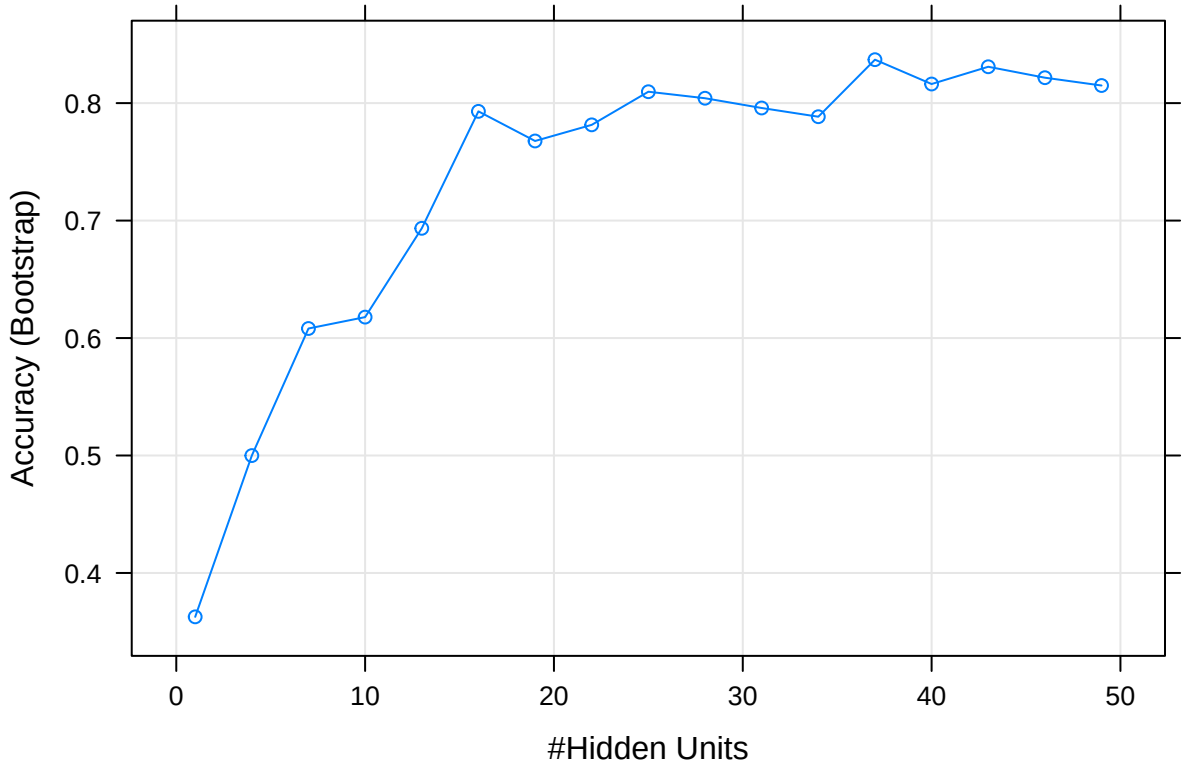
```
pred<-predict(model, testDf)
print(confusionMatrix(pred,testDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
```

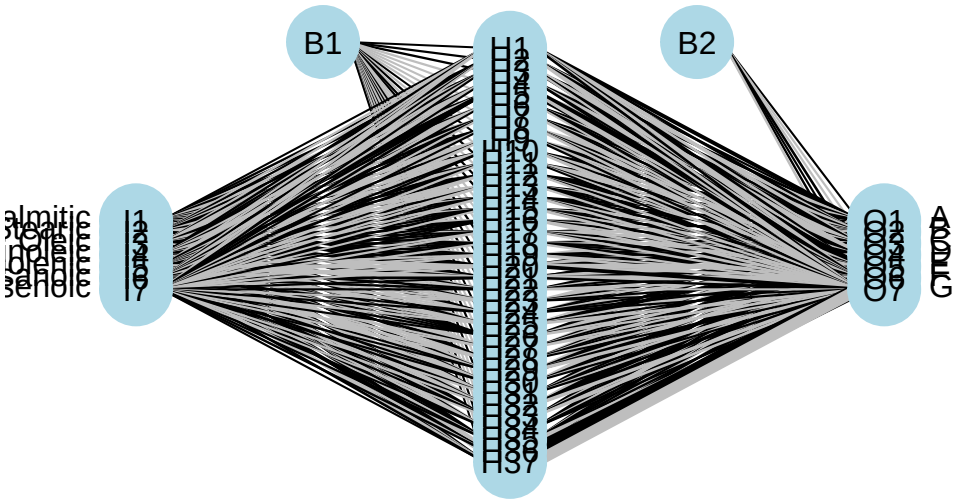
```
##          A 18  0  0  0  0  0  1
##          B  0 13  0  0  0  0  0
##          C  0  0  1  1  0  0  0
##          D  0  0  0  1  0  0  0
##          E  0  0  0  0  4  0  0
##          F  0  0  0  1  0  5  0
##          G  0  0  0  0  1  0  0
##
## Overall Statistics
##
##                Accuracy : 0.913
##                  95% CI : (0.7921, 0.9758)
##     No Information Rate : 0.3913
##     P-Value [Acc > NIR] : 1.829e-13
##
##                   Kappa : 0.8812
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            1.0000   1.0000  1.00000  0.33333  0.80000   1.0000
## Specificity            0.9643   1.0000  0.97778  1.00000  1.00000   0.9756
## Pos Pred Value         0.9474   1.0000  0.50000  1.00000  1.00000   0.8333
## Neg Pred Value         1.0000   1.0000  1.00000  0.95556  0.97619   1.0000
## Prevalence             0.3913   0.2826  0.02174  0.06522  0.10870   0.1087
## Detection Rate         0.3913   0.2826  0.02174  0.02174  0.08696   0.1087
## Detection Prevalence   0.4130   0.2826  0.04348  0.02174  0.08696   0.1304
## Balanced Accuracy      0.9821   1.0000  0.98889  0.66667  0.90000   0.9878
##                      Class: G
## Sensitivity           0.00000
## Specificity           0.97778
## Pos Pred Value        0.00000
## Neg Pred Value        0.97778
## Prevalence            0.02174
## Detection Rate        0.00000
## Detection Prevalence  0.02174
## Balanced Accuracy     0.48889
```

