ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №1
З ДИСЦИПЛІНИ «СТАТИСТИЧНІ АЛГОРИТМИ НАВЧАННЯ»
АЛГОРИТМИ КЛАСИФІКАЦІЇ

Виконав студент 2к. маг. Ломако Олександр

Варіант 9

Маємо датасет, в якому прописані різні хімічні характеристики скла, а також його тип. Метою даної лабораторної роботи є побудова класифікатора, який би за хімічним вмістом різних елементів у склі, а також його коефіцієнтом відбиття передбачав би його клас.

Тому, в якості відгука розглядатимемо тип скла (їх в датасеті 7, або 6):

1 - building_windows_float_processed
2 - building_windows_non_float_processed
3 - vehicle_windows_float_processed
4 - vehicle_windows_non_float_processed (none in this database) – його в датасеті немає
5 – containers
6 – tableware
7 – headlamps

В якості регресорів розглядатимуться хімічні елементи (вагові відсотки у відповідному оксиді) і коефіцієнт відбиття (таких регресорів в датасеті 9):

RI: refractive index (коефіцієнт відбиття)
Na: Sodium (натрій)
Mg: Magnesium (магній)
Al: Aluminum (алюміній)
Si: Silicon (силіцій)
K: Potassium (калій)
Ca: Calcium (кальцій)
Ba: Barium (барій)
Fe: Iron (ферум, залізо)

Одразу зауважу, що змінною *ID* ми нехтуємо, оскільки вона просто, формально, показуватиме номер рядка.

1. Спершу опрацюємо наші дані. Виконувати роботу будемо в середовищі R. Зчитаємо дані, і перевіримо на предмет пропущених даних.

```
> # підключаємо бібліотеки
>
> library(e1071)
> library(fastAdaboost)
> library(class)
> library(randomForest)
> library(C50)
>
> # зчитуємо дані
>
> data <- read.table('C:\\Users\\Razor\\Desktop\\дистанційне навчання\\статис
тичні алгоритми навчання\\lab1\\glass_data.DATA',
+                    header = F, sep = ',')
> data <- data[,-1] # видаляємо стовпчик з ID
```

```
> colnames(data) <- c('RI', 'Na', 'Mg', "Al", "Si", "K", "Ca", "Ba", "Fe", "T
ype")
> summary(data)
       RI              Na              Mg              Al              Si              K               Ca
 Min.   :1.511   Min.   :10.73   Min.   :0.000   Min.   :0.290   Min.   :69.81   Min.   :0.0000   Min.   : 5.430
 1st Qu.:1.517   1st Qu.:12.91   1st Qu.:2.115   1st Qu.:1.190   1st Qu.:72.28   1st Qu.:0.1225   1st Qu.: 8.240
 Median :1.518   Median :13.30   Median :3.480   Median :1.360   Median :72.79   Median :0.5550   Median : 8.600
 Mean   :1.518   Mean   :13.41   Mean   :2.685   Mean   :1.445   Mean   :72.65   Mean   :0.4971   Mean   : 8.957
 3rd Qu.:1.519   3rd Qu.:13.82   3rd Qu.:3.600   3rd Qu.:1.630   3rd Qu.:73.09   3rd Qu.:0.6100   3rd Qu.: 9.172
 Max.   :1.534   Max.   :17.38   Max.   :4.490   Max.   :3.500   Max.   :75.41   Max.   :6.2100   Max.   :16.190
       Ba              Fe              Type
 Min.   :0.000   Min.   :0.00000   Min.   :1.00
 1st Qu.:0.000   1st Qu.:0.00000   1st Qu.:1.00
 Median :0.000   Median :0.00000   Median :2.00
 Mean   :0.175   Mean   :0.05701   Mean   :2.78
 3rd Qu.:0.000   3rd Qu.:0.10000   3rd Qu.:3.00
 Max.   :3.150   Max.   :0.51000   Max.   :7.00
```

Бачимо, що всі дані є числовими, а значить з пропусками, очевидно, ми справи не маємо.

Для подальшої роботи розіб’ємо вибірку на тренувальну, і тестову (у відношенні 80% до 20%).

```
> s = round(length(data[,1])*0.8)
> # розіб'ємо вибірку на тренувальну і тестову
> Y <- data[,10] # відгук
> X <- data[,1:9]
>
> S <- sample(1:length(data[,1]), s)
> # тренувальна вибірка
> Ytrain <- Y[S]
> Xtrain <- X[S,]
> # тестова вибірка
> Ytest <- Y[-S]
> Xtest <- X[-S,]
```

2. Спершу для класифікації спробуємо скористатись методом $k-nn$ (*k найближчих сусідів*).

```
> #
> # метод k-nn
> #
>
> result_knn <- knn(Xtrain, Xtest, cl = Ytrain)
>
> table(result_knn, Ytest)
          Ytest
result_knn  1  2  3  5  6  7
         1  6  3  3  0  0  0
         2  0 12  0  0  1  0
         3  2  0  2  0  0  0
         5  0  0  0  3  0  1
         6  0  0  0  0  2  0
         7  0  0  0  1  0  7
> sum(diag(table(result_knn, Ytest)))/length(Ytest)
[1] 0.744186
```

Бачимо, що точність роботи даного класифікатора становить приблизно 74.4%.

