Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного автономного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ***ИУК «Информатика и управление»***

КАФЕДРА ***ИУК5 «Системы обработки информации»***

Лабораторная работа №1

Линейные классификаторы

ДИСЦИПЛИНА: «Методы машинного обучения»

Выполнил: студент гр. ИУК5-72Б _____ (Подпись) Ли Р. В. (Ф.И.О.)

Проверил: _____ (Подпись) _____ (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга 2025

Вариант 14

1. Создайте фрейм данных из $N = 27$ записей со следующими полями: *Nrow* – номер записи, *Name* – место нахождение цветного металла, *BirthYear* – год обнаружения, Count –прогнозируемое количество, *Salary* – стоимость добычи. Cost – стоимость перевозки, PowerIOn – значение энергии ионизации, Electro – относительная электроотрицательности, Radius – радиус атома. Заполните данный фрейм данными так, что *Nrow* изменяется от 1 до *N*,

Name задается произвольно, *BithYear* распределен равномерно (случайно) на отрезке [1970,1994], , Cost для ископаемых найденных до 1977 г.р. определяется по формуле Cost = (ln(2009 − Birth*Y ear*) +1) ∗ 59000, для остальных Cost = (log2(2009 − Birth*Year*) + 1) ∗ 69000. Стоимость добычи *Salary* = (log2(2007 + Birth*Year*)+1)*79000

Ранжируйте ископаемые по стоимости добычи, начиная с минимальной суммы. Добавьте в таблицу поле, соответствующее общему социальному вычету за перевозки грузов (ставка 10%), выплаченному, если стоимость перевозок за каждый год начислялась согласно формулам для Cost, где вместо 2009 следует последовательно подставить каждый год добычи полезных ископаемых.

2. Постройте линейный классификатор для классификации тяжелых и легких цветных металлов цинка, олова, бериллия, титана, лития, рубидия на основе трех признаков – Энергия ионизации атома, радиус атома, Относительная электроотрицательность. Использовать машину опорных векторов и алгоритм персептрона. Полученные результаты сравнить с использованием функций языка R - Predict и Table.

Для машины опорных векторов типа "C-classification" с сигмоидальным ядром, добейтесь нулевой ошибки сначала на обучающей выборке, а затем на тестовой, путем изменения параметра *C*. Изменяя значение параметра gamma, продемонстрируйте эффект переобучения, выполните при этом визуализацию разбиения пространства признаков на области

3. Допустим, что решающая функция линейного классификатора в упрощенном виде выглядит так:

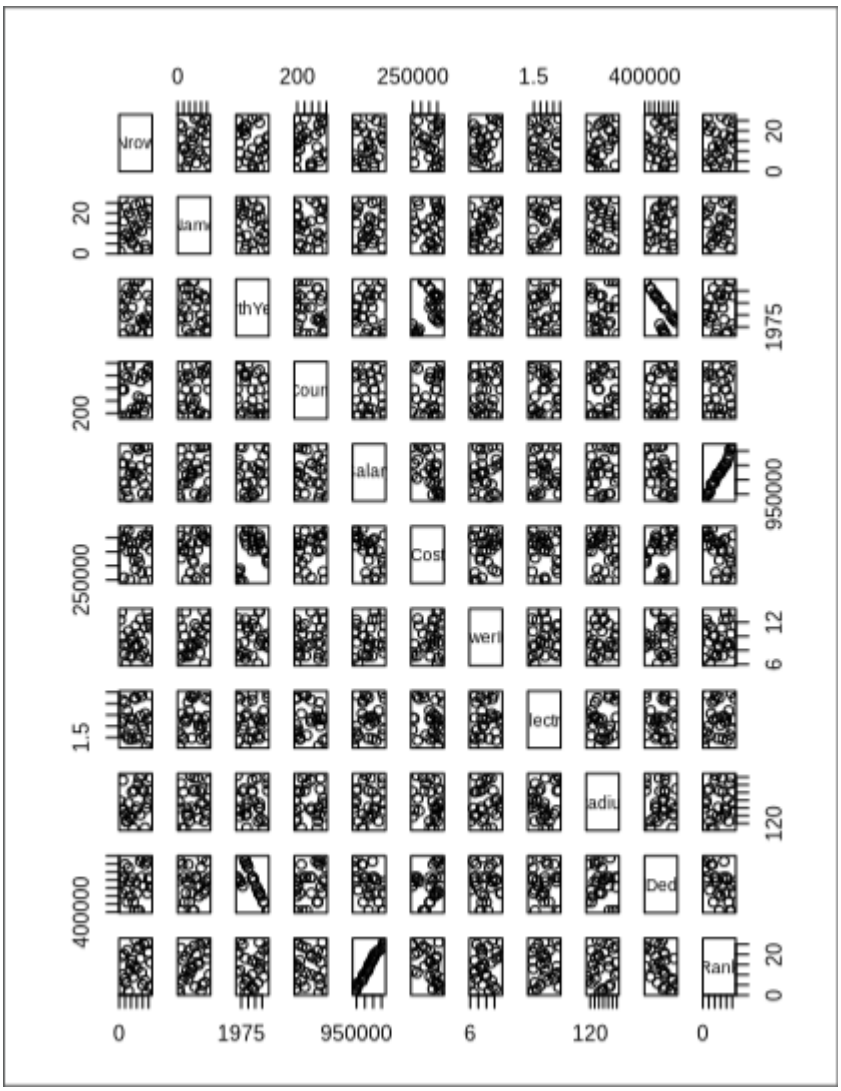
$$f(x_1, x_2) = 5 \cdot x_1^2 + 5 \cdot x_2^2 - 8 \cdot x_1 \cdot x_2$$