Érdemes megtekinteni a tanulási görbét, amelyen jól látszik, hogy a görbe a neuronok számával meredeken emelkedik, majd végül tetőzik, és enyhe csökkenésbe kezd:

```
plot(model)
```




```
plotnet(model$finalModel)
```



12 Együttes osztályozók

```
# Használt csomagok
library(caret)
library(randomForest)
library(xgboost)
```

12.1 Bevezetés

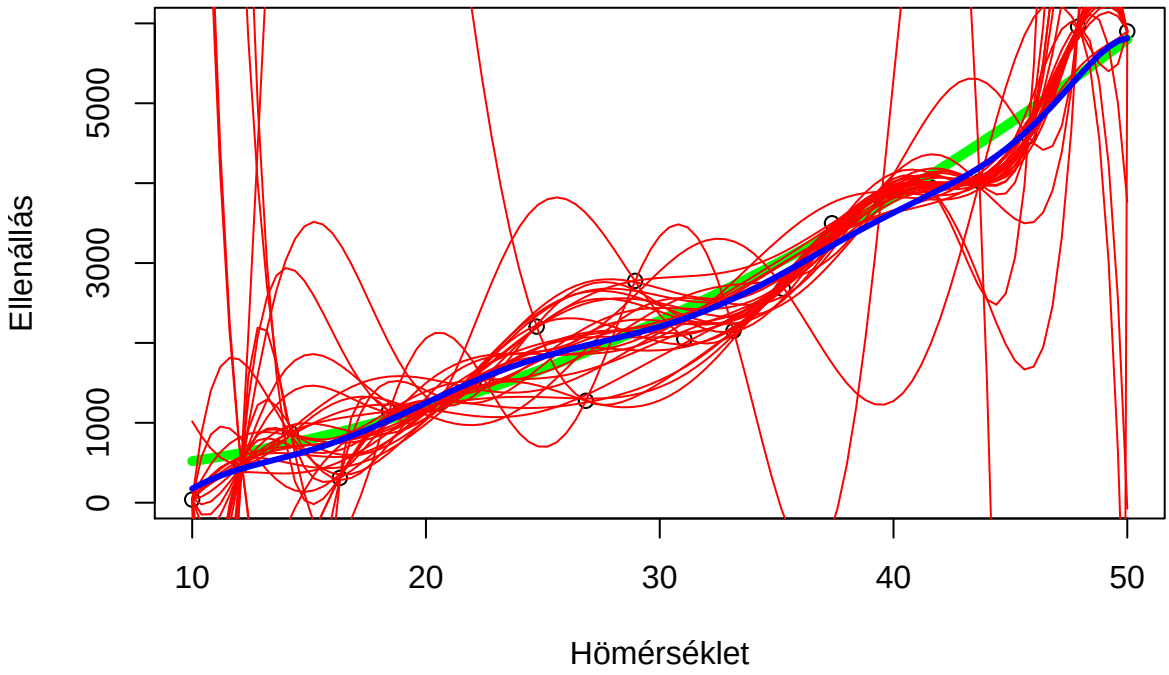
Általánosságban megfigyelhettük, hogy a statisztikai átlagolás lehetővé teszi számos következtetés javítását. Az együttes tanulók célja, hogy összetett algoritmusok helyett nagy számú gyenge tanulót (ún. *weak learner*) alkalmazva, és azok kimenetének átlagát (osztályozás esetén pl. a többségi voksolás segítségével) használjuk predikciónak. Teljesség kedvéért megemlítjük, hogy a módszer nem csak osztályozásra, de regresszióra is alkalmas.

12.2 Modell átlagolás (haladó)

A modell átlagolás módszere közvetlen következménye a statisztika egyik alapvető elvének, a bias-variancia kompromisszumnak (bias-variance tradeoff). Számos független modell átlagolása az átlagos hibát a modellek számával fordítottan arányosan javítja. Fontos megjegyezni, hogy az egyes modellek közötti függetlenség biztosítása kiemelt fontosságú feladat.

Tekintsük a modellválasztás során bemutatott példát! Tételezzük fel, hogy 20 adatpontunk van, amelyekből 30 darab bootstrap adathalmazt generálunk. Az egyes bootstrap halmazokra 9-ed fokú polinomokat illesztük. Az egyes polinomok erősen túlilleszettek lesznek, ahogy az ábrán a piros vonalak ezt jelzik. Az egyes modellek statisztikai kiátlagolásával a kék vonalat kapjuk, amely kielégítően közelíti a valós folyamatot (zöld). Érdemes megjegyezni, hogy a bootstrap adathalmazok nem teljesen függetlenek, ugyanakkor az átlagolás az eredetileg erősen túlillesztett modelleken jelentősen javított.

```
set.seed(3)
fn<-function(t) {
  2.2*t^2 + 300
}

x<-seq(10,50,length.out=20)
y<-fn(x) + rnorm(length(x),sd=500)
df<-data.frame(x=x,y=y)
plot(df,xlab='Hömérséklet',ylab='Ellenállás')
curve(fn,add=T,col='green',lwd=5)

res<-sapply(1:30,function(x) {
  train<-df[sample(nrow(df),replace = T),]

  m<-lm(y~poly(x,9),train)
  curve(predict(m,data.frame(x=x)),add=T,col='red')

  m$coefficients
})

m<-lm(y~poly(x,9),df)
m$coefficients<-rowMeans(res)
curve(predict(m,data.frame(x=x)),add=T,col='blue',lwd=3)
```



12.3 Véletlen erdők

A véletlen erdők (*random forest*) gyenge tanuló osztályozónak mély döntési fákat alkalmaznak. A mély döntési fák önmagukban teljes mélységig tanításra kerülnek, így az egyes fák esetén a túlillesztéssel nem foglalkozik az algoritmus. A véletlen erdők két kulcsfontosságú alapelvet alkalmaznak:

- A véletlen erdő algoritmus N darab fát tanítanak be, N darab bootstrap mintahalmazon. Ezt a módszert *bootstrap aggregation*-nek, vagy *bagging*-nek hívjuk.
- Az egyes fák építése során, minden faelágazásnál *véletlen attribútumhalmazon* döntik el az elágazási feltételt. Ez azért fontos, mert így az egyes fák nagyobb eséllyel lesznek függetlenek.