Спробуємо поекспериментувати з різними k (за замовчуванням у функції *knn* $k=1$). Оскільки маємо роботу з даними не дуже масивної за обсягом природи, просто пробіжимося по різним k і порівняємо точності.

```
> for(i in 2:20){
+   r <- knn(Xtrain, Xtest, cl = Ytrain, k = i)
+   print(paste('k=',i,',точність: ', sum(diag(table(r, Ytest)))/length(Ytest
)))
+ }
[1] "k= 2 ,точність:  0.604651162790698"
```

```
[1] "k= 3 ,точність:  0.581395348837209"
[1] "k= 4 ,точність:  0.627906976744186"
[1] "k= 5 ,точність:  0.604651162790698"
[1] "k= 6 ,точність:  0.651162790697674"
[1] "k= 7 ,точність:  0.651162790697674"
[1] "k= 8 ,точність:  0.651162790697674"
[1] "k= 9 ,точність:  0.534883720930233"
[1] "k= 10 ,точність:  0.558139534883721"
[1] "k= 11 ,точність:  0.604651162790698"
[1] "k= 12 ,точність:  0.604651162790698"
[1] "k= 13 ,точність:  0.627906976744186"
[1] "k= 14 ,точність:  0.674418604651163"
[1] "k= 15 ,точність:  0.627906976744186"
[1] "k= 16 ,точність:  0.651162790697674"
[1] "k= 17 ,точність:  0.651162790697674"
[1] "k= 18 ,точність:  0.674418604651163"
[1] "k= 19 ,точність:  0.674418604651163"
[1] "k= 20 ,точність:  0.651162790697674"
```

Бачимо, що перевершити точність, отриману з параметром $k = 1$, саме в даному випадку для даної вибірки, не вдалося.

3. Тепер проведемо класифікацію за допомогою методу SVM.

```
> #
> # метод svm
> #
>
> train_sample <- data.frame(as.factor(Ytrain), Xtrain) # тренувальна вибірка
> test_sample <- data.frame(as.factor(Ytest), Xtest) # тестова вибірка
> colnames(train_sample)[1] <- 'Y'
> colnames(test_sample)[1] <- 'Y'
>
> model_svm <- svm(Y ~., data = train_sample, cost = 1, type = 'C-classificat
ion', kernel = 'polynomial')
>
> result_svm <- predict(model_svm, test_sample)
>
> table(result_svm, Ytest)
          Ytest
result_svm  1  2  3  5  6  7
         1  8 10  5  1  1  1
         2  0  2  0  0  1  0
         3  0  0  0  0  0  0
         5  0  3  0  2  0  1
         6  0  0  0  0  1  0
         7  0  0  0  1  0  6
> sum(diag(table(result_svm, Ytest)))/length(Ytest)
[1] 0.4418605
```

Отримали якість роботи класифікатора на рівні 44.1%, причому, бачимо, що в основному класифікатор помилково відносив різні типи скла до першого типу.

Спробуємо віднайти оптимальні параметри для моделі. Для цього скористаємося функцією *tune*, і побудуємо прогноз на найкращій моделі.

```
> tuned_svm_model <- tune(svm, Y ~., data = train_sample,
+                         ranges = list(cost = c(0.01:50, 0.01)),
+                         kernel = c('linear', 'radial'))
There were 50 or more warnings (use warnings() to see the first 50)
>
> tuned_svm_model

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
```

```
 cost
 7.01

- best performance: 0.3218954

> result_tuned_svm <- predict(tuned_svm_model$best.model, test_sample)
>
> table(result_tuned_svm, Ytest)
                Ytest
result_tuned_svm 1 2 3 5 6 7
               1 5 7 4 0 0 0
               2 2 7 1 1 2 1
               3 1 1 0 0 0 0
               5 0 0 0 2 0 1
               6 0 0 0 0 1 0
               7 0 0 0 1 0 6
> sum(diag(table(result_tuned_svm, Ytest)))/length(Ytest)
[1] 0.4883721
```

Бачимо, що результат, звісно, вдалось поліпшити, але він все одно менше навіть 50%, що свідчить про неефективність такого класифікатора.

4. Тепер використаємо методи, засновані на деревах.