Найти координаты и значение функции в точке минимума методом градиентного спуска.

task1.R

```
Name <- c(
    "Aluminum",
    "Copper",
    "Zinc",
    "Nickel",
    "Iron",
    "Tin",
    "Lead",
    "Silver",
    "Gold",
    "Platinum",
    "Chromium",
    "Cobalt",
    "Manganese",
    "Titanium",
    "Magnesium",
    "Cadmium",
    "Mercury",
    "Tungsten",
    "Lithium",
    "Sodium",
    "Potassium",
    "Calcium",
    "Barium",
    "Strontium",
    "Vanadium",
    "Molybdenum",
    "Palladium"
)
set.seed(123) # reproducibility

N <- length(Name)

# Generate BirthYear first
BirthYear <- sample(1970:1994, N, replace = TRUE)

Cost <- ifelse(
    BirthYear < 1977,
    runif(
        N,
        min = 0.9 * (log(2009 - BirthYear) + 1) * 59000,
        max = 1.1 * (log(2009 - BirthYear) + 1) * 59000
    ),
    runif(
        N,
        min = 0.9 * (log2(2009 - BirthYear) + 1) * 69000,
        max = 1.1 * (log2(2009 - BirthYear) + 1) * 69000
    )
)
```

```
Salary <- runif(
    N,
    min = 0.9 * (log2(2007 + BirthYear) + 1) * 79000,
    max = 1.1 * (log2(2007 + BirthYear) + 1) * 79000
)

df <- data.frame(
    Nrow = 1:N,
    Name = Name,
    BirthYear = BirthYear,
    Count = sample(100:1000, N, replace = TRUE),
    Salary = round(Salary),
    Cost = round(Cost),
    PowerIOn = runif(N, 5, 15),
    Electro = runif(N, 0.7, 3.5),
    Radius = runif(N, 100, 250)
)

social_deduction <- numeric(N)

for (i in 1:N) {
    years <- df$BirthYear[i]:2009 + 1
    if (df$BirthYear[i] < 1977) {
        yearly_cost <- (log(years - df$BirthYear[i]) + 1) * 59000
    } else {
        yearly_cost <- (log2(years - df$BirthYear[i]) + 1) * 69000
    }
    social_deduction[i] <- sum(yearly_cost) * 0.1
}

df$SocialDeduction <- round(social_deduction)

df$Rank <- rank(df$Salary)

df_ranked <- df[order(df$Rank), ]

options(width = 200)
print(df_ranked)
# print(df_ranked[, setdiff(names(df), "Rank")])
plot(df)
```