A véletlen erdő algoritmus a predikció során többségi voksolást alkalmaz, vagy valószínűségi predikció esetén az egyes fák kimenetét átlagolják.

Tekintsük az `iris` adathalmazt, melyből eltávolítjuk a `setosa` fajt. Erre a bináris osztályozásra látjuk $n = 100$ fára a véletlen erdő algoritmus döntési határait:

12.4 Gradient boosting

Napjain egyik leghatékonyabb adatelemzési és predikciós algoritmusa az ún. *gradient boosted tree*, vagy annak is egy speciális megvalósítása, az ún. *XGBoost*. A módszer maga egy együttes osztályozót takar, amely az ún. *boost* módszert használja hatékonyabb tanuló algoritmus megalkotására. Az XGBoost tetszőleges gyenge tanuló megoldás alkalmazására képes, ugyanakkor itt csak a fa alapú megvalósítást tekintjük.

A boosted tree algoritmus lényege, hogy a kiépített fáinkat iteratív módszerrel javítjuk, tehát a hibásan osztályozott adatsorok nagyobb súlyt kapnak a következő iterációban. A módszer működéséről lásd. a következő ábrát:

Tekintsük az `iris` adathalmazt, melyből eltávolítjuk a `setosa` fajt. Erre a bináris osztályozásra látjuk $n = 200$ fára a véletlen erdő algoritmus döntési határait:

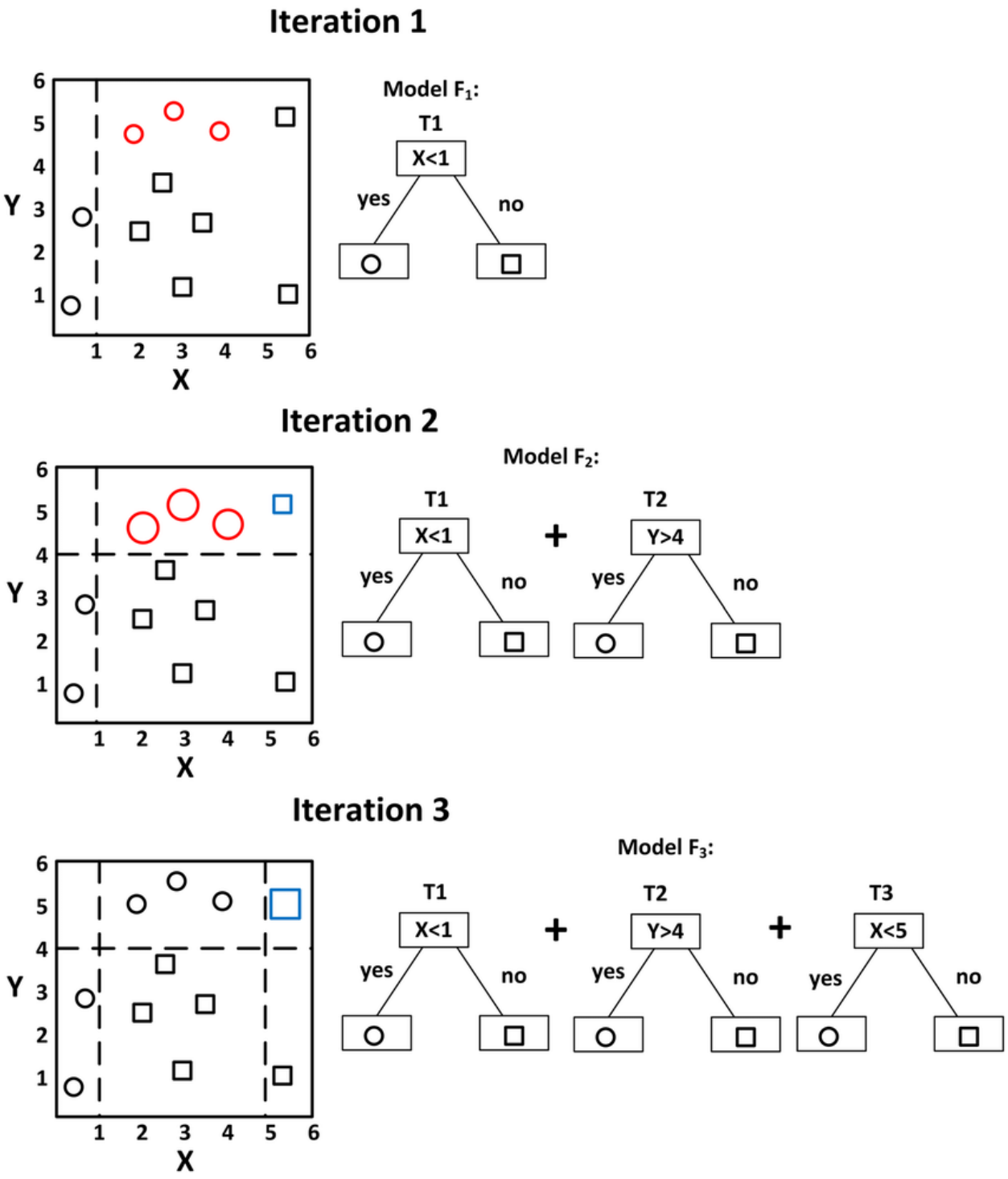
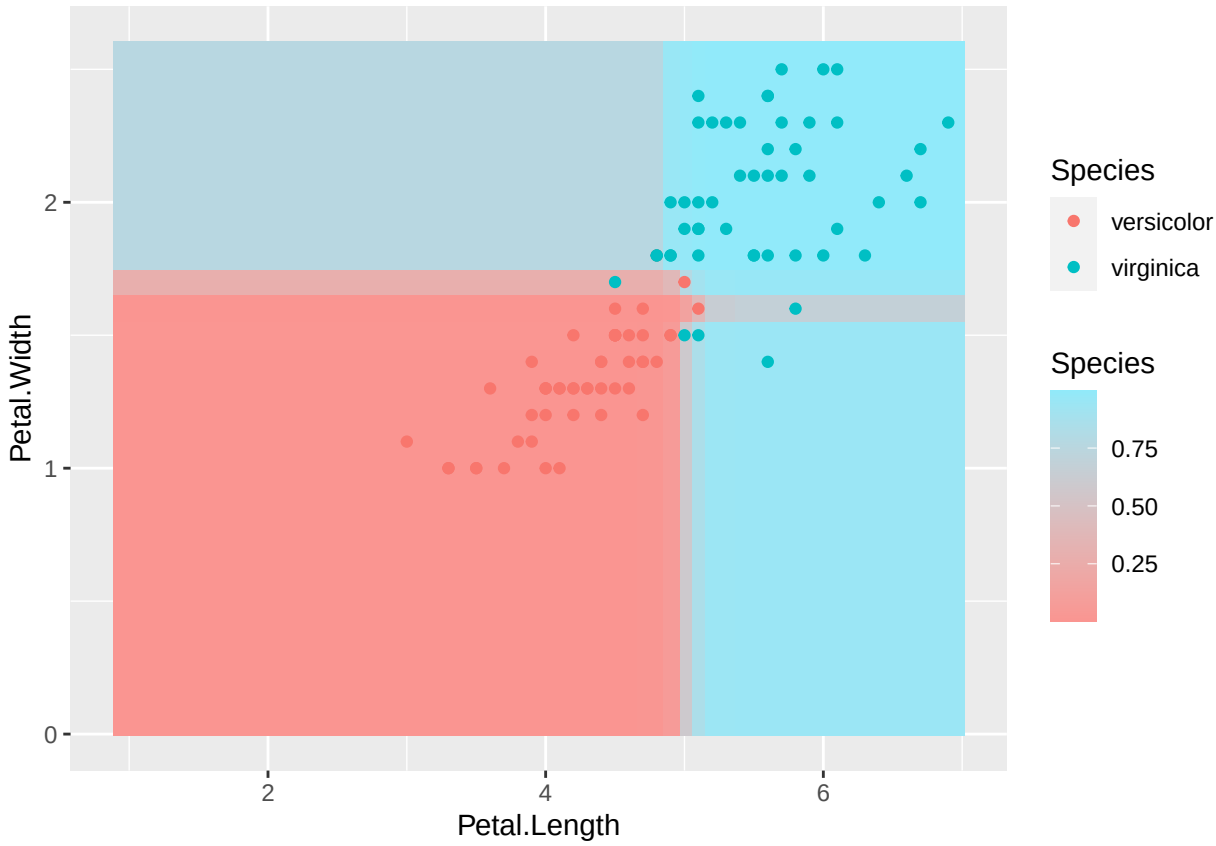