Спершу спробуємо використати *randomForest* (випадковий ліс).

```
> #
> # моделі, засновані на деревах
> #
>
> # randomForest
> model_rF <- randomForest(Y~. , data = train_sample)
> result_rF <- predict(model_rF, test_sample)
>
> table(result_rF, Ytest)
         Ytest
result_rF  1  2  3  5  6  7
        1  5  3  2  0  1  0
        2  0 12  2  0  1  1
        3  3  0  1  0  0  0
        5  0  0  0  3  0  0
        6  0  0  0  0  1  0
        7  0  0  0  1  0  7
> sum(diag(table(result_rF, Ytest)))/length(Ytest)
[1] 0.6744186
```

Отримали точність роботи класифікатора на тестовій вибірці у 67.4%, що, в наших умовах, не так і погано.

Далі спробуємо використати дерева прийняття рішень (*decision trees, C50).*

```
> # c5.0
> model_c50 <- C5.0(Y ~. , data = train_sample)
> result_c50 <- predict(model_c50, test_sample)
>
> table(result_c50, Ytest)
          Ytest
result_c50 1 2 3 5 6 7
         1 6 3 3 0 0 0
         2 0 9 0 0 0 0
         3 1 1 2 0 0 0
         5 0 0 0 3 0 1
         6 1 2 0 0 3 0
         7 0 0 0 1 0 7
> sum(diag(table(result_c50, Ytest)))/length(Ytest)
[1] 0.6976744
```

У порівнянні з минулим випадком, ми дещо підвищили точність до 69.7%, але, звісно, хотілось би більше.

5. Далі змінну *Fe* перетворимо у факторну із 2 значеннями: більше та менше середнього.

```
> #
> # введемо факторну змінну
> #
>
> m <- mean(data[,9])
> data_mod <- data
> data_mod[data_mod[,9] >= m, 9] <- 1
> data_mod[data_mod[,9] < m, 9] <- 0
>
> # розіб'ємо на тренувальну та тестові вибірки
>
> Y_mod <- data_mod[,10] # відгук
> X_mod <- data_mod[,1:9]
>
> # тренувальна вибірка
> Ytrain_mod <- Y_mod[S]
> Xtrain_mod <- X_mod[S,]
> # тестова вибірка
> Ytest_mod <- Y_mod[-S]
> Xtest_mod <- X_mod[-S,]
> train_sample_mod <- data.frame(as.factor(Ytrain_mod), Xtrain_mod) # тренува
льна вибірка
> test_sample_mod <- data.frame(as.factor(Ytest_mod), Xtest_mod) # тестова ви
бірка
> colnames(train_sample_mod)[1] <- 'Y'
> colnames(test_sample_mod)[1] <- 'Y'
```