task2.R

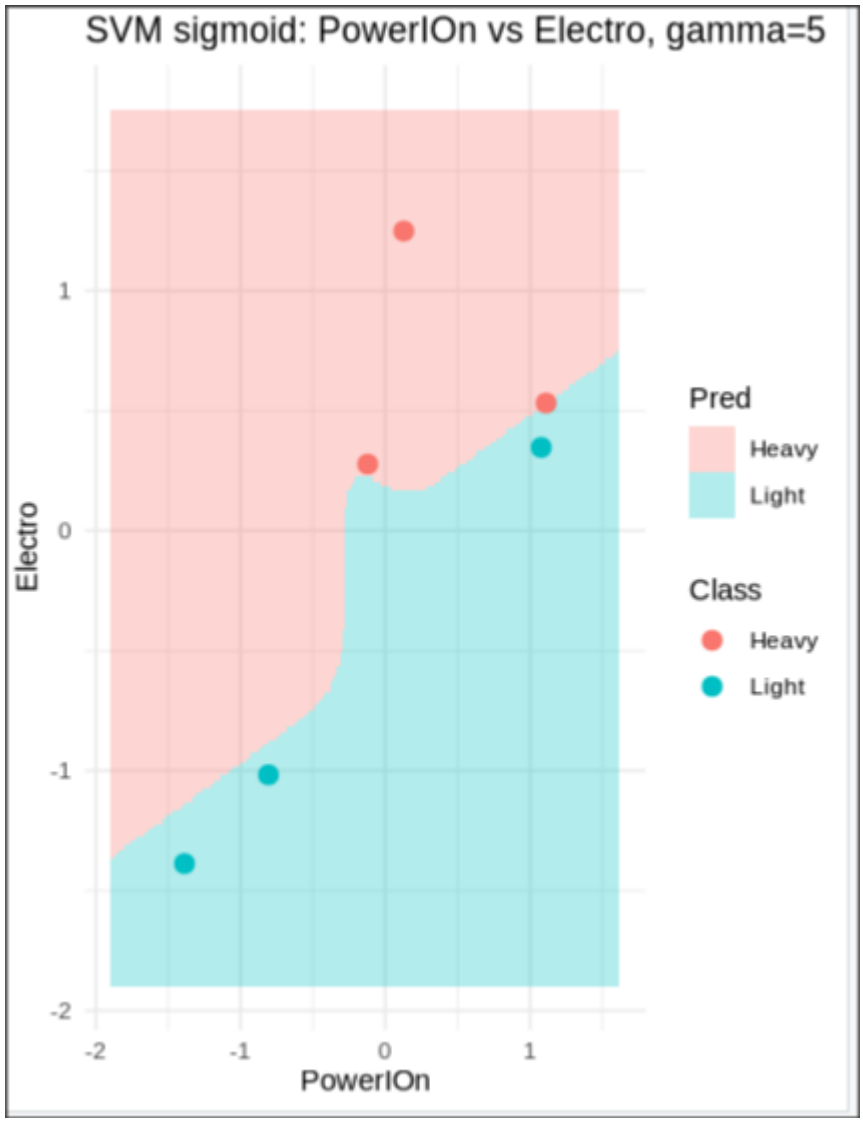
```
if(!require(e1071)) install.packages("e1071")
if(!require(ggplot2)) install.packages("ggplot2")
if(!require(caret)) install.packages("caret")
library(e1071)
library(ggplot2)
library(caret)

Name <- c("Zn", "Sn", "Be", "Ti", "Li", "Rb")
Class <- c("Heavy", "Heavy", "Light", "Heavy", "Light", "Light")
PowerIOn <- c(9.39, 7.34, 9.32, 6.82, 5.39, 4.18)
Radius <- c(135, 145, 112, 147, 152, 248)
Electro <- c(1.65, 1.96, 1.57, 1.54, 0.98, 0.82)

df <- data.frame(Name, PowerIOn, Radius, Electro, Class, stringsAsFactors =
TRUE)
df$ClassBinary <- ifelse(df$Class == "Heavy", 1, 0)

print(df)
```

```
features <- c("PowerIOn","Radius","Electro")
preProc <- preProcess(df[, features], method = c("center", "scale"))
Xstd <- predict(preProc, df[, features])
df_std <- cbind(df[, c("Name","Class","ClassBinary")], Xstd)

set.seed(42)
train_idx <- createDataPartition(df_std$Class, p = 0.67, list = FALSE)
train <- df_std[train_idx, ]
test  <- df_std[-train_idx, ]

cat("Train:\n"); print(train[, c("Name","Class")])
cat("Test:\n");  print(test[, c("Name","Class")])


perceptron_train <- function(X, y, lr = 0.1, epochs = 1000){
  n <- nrow(X); m <- ncol(X)
  w <- rep(0, m)
  b <- 0
  for (e in 1:epochs){
    errors <- 0
    for(i in 1:n){
      xi <- as.numeric(X[i,])
      yi <- y[i]
      activation <- sum(w * xi) + b
      pred <- ifelse(activation >= 0, 1, -1)
      if (yi != pred){
        w <- w + lr * yi * xi
        b <- b + lr * yi
        errors <- errors + 1
      }
    }
    if (errors == 0) break
  }
  list(w = w, b = b, epochs = e, errors = errors)
}

perceptron_predict <- function(model, X){
  scores <- as.matrix(X) %*% model$w + model$b
  ifelse(scores >= 0, "Heavy", "Light")
}

X_train <- as.matrix(train[, features])
y_train <- ifelse(train$Class == "Heavy", 1, -1)

per_model <- perceptron_train(X_train, y_train, lr = 0.1, epochs = 1000)
cat("Perceptron training finished. epochs used:", per_model$epochs, "final
errors in last epoch:", per_model$errors, "\n")

pred_per_train <- perceptron_predict(per_model, train[, features])
pred_per_test  <- perceptron_predict(per_model, test[, features])

cat("Perceptron - Train confusion:\n"); print(table(Predicted =
pred_per_train, Actual = train$Class))
cat("Perceptron - Test confusion:\n");  print(table(Predicted =
pred_per_test,  Actual = test$Class))

find_C_zero_train <- function(train_df, features, Cs = 10^seq(-2, 6, by = 1),
gamma = 1, coef0 = 0){
```

```
  res <- data.frame(C = numeric(), train_err = numeric(), test_err =
numeric(), stringsAsFactors = FALSE)
  for(Cval in Cs){
    model <- svm(as.formula(paste("Class ~", paste(features, collapse =
"+"))),
                 data = train_df, type = "C-classification",