Figure 4: A boosted tree algoritmus



12.5 Egyszerű példa

Vegyük az étkezési olajokat tartalmazó adathalmazt!

```
set.seed(1)

data(oil)
df<-cbind(fattyAcids,oilType)
summary(df)
```

```
##     Palmitic        Stearic          Oleic          Linoleic
##  Min.   : 4.50   Min.   :1.700   Min.   :22.80   Min.   : 7.90
##  1st Qu.: 6.20   1st Qu.:3.475   1st Qu.:26.30   1st Qu.:43.10
##  Median : 9.85   Median :4.200   Median :30.70   Median :50.80
##  Mean   : 9.04   Mean   :4.200   Mean   :36.73   Mean   :46.49
##  3rd Qu.:11.12   3rd Qu.:5.000   3rd Qu.:38.62   3rd Qu.:58.08
##  Max.   :14.90   Max.   :6.700   Max.   :76.70   Max.   :66.10
##
##    Linolenic       Eicosanoic      Eicosenoic     oilType
##  Min.   :0.100   Min.   :0.100   Min.   :0.1000   A:37
##  1st Qu.:0.375   1st Qu.:0.100   1st Qu.:0.1000   B:26
##  Median :0.800   Median :0.400   Median :0.1000   C: 3
##  Mean   :2.272   Mean   :0.399   Mean   :0.3115   D: 7
##  3rd Qu.:2.650   3rd Qu.:0.400   3rd Qu.:0.3000   E:11
##  Max.   :9.500   Max.   :2.800   Max.   :1.8000   F:10
##                                                   G: 2
```

Válasszuk szét az adathalmazt tanító és tesztelő adathalmazra!

```
trainIdx<-createDataPartition(df$oilType, p=0.5, list=F)
trainDf<-df[trainIdx,]
testDf<-df[-trainIdx,]

summary(trainDf)
```

```
##     Palmitic         Stearic          Oleic          Linoleic
##  Min.   : 4.500   Min.   :1.700   Min.   :22.80   Min.   : 8.00
##  1st Qu.: 6.200   1st Qu.:3.375   1st Qu.:26.20   1st Qu.:43.75
##  Median :10.000   Median :4.200   Median :30.70   Median :51.30
##  Mean   : 9.068   Mean   :4.204   Mean   :37.04   Mean   :46.16
##  3rd Qu.:11.175   3rd Qu.:5.075   3rd Qu.:37.62   3rd Qu.:57.02
##  Max.   :13.000   Max.   :6.700   Max.   :75.80   Max.   :66.10
##
##    Linolenic       Eicosanoic      Eicosenoic    oilType
##  Min.   :0.100   Min.   :0.100   Min.   :0.100   A:19
##  1st Qu.:0.300   1st Qu.:0.100   1st Qu.:0.100   B:13
##  Median :0.800   Median :0.400   Median :0.100   C: 2
##  Mean   :2.192   Mean   :0.378   Mean   :0.318   D: 4
##  3rd Qu.:2.175   3rd Qu.:0.400   3rd Qu.:0.200   E: 6
##  Max.   :9.500   Max.   :1.500   Max.   :1.800   F: 5
##                                                  G: 1
```

```
summary(testDf)
```

```
##     Palmitic         Stearic          Oleic          Linoleic
##  Min.   : 4.800   Min.   :1.700   Min.   :23.10   Min.   : 7.90
##  1st Qu.: 6.200   1st Qu.:3.600   1st Qu.:26.43   1st Qu.:42.75
##  Median : 9.700   Median :4.200   Median :30.70   Median :50.15
##  Mean   : 9.009   Mean   :4.196   Mean   :36.39   Mean   :46.84
##  3rd Qu.:11.100   3rd Qu.:5.000   3rd Qu.:39.60   3rd Qu.:58.62
##  Max.   :14.900   Max.   :6.300   Max.   :76.70   Max.   :64.90
##
##    Linolenic       Eicosanoic       Eicosenoic     oilType
##  Min.   :0.100   Min.   :0.1000   Min.   :0.1000   A:18
##  1st Qu.:0.500   1st Qu.:0.1000   1st Qu.:0.1000   B:13
##  Median :0.850   Median :0.4000   Median :0.1000   C: 1
##  Mean   :2.359   Mean   :0.4217   Mean   :0.3043   D: 3
##  3rd Qu.:3.150   3rd Qu.:0.4750   3rd Qu.:0.3000   E: 5
##  Max.   :9.300   Max.   :2.8000   Max.   :1.6000   F: 5
##                                                    G: 1
```

Futtassuk le a véletlen erdő algoritmust! Minden paramétert tartsunk alapállapotban:

```
model<-train(oilType~., trainDf, method='rf', trControl = trainControl(sampling='up'))
```

```
## Warning: model fit failed for Resample01: mtry=2 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample01: mtry=4 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample01: mtry=7 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample04: mtry=2 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.
```

```
## Warning: model fit failed for Resample04: mtry=4 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample04: mtry=7 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample08: mtry=2 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample08: mtry=4 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample08: mtry=7 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample10: mtry=2 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample10: mtry=4 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample10: mtry=7 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample11: mtry=2 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample11: mtry=4 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample11: mtry=7 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample13: mtry=2 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample13: mtry=4 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample13: mtry=7 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample17: mtry=2 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample17: mtry=4 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample17: mtry=7 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample19: mtry=2 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample19: mtry=4 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample19: mtry=7 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample20: mtry=2 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.
```

```
## Warning: model fit failed for Resample20: mtry=4 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample20: mtry=7 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample21: mtry=2 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample21: mtry=4 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample21: mtry=7 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample23: mtry=2 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample23: mtry=4 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning: model fit failed for Resample23: mtry=7 Error in randomForest.default(x, y, mtry = param$mtr
##   Can't have empty classes in y.

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
print(model)
```

```
## Random Forest
##
## 50 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 50, 50, 50, 50, 50, 50, ...
## Addtional sampling using up-sampling
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.9686083  0.9568001
##   4     0.9725766  0.9628081
##   7     0.9725766  0.9628081
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 4.
```

```
print(model$finalModel)
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
```

```
##         OOB estimate of  error rate: 0.75%
## Confusion matrix:
##    A  B  C  D  E  F  G class.error
## A 18  0  0  0  1  0  0  0.05263158
## B  0 19  0  0  0  0  0  0.00000000
## C  0  0 19  0  0  0  0  0.00000000
## D  0  0  0 19  0  0  0  0.00000000
## E  0  0  0  0 19  0  0  0.00000000
## F  0  0  0  0  0 19  0  0.00000000
## G  0  0  0  0  0  0 19  0.00000000
```

```
pred<-predict(model, trainDf)
print(confusionMatrix(pred,trainDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 19  0  0  0  0  0  0
##          B  0 13  0  0  0  0  0
##          C  0  0  2  0  0  0  0
##          D  0  0  0  4  0  0  0
##          E  0  0  0  0  6  0  0
##          F  0  0  0  0  0  5  0
##          G  0  0  0  0  0  0  1
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9289, 1)
##     No Information Rate : 0.38
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity              1.00     1.00     1.00     1.00     1.00      1.0
## Specificity              1.00     1.00     1.00     1.00     1.00      1.0
## Pos Pred Value           1.00     1.00     1.00     1.00     1.00      1.0
## Neg Pred Value           1.00     1.00     1.00     1.00     1.00      1.0
## Prevalence               0.38     0.26     0.04     0.08     0.12      0.1
## Detection Rate           0.38     0.26     0.04     0.08     0.12      0.1
## Detection Prevalence     0.38     0.26     0.04     0.08     0.12      0.1
## Balanced Accuracy        1.00     1.00     1.00     1.00     1.00      1.0
##                      Class: G
## Sensitivity              1.00
## Specificity              1.00
## Pos Pred Value           1.00
## Neg Pred Value           1.00
## Prevalence               0.02
## Detection Rate           0.02
```

```
## Detection Prevalence     0.02
## Balanced Accuracy        1.00
```

```
pred<-predict(model, testDf)
print(confusionMatrix(pred,testDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 18  0  0  0  0  0  0
##          B  0 13  0  0  0  0  0
##          C  0  0  1  0  0  0  0
##          D  0  0  0  2  0  0  0
##          E  0  0  0  0  5  0  0
##          F  0  0  0  0  0  5  0
##          G  0  0  0  1  0  0  1
##
## Overall Statistics
##
##                Accuracy : 0.9783
##                  95% CI : (0.8847, 0.9994)
##     No Information Rate : 0.3913
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9706
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            1.0000   1.0000  1.00000  0.66667   1.0000   1.0000
## Specificity            1.0000   1.0000  1.00000  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000  1.00000  1.00000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000  1.00000  0.97727   1.0000   1.0000
## Prevalence             0.3913   0.2826  0.02174  0.06522   0.1087   0.1087
## Detection Rate         0.3913   0.2826  0.02174  0.04348   0.1087   0.1087
## Detection Prevalence   0.3913   0.2826  0.02174  0.04348   0.1087   0.1087
## Balanced Accuracy      1.0000   1.0000  1.00000  0.83333   1.0000   1.0000
##                      Class: G
## Sensitivity           1.00000
## Specificity           0.97778
## Pos Pred Value        0.50000
## Neg Pred Value        1.00000
## Prevalence            0.02174
## Detection Rate        0.02174
## Detection Prevalence  0.04348
## Balanced Accuracy     0.98889
```

Praktikus, hogy az erdőben meg tudjuk tekinteni, hogy az egyes fákban a csomópontok elágazásánál átlagosan az egyes változók mekkora súllyal szerepelnek. Ezt az ún. `importance` függvénnyel tudjuk megtenni:

```
importance(model$finalModel)
```

```
##            MeanDecreaseGini
```

```
## Palmitic          13.801246
## Stearic           27.064914
## Oleic             16.449609
## Linoleic          23.689630
## Linolenic         17.073416
## Eicosanoic        10.059553
## Eicosenoic         4.994113
```

Futtassuk le az XGBoost algoritmust! Minden paramétert tartsunk alapállapotban:

```
model<-train(oilType~., trainDf, method='xgbTree', trControl = trainControl(sampling='up'))
print(model)
```

```
## eXtreme Gradient Boosting
##
## 50 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 50, 50, 50, 50, 50, 50, ...
## Addtional sampling using up-sampling
##
## Resampling results across tuning parameters:
##
##   eta  max_depth  colsample_bytree  subsample  nrounds  Accuracy   Kappa
##   0.3  1          0.6               0.50        50      0.9102601  0.8839504
##   0.3  1          0.6               0.50       100      0.9054071  0.8769412
##   0.3  1          0.6               0.50       150      0.9054071  0.8768690
##   0.3  1          0.6               0.75        50      0.9001007  0.8710503
##   0.3  1          0.6               0.75       100      0.9027673  0.8742946
##   0.3  1          0.6               0.75       150      0.9027673  0.8742553
##   0.3  1          0.6               1.00        50      0.9132960  0.8867962
##   0.3  1          0.6               1.00       100      0.9132960  0.8867231
##   0.3  1          0.6               1.00       150      0.9132960  0.8867231
##   0.3  1          0.8               0.50        50      0.9042791  0.8739369
##   0.3  1          0.8               0.50       100      0.9042791  0.8739223
##   0.3  1          0.8               0.50       150      0.9065013  0.8785879
##   0.3  1          0.8               0.75        50      0.9019071  0.8729210
##   0.3  1          0.8               0.75       100      0.9001680  0.8702799
##   0.3  1          0.8               0.75       150      0.9001680  0.8705859
##   0.3  1          0.8               1.00        50      0.9069778  0.8798479
##   0.3  1          0.8               1.00       100      0.9069778  0.8797757
##   0.3  1          0.8               1.00       150      0.9052387  0.8774407
##   0.3  2          0.6               0.50        50      0.9109374  0.8818292
##   0.3  2          0.6               0.50       100      0.9109374  0.8822664
##   0.3  2          0.6               0.50       150      0.9127556  0.8846768
##   0.3  2          0.6               0.75        50      0.9105373  0.8825224
##   0.3  2          0.6               0.75       100      0.9132040  0.8860727
##   0.3  2          0.6               0.75       150      0.9132040  0.8860727
##   0.3  2          0.6               1.00        50      0.8983060  0.8674118
##   0.3  2          0.6               1.00       100      0.9001241  0.8697710
##   0.3  2          0.6               1.00       150      0.9001241  0.8697710
##   0.3  2          0.8               0.50        50      0.9045209  0.8740884
```

```
##   0.3  2          0.8              0.50        100       0.9024157  0.8715460
##   0.3  2          0.8              0.50        150       0.9024157  0.8716256
##   0.3  2          0.8              0.75         50       0.9137405  0.8865441
##   0.3  2          0.8              0.75        100       0.9157405  0.8891423
##   0.3  2          0.8              0.75        150       0.9157405  0.8891752
##   0.3  2          0.8              1.00         50       0.9022059  0.8729411
##   0.3  2          0.8              1.00        100       0.9022059  0.8729411
##   0.3  2          0.8              1.00        150       0.9022059  0.8729411
##   0.3  3          0.6              0.50         50       0.9099071  0.8813757
##   0.3  3          0.6              0.50        100       0.9075542  0.8783865
##   0.3  3          0.6              0.50        150       0.9125738  0.8847804
##   0.3  3          0.6              0.75         50       0.8967086  0.8631073
##   0.3  3          0.6              0.75        100       0.8967086  0.8633237
##   0.3  3          0.6              0.75        150       0.8943557  0.8603315
##   0.3  3          0.6              1.00         50       0.9152960  0.8890996
##   0.3  3          0.6              1.00        100       0.9152960  0.8890096
##   0.3  3          0.6              1.00        150       0.9152960  0.8890996
##   0.3  3          0.8              0.50         50       0.9077735  0.8801191
##   0.3  3          0.8              0.50        100       0.9053694  0.8771420
##   0.3  3          0.8              0.50        150       0.9072387  0.8786358
##   0.3  3          0.8              0.75         50       0.9135569  0.8863375
##   0.3  3          0.8              0.75        100       0.9135569  0.8862879
##   0.3  3          0.8              0.75        150       0.9113347  0.8824031
##   0.3  3          0.8              1.00         50       0.9090451  0.8806092
##   0.3  3          0.8              1.00        100       0.9090451  0.8806092
##   0.3  3          0.8              1.00        150       0.9090451  0.8806092
##   0.4  1          0.6              0.50         50       0.9008398  0.8695489
##   0.4  1          0.6              0.50        100       0.8990216  0.8671495
##   0.4  1          0.6              0.50        150       0.9008398  0.8696798
##   0.4  1          0.6              0.75         50       0.9060241  0.8783053
##   0.4  1          0.6              0.75        100       0.9038019  0.8743708
##   0.4  1          0.6              0.75        150       0.9038019  0.8743708
##   0.4  1          0.6              1.00         50       0.9099054  0.8836024
##   0.4  1          0.6              1.00        100       0.9099054  0.8836024
##   0.4  1          0.6              1.00        150       0.9099054  0.8836024
##   0.4  1          0.8              0.50         50       0.9025544  0.8736275
##   0.4  1          0.8              0.50        100       0.9021597  0.8730426
##   0.4  1          0.8              0.50        150       0.9021597  0.8730426
##   0.4  1          0.8              0.75         50       0.9006983  0.8713132
##   0.4  1          0.8              0.75        100       0.9006983  0.8713132
##   0.4  1          0.8              0.75        150       0.9006983  0.8713132
##   0.4  1          0.8              1.00         50       0.9019930  0.8730551
##   0.4  1          0.8              1.00        100       0.8999930  0.8704323
##   0.4  1          0.8              1.00        150       0.8999930  0.8704577
##   0.4  2          0.6              0.50         50       0.9094292  0.8807983
##   0.4  2          0.6              0.50        100       0.9094292  0.8811238
##   0.4  2          0.6              0.50        150       0.9094292  0.8811238
##   0.4  2          0.6              0.75         50       0.8999668  0.8684848
##   0.4  2          0.6              0.75        100       0.9017850  0.8711345
##   0.4  2          0.6              0.75        150       0.8997850  0.8683567
##   0.4  2          0.6              1.00         50       0.9017100  0.8720146
##   0.4  2          0.6              1.00        100       0.9035282  0.8743666
##   0.4  2          0.6              1.00        150       0.9035282  0.8743666
##   0.4  2          0.8              0.50         50       0.9120740  0.8855026
```

```
##   0.4  2          0.8              0.50       100      0.9141793  0.8891839
##   0.4  2          0.8              0.50       150      0.9097211  0.8828090
##   0.4  2          0.8              0.75        50      0.9039071  0.8746766
##   0.4  2          0.8              0.75       100      0.9014071  0.8712357
##   0.4  2          0.8              0.75       150      0.9039071  0.8746766
##   0.4  2          0.8              1.00        50      0.9046908  0.8770290
##   0.4  2          0.8              1.00       100      0.9046908  0.8770290
##   0.4  2          0.8              1.00       150      0.9028726  0.8746047
##   0.4  3          0.6              0.50        50      0.9097652  0.8808459
##   0.4  3          0.6              0.50       100      0.9097652  0.8808459
##   0.4  3          0.6              0.50       150      0.9097652  0.8808459
##   0.4  3          0.6              0.75        50      0.9054109  0.8748688
##   0.4  3          0.6              0.75       100      0.9054109  0.8748912
##   0.4  3          0.6              0.75       150      0.9054109  0.8748912
##   0.4  3          0.6              1.00        50      0.9045235  0.8744741
##   0.4  3          0.6              1.00       100      0.9023013  0.8710025
##   0.4  3          0.6              1.00       150      0.9023013  0.8710025
##   0.4  3          0.8              0.50        50      0.8970930  0.8637146
##   0.4  3          0.8              0.50       100      0.9009112  0.8690460
##   0.4  3          0.8              0.50       150      0.8968060  0.8633837
##   0.4  3          0.8              0.75        50      0.9069778  0.8783292
##   0.4  3          0.8              0.75       100      0.9094778  0.8810011
##   0.4  3          0.8              0.75       150      0.9069778  0.8784032
##   0.4  3          0.8              1.00        50      0.9077541  0.8786232
##   0.4  3          0.8              1.00       100      0.9077541  0.8786232
##   0.4  3          0.8              1.00       150      0.9077541  0.8786232
##
## Tuning parameter 'gamma' was held constant at a value of 0
## Tuning
##  parameter 'min_child_weight' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were nrounds = 100, max_depth = 2, eta
##  = 0.3, gamma = 0, colsample_bytree = 0.8, min_child_weight = 1 and subsample
##  = 0.75.
```

```
print(model$finalModel)
```

```
## ##### xgb.Booster
## raw: 318.2 Kb
## call:
##   xgboost::xgb.train(params = list(eta = param$eta, max_depth = param$max_depth,
##     gamma = param$gamma, colsample_bytree = param$colsample_bytree,
##     min_child_weight = param$min_child_weight, subsample = param$subsample),
##     data = x, nrounds = param$nrounds, num_class = length(lev),
##     objective = "multi:softprob")
## params (as set within xgb.train):
##   eta = "0.3", max_depth = "2", gamma = "0", colsample_bytree = "0.8", min_child_weight = "1", subsa
## xgb.attributes:
##   niter
## callbacks:
##   cb.print.evaluation(period = print_every_n)
## # of features: 7
## niter: 100
## nfeatures : 7
## xNames : Palmitic Stearic Oleic Linoleic Linolenic Eicosanoic Eicosenoic
```

```
## problemType : Classification
## tuneValue :
##     nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
## 32     100         2 0.3     0              0.8                1      0.75
## obsLevels : A B C D E F G
## param :
##  list()
```

```
pred<-predict(model, trainDf)
print(confusionMatrix(pred,trainDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 19  0  0  0  0  0  0
##          B  0 13  0  0  0  0  0
##          C  0  0  2  0  0  0  0
##          D  0  0  0  4  0  0  0
##          E  0  0  0  0  6  0  0
##          F  0  0  0  0  0  5  0
##          G  0  0  0  0  0  0  1
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9289, 1)
##     No Information Rate : 0.38
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity              1.00     1.00     1.00     1.00     1.00      1.0
## Specificity              1.00     1.00     1.00     1.00     1.00      1.0
## Pos Pred Value           1.00     1.00     1.00     1.00     1.00      1.0
## Neg Pred Value           1.00     1.00     1.00     1.00     1.00      1.0
## Prevalence               0.38     0.26     0.04     0.08     0.12      0.1
## Detection Rate           0.38     0.26     0.04     0.08     0.12      0.1
## Detection Prevalence     0.38     0.26     0.04     0.08     0.12      0.1
## Balanced Accuracy        1.00     1.00     1.00     1.00     1.00      1.0
##                      Class: G
## Sensitivity              1.00
## Specificity              1.00
## Pos Pred Value           1.00
## Neg Pred Value           1.00
## Prevalence               0.02
## Detection Rate           0.02
## Detection Prevalence     0.02
## Balanced Accuracy        1.00
```

```
pred<-predict(model, testDf)
print(confusionMatrix(pred,testDf$oilType))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 16  0  0  0  0  0  0
##          B  0 13  0  0  0  0  0
##          C  0  0  1  0  0  0  0
##          D  0  0  0  2  0  0  0
##          E  0  0  0  0  5  0  0
##          F  0  0  0  0  0  4  0
##          G  2  0  0  1  0  1  1
##
## Overall Statistics
##
##                Accuracy : 0.913
##                  95% CI : (0.7921, 0.9758)
##     No Information Rate : 0.3913
##     P-Value [Acc > NIR] : 1.829e-13
##
##                   Kappa : 0.8851
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.8889   1.0000  1.00000  0.66667   1.0000  0.80000
## Specificity            1.0000   1.0000  1.00000  1.00000   1.0000  1.00000
## Pos Pred Value         1.0000   1.0000  1.00000  1.00000   1.0000  1.00000
## Neg Pred Value         0.9333   1.0000  1.00000  0.97727   1.0000  0.97619
## Prevalence             0.3913   0.2826  0.02174  0.06522   0.1087  0.10870
## Detection Rate         0.3478   0.2826  0.02174  0.04348   0.1087  0.08696
## Detection Prevalence   0.3478   0.2826  0.02174  0.04348   0.1087  0.08696
## Balanced Accuracy      0.9444   1.0000  1.00000  0.83333   1.0000  0.90000
##                      Class: G
## Sensitivity           1.00000
## Specificity           0.91111
## Pos Pred Value        0.20000
## Neg Pred Value        1.00000
## Prevalence            0.02174
## Detection Rate        0.02174
## Detection Prevalence  0.10870
## Balanced Accuracy     0.95556
```

