НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Дисциплина: «Анализ данных»

Домашнее задание на тему:
«Лабораторная работа №9»

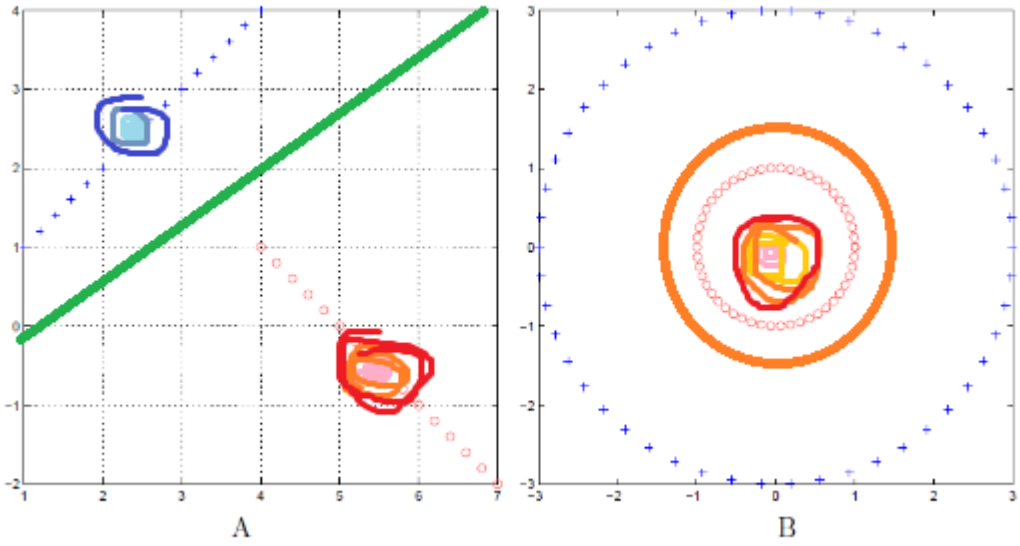
Выполнил: Осипов Лев,
студент группы 301ПИ (1).

Москва, 2015 г.

СОДЕРЖАНИЕ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

ЗАДАНИЕ 3



ПРАКТИЧЕСКАЯ ЧАСТЬ

Для решения задания была написана программа, обучающая гауссовский классификатор и линейный дискриминант Фишера с двумя видами матриц ковариаций: полной и диагональной. Классификаторы были протестированы на выборках разных размерностей: 2 и 200. Для размерности 2 были визуализированы решающие правила.

Результаты работы программы:

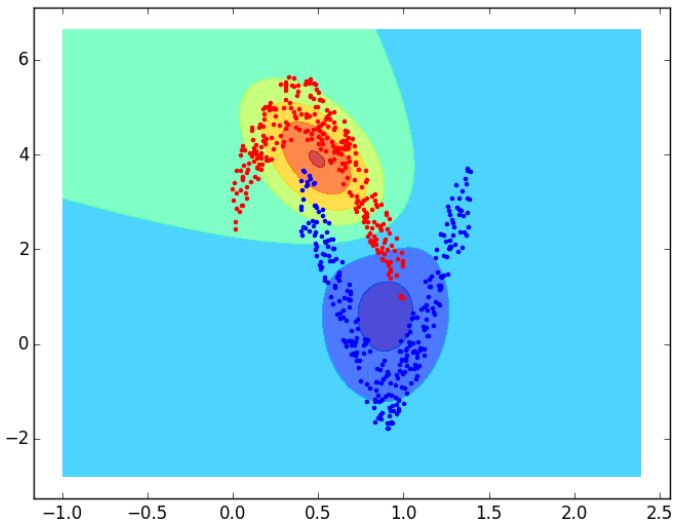


Рис. 1. Гауссовский классификатор, полная матрица

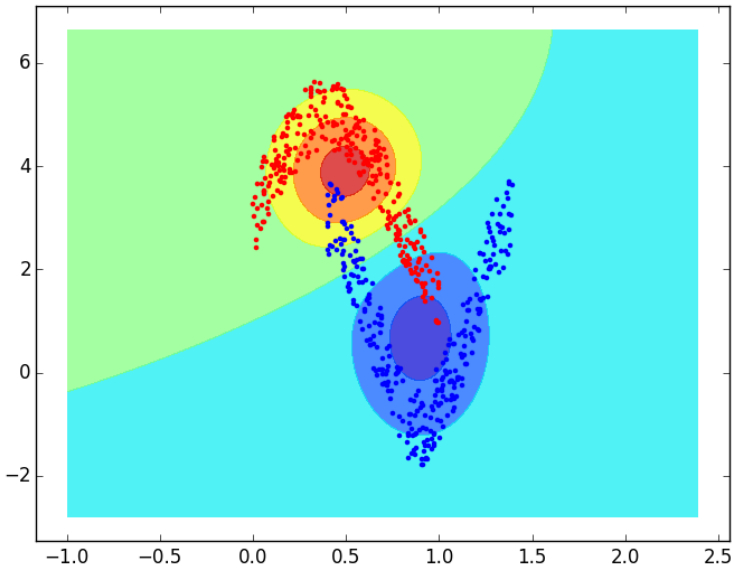


Рис. 2. Гауссовский классификатор, диагональная матрица

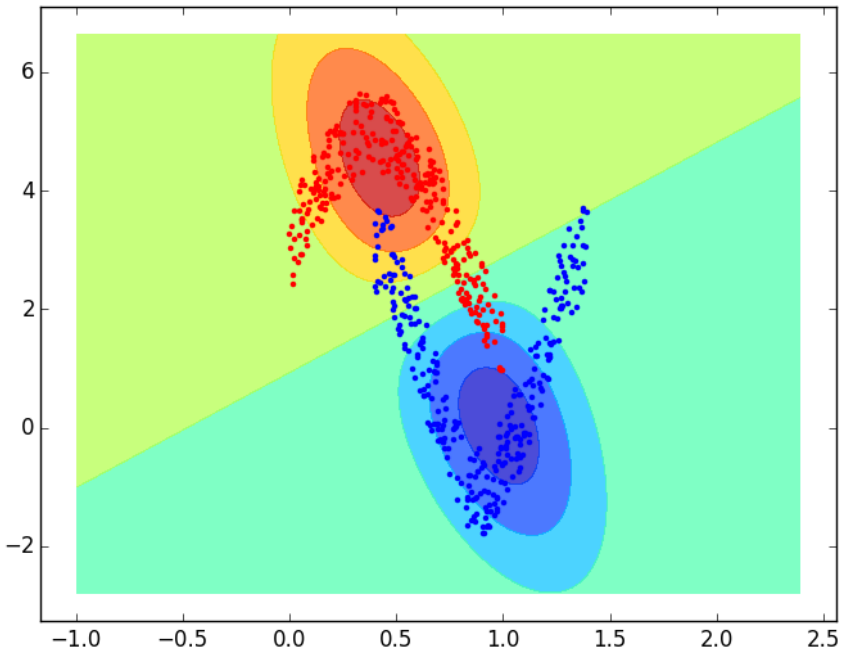


Рис. 3. Линейный дискриминант Фишера, полная матрица

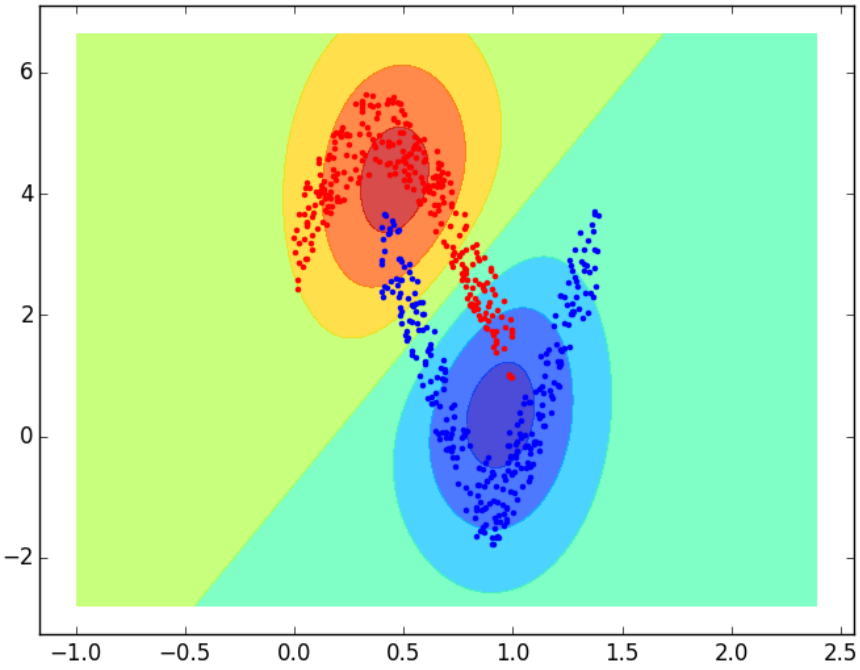


Рис. 4. Линейный дискриминант Фишера, диагональная матрица

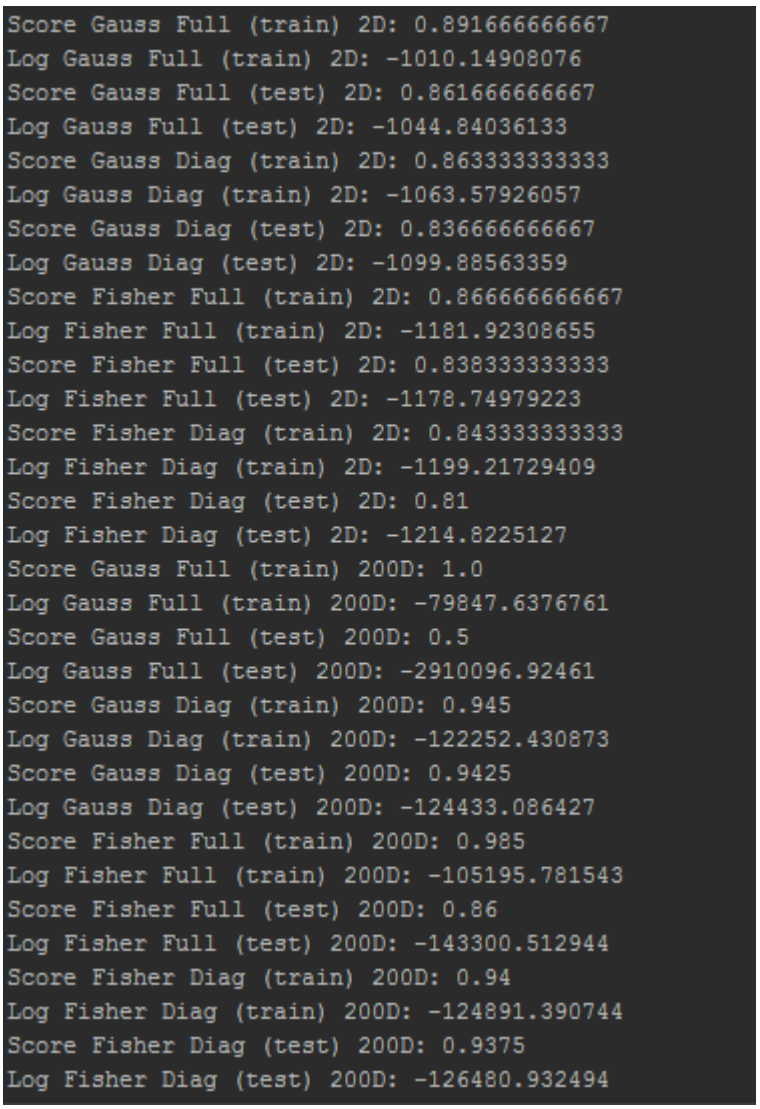


Рис. 4. Результаты точности классификаторов и логарифма правдоподобия на обоих классификаторах с разными видами матриц и размерностями данных на обучающей и тестовой выборках

В целом заметно, что при использовании диагональных матриц решающее правило разделяет пространство таким образом, что в среднем расстояние от точек разных классов