                 kernel = "sigmoid", cost = Cval, gamma = gamma, coef0 =
coef0, scale = FALSE)
    pred_train <- predict(model, train_df)
    train_err <- sum(pred_train != train_df$Class)
    res <- rbind(res, data.frame(C = Cval, train_err = train_err, test_err =
NA))
    if(train_err == 0){
      return(list(C = Cval, model = model, summary = res))
    }
  }
  return(list(C = NA, model = NULL, summary = res))
}

Cs_try <- c(0.01, 0.1, 1, 10, 100, 1000, 1e4)
svm_search <- find_C_zero_train(train, features, Cs = Cs_try, gamma = 0.5,
coef0 = 0)
svm_search$summary
if(!is.na(svm_search$C)){
  cat("Найдено C с нулевой ошибкой на train:", svm_search$C, "\n")
  svm_model <- svm_search$model
  pred_train_svm <- predict(svm_model, train)
  pred_test_svm  <- predict(svm_model, test)
  cat("SVM - Train confusion:\n"); print(table(Predicted = pred_train_svm,
Actual = train$Class))
  cat("SVM - Test confusion:\n");  print(table(Predicted = pred_test_svm,
Actual = test$Class))
} else {
  cat("Не удалось достичь нулевой ошибки на train в диапазоне C_try.
Увеличьте диапазон C.\n")
}

plot_decision_region <- function(df_full, features_pair, fixed_feature,
fixed_value,
                                 kernel = "sigmoid", cost = 1, gamma = 0.5,
coef0 = 0,
                                 title = ""){
  x1 <- seq(min(df_full[[features_pair[1]]]) - 0.5,
            max(df_full[[features_pair[1]]]) + 0.5, length.out = 200)
  x2 <- seq(min(df_full[[features_pair[2]]]) - 0.5,
            max(df_full[[features_pair[2]]]) + 0.5, length.out = 200)
  grid <- expand.grid(x1, x2)
  names(grid) <- features_pair
  grid[[fixed_feature]] <- fixed_value

  model_all <- svm(Class ~ PowerIOn + Radius + Electro,
                   data = df_full,
                   type = "C-classification",
                   kernel = kernel, cost = cost, gamma = gamma, coef0 =
coef0, scale = FALSE)

  grid_pred <- predict(model_all, newdata = grid)
  plot_df <- cbind(grid, Pred = grid_pred)
```

```
  p <- ggplot() +
    geom_tile(data = plot_df, aes(x = !!sym(features_pair[1]),
                                  y = !!sym(features_pair[2]),
                                  fill = Pred), alpha = 0.3) +
    geom_point(data = df_full, aes(x = !!sym(features_pair[1]),
                                   y = !!sym(features_pair[2]),
                                   color = Class), size = 3) +
    labs(title = title,
         x = features_pair[1],
         y = features_pair[2]) +
    theme_minimal()

  print(p)
}


medianElectro <- median(df_std$Electro)
medianPowerIOn <- median(df_std$PowerIOn)
medianRadius <- median(df_std$Radius)

par(mfrow = c(1,1))
plot_decision_region(df_std, c("PowerIOn","Radius"), "Electro",