Az egyes változók fontossága, hasonlóan a véletlen erdőkhöz:

```
xgb.importance(model=model$finalModel)
```

```
##       Feature        Gain      Cover  Frequency
## 1:    Stearic 0.244130595 0.20738355 0.17241379
## 2:      Oleic 0.211414689 0.21958305 0.16748768
## 3:   Palmitic 0.181066027 0.15567479 0.18719212
```

```
## 4: Eicosanoic 0.155070198 0.13725996 0.10837438
## 5:  Linolenic 0.106491101 0.10594952 0.11822660
## 6:   Linoleic 0.092507785 0.14708166 0.20689655
## 7: Eicosenoic 0.009319606 0.02706746 0.03940887
```

13 Gyakorló feladatok

Adathalmazok:

1. `iris` adathalmaz, beépített.
2. `scat` adathalmaz, széklet minta három állatfajtól, 19 változó. Megtalálható a `caret` csomagban.
3. `Default` adathalmaz, az `ISLR` csomagban

Feladatok:

- Végezze el a `iris` adathalmazban a `Species` változó one-hot és dummy kódolását! Végezzük el a kódolás a R alapvető függvényeivel, és egy tetszőlegesen választott csomaggal is!
- Végezzen lineáris regressziót a `Default` adathalmazra a `caret` csomag segítségével, a `balance` célváltozóra. Értékelje a regressziót! Lehetséges a `balance` változó becslése? (Segítség: a lineáris regresszió a `method='lm'` módszerrel érhető el, valamint figyeljünk oda a dummy változók létrehozására (a faktor változók esetén!))
- A legtöbb osztályozó algoritmus a `caret` csomagban a pontosságot (accuracy) optimalizálja a validáció során. Probáljuk meg a szenzitivitást, vagy a kiegyensúlyozott pontosságot optimalizálni! Imserjük meg a `train()` függvény `metric` argumentumát!
- A `scat` adathalmazban csináljon egyszerű osztályozási szabályt a következők szerint: ha az átmérő (diameter) nagyobb, mint 25, akkor prérifarkas (coyote), ha kisebb, mint 25, de nagyobb mint 15, akkor hiúz (bobcat), egyébként róka (gray_fox). Milyen az osztályozás pontossága (accuracy)?
- Készítse el az előző feladatban létrehozott osztályozóra a keveredési mátrixot! A mátrixot a gyakorlás érdekében számítsuk ki az R beépített megoldásaival (pl. table), majd a `caret` csomag segítségével is. A mátrix alapján számítsuk ki a pontosságot, a szenzitivitást és a specifitást is! A két megoldás összehasonlításával ellenőrizzük a megoldásunkat!
- Készítese el a `scat` adathalmaz kiegyenlített változatát. Végezze el a feladat alul- és túlmintavételezéssel is!
- Ábrázolja a `scat` adathalmaz jellemzőit kontingencia táblázat vagy sűrűségábrák segítségével! Lát olyan jellemzőt, ami hatékonyan elősegítené az osztályozást? Végezze el az ábrázolást az R beépített megoldásaival és a `caret` csomag `featurePlot` függvényének segítségével is!
- A `scat` adathalmazban osztályozása két dimenzióban. A következő feladatokhoz nem kötelező az adathalmaz szétbontása teszt adathalmazra.
  - Távolítsa el azokat a sorokat, ahol a `d15N` vagy a `Diameter` változóban `NA` érték található!
  - Ábrázolja két dimenzióban az egyes fajokat, a `d15N` és `Diameter` változó terében! Az egyes fajokat színnel jelölje!
  - Készítsen egy döntési fát a `d15N` és a `Diameter` változókra! Állapítsa meg az osztályozás minőségét keveredési mátrix és különböző mérőszámok segítségével!
  - Ábrázolja a döntési fát!
  - (Opcionális) Ábrázolja a döntési határokat a `d15N` és `Diameter` változó terében! (Tipp: használjuk a ggplot `geom_tile()` megoldását!)
- Mit gondol, a `scat` adathalmaz esetén az osztályozáshoz célszerű az összes változót használni, vagy némelyiket célszerű elhagyni? Ez a feladat alkalmas az adathalmazban történő elmélyedésre. Segítségnek a következőket jegyezzük meg:
  - Célszerű megvizsgálni, hogy például a hónaptól függenek-e a mérések, például az átmérő vagy a súly. Ehhez célszerű a hónapokat újrafaktorálni, azaz sorrendbe rakni a hónapok szerint (azaz, január-február, stb., pl. `ordered()`). `boxplot()` segítségével megvizsgálhatjuk az összefüggést.
  - Hasznos megvizsgálni a minta helyszínét, vajon mutat-e összefüggést, korrelációt az adatokkal.
- Végezze el a `scat` adathalmaz osztályozását a tanult módszerekkel! Figyeljen oda az adathalmaz teszt- és tanítóhalmazra bontására! Vizsgálja meg a tesztadathalmazra az osztályozások minőségét az eredeti

adathalmaz és a kiegyenlített adathalmaz esetére is!
- Számos adathalmazt találhatunk osztályozásra, például a `caret` csomagban, vagy a Kaggle és az UCI oldalakon!