Далі застосуємо всі алгоритми класифікації, згадані і використані до цього.

```
> #
> # метод k-nn
> #
>
> result_knn_mod <- knn(Xtrain_mod, Xtest_mod, cl = Ytrain_mod)
>
> table(result_knn_mod, Ytest_mod)
              Ytest_mod
result_knn_mod  1  2  3  5  6  7
             1  5  5  3  0  0  0
             2  1 10  0  0  1  0
             3  2  0  2  0  0  0
             5  0  0  0  3  0  1
             6  0  0  0  0  2  0
             7  0  0  0  1  0  7
> sum(diag(table(result_knn_mod, Ytest_mod)))/length(Ytest_mod)
[1] 0.6744186
>
> for(i in 2:20){
+   r <- knn(Xtrain_mod, Xtest_mod, cl = Ytrain_mod, k = i)
+   print(paste('k=',i,',точність: ', sum(diag(table(r, Ytest_mod)))/length(Y
test_mod)))
+ }
[1] "k= 2 ,точність:  0.534883720930233"
[1] "k= 3 ,точність:  0.581395348837209"
[1] "k= 4 ,точність:  0.651162790697674"
[1] "k= 5 ,точність:  0.720930232558139"
[1] "k= 6 ,точність:  0.604651162790698"
[1] "k= 7 ,точність:  0.558139534883721"
[1] "k= 8 ,точність:  0.581395348837209"
[1] "k= 9 ,точність:  0.558139534883721"
[1] "k= 10 ,точність:  0.581395348837209"
[1] "k= 11 ,точність:  0.581395348837209"
```

```
[1] "k= 12 ,точність:  0.604651162790698"
[1] "k= 13 ,точність:  0.581395348837209"
[1] "k= 14 ,точність:  0.604651162790698"
[1] "k= 15 ,точність:  0.604651162790698"
[1] "k= 16 ,точність:  0.604651162790698"
[1] "k= 17 ,точність:  0.604651162790698"
[1] "k= 18 ,точність:  0.604651162790698"
[1] "k= 19 ,точність:  0.604651162790698"
[1] "k= 20 ,точність:  0.604651162790698"
>
> #
> # метод svm
> #
>
> model_svm_mod <- svm(Y ~., data = train_sample_mod, cost = 1, type = 'C-cla
ssification', kernel = 'polynomial')
>
> result_svm_mod <- predict(model_svm_mod, test_sample_mod)
>
> table(result_svm_mod, Ytest_mod)
              Ytest_mod
result_svm_mod 1 2 3 5 6 7
             1 8 9 5 0 1 1
             2 0 3 0 1 1 0
             3 0 0 0 0 0 0
             5 0 3 0 2 0 1
             6 0 0 0 0 1 0
             7 0 0 0 1 0 6
> sum(diag(table(result_svm_mod, Ytest_mod)))/length(Ytest_mod)
[1] 0.4651163
>
> tuned_svm_model_mod <- tune(svm, Y ~., data = train_sample_mod,
+                           ranges = list(cost = c(0.01:50, 0.01)),
+                           kernel = c('linear', 'radial'))
There were 50 or more warnings (use warnings() to see the first 50)
>
> tuned_svm_model_mod

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
 2.01

- best performance: 0.3052288

> result_tuned_svm_mod <- predict(tuned_svm_model_mod$best.model, test_sample
_mod)
>
> table(result_tuned_svm_mod, Ytest_mod)
                   Ytest_mod
result_tuned_svm_mod 1 2 3 5 6 7
                   1 5 7 3 0 0 0
                   2 2 5 2 0 2 2
                   3 1 1 0 0 0 0
                   5 0 2 0 3 0 0
                   6 0 0 0 0 1 0
                   7 0 0 0 1 0 6
> sum(diag(table(result_tuned_svm_mod, Ytest_mod)))/length(Ytest_mod)
[1] 0.4651163
>
> #
> # моделі, засновані на деревах
> #
>
> # randomForest
> model_rF_mod <- randomForest(Y~. , data = train_sample_mod)
> result_rF_mod <- predict(model_rF_mod, test_sample_mod)
>
> table(result_rF_mod, Ytest_mod)
```

```
             Ytest_mod
result_rF_mod  1  2  3  5  6  7
            1  5  1  2  0  1  0
            2  0 13  2  0  1  1
            3  3  0  1  0  0  0
            5  0  0  0  3  0  0
            6  0  0  0  0  1  0
            7  0  1  0  1  0  7
> sum(diag(table(result_rF_mod, Ytest_mod)))/length(Ytest_mod)
[1] 0.6976744
>
> # c5.0
> model_c50_mod <- C5.0(Y ~. , data = train_sample_mod)
> result_c50_mod <- predict(model_c50_mod, test_sample_mod)
>
> table(result_c50_mod, Ytest_mod)
              Ytest_mod
result_c50_mod 1 2 3 5 6 7
             1 6 3 3 0 0 0
             2 0 8 0 1 0 0
             3 1 1 2 0 0 0
             5 0 1 0 2 0 1
             6 1 2 0 0 3 0
             7 0 0 0 1 0 7
> sum(diag(table(result_c50_mod, Ytest_mod)))/length(Ytest_mod)
[1] 0.6511628
```

6. Для більш кращого розуміння отриманих результатів, внесемо дані в табличку (вноситимемо найкращу точність роботи класифікатора):

Метод	Без факторної змінної *Fe*	Із факторною змінною *Fe*
k-nn	74.41%	72.09%
svm	48.83%	46.51%
randomForest	67.44%	69.76%
C5.0	69.76%	65.11%

Отже, можемо бачити, що в цілому, найвищу точність роботи класифікатора отримували при застосуванні метода *k-nn*, в той час як метод *svm* дав найгірший результат. Методи, засновані на деревах, в цілому, кардинально по точності не відрізнялись.

Введення факторної змінної не виявилося фатальним: якість роботи класифікатора не змінювалась більш, ніж на 4.5%. Хоча, в цілому, введення саме такої факторної змінної в цілому якість зменшувала. Можливо, це зумовлене тим, що ця змінна в принципі в наших даних набувала значень або 0, або дуже і дуже не значних (на кшалт 0.11, 0.24), а ми цим незначним значенням присвоювали цілу 1, в той час як 0 залишався 0.

В цілому, здається, якість роботи класифікатора мала би бути кращою. Я, на жаль, не можу стверджувати, чому *svm* настільки спрацював неефективно, можливо, тому що у нас в якості відгуку стояла саме факторна змінна…

Також варто зазначити, що у нас в цілому була достатньо обмежена кількість даних в датасеті (214), переважна більшість яких належала першому класу.