medianElectro,
                     cost = 10, gamma = 0.1, coef0 = 0, title = "SVM sigmoid:
gamma=0.1 (меньше переобуч.)")
plot_decision_region(df_std, c("PowerIOn","Radius"), "Electro",
medianElectro,
                     cost = 10, gamma = 5, coef0 = 0, title = "SVM sigmoid:
gamma=5 (возможное переобуч.)")

plot_decision_region(df_std, c("PowerIOn","Electro"), "Radius", medianRadius,
                     cost = 10, gamma = 0.1, coef0 = 0, title = "SVM sigmoid:
PowerIOn vs Electro, gamma=0.1")
plot_decision_region(df_std, c("PowerIOn","Electro"), "Radius", medianRadius,
                     cost = 10, gamma = 5, coef0 = 0, title = "SVM sigmoid:
PowerIOn vs Electro, gamma=5")

Cs_grid <- c(0.1, 1, 10, 100, 1000)
gammas  <- c(0.01, 0.1, 0.5, 1, 2, 5)
res_grid <- data.frame(C = numeric(), gamma = numeric(), train_err =
numeric(), test_err = numeric(), stringsAsFactors = FALSE)
for(Cv in Cs_grid){
  for(gv in gammas){
    m <- svm(Class ~ PowerIOn + Radius + Electro, data = train, type = "C-
classification",
             kernel = "sigmoid", cost = Cv, gamma = gv, coef0 = 0, scale =
FALSE)
    ptrain <- predict(m, train); ptest <- predict(m, test)
    res_grid <- rbind(res_grid, data.frame(C = Cv, gamma = gv,
                                           train_err = sum(ptrain !=
train$Class),
                                           test_err  = sum(ptest  !=
test$Class)))
  }
}
print(res_grid[order(res_grid$train_err, res_grid$test_err), ])

cat("\n=== Персептрон: итоговые таблицы ===\n")
```

```
cat("Train:\n"); print(table(Predicted = pred_per_train, Actual =
train$Class))
cat("Test:\n");  print(table(Predicted = pred_per_test,  Actual =
test$Class))

if(exists("svm_model")){
  cat("\n=== SVM (sigmoid) при C=", svm_search$C, " gamma=0.5: таблицы
===\n")
  cat("Train:\n"); print(table(Predicted = pred_train_svm, Actual =
train$Class))
  cat("Test:\n");  print(table(Predicted = pred_test_svm,  Actual =
test$Class))
} else {
  cat("\nSVM с нулевой ошибкой на train в выбранном диапазоне C не найден.
\n")
}
```



task3.R

```
f <-
function(x1,
x2) {
  5*x1^2 +
5*x2^2 -
8*x1*x2
}

grad <- function(x) {
  x1 <- x[1]; x2 <- x[2]
  c(10*x1 - 8*x2,
```

```
    10*x2 - 8*x1)
}

gradient_descent <- function(start, lr = 0.1, epochs = 1000, tol = 1e-6) {
  x <- start
  for (i in 1:epochs) {
    g <- grad(x)
    new_x <- x - lr * g
    if (sqrt(sum((new_x - x)^2)) < tol) { # проверка сходимости
      cat("Сошлось за", i, "итераций\n")
      return(list(min_point = new_x, min_value = f(new_x[1], new_x[2])))
    }
    x <- new_x
  }
  cat("Достигнут лимит итераций\n")
  list(min_point = x, min_value = f(x[1], x[2]))
}

set.seed(42)
res <- gradient_descent(start = c(2, 2), lr = 0.05)
print(res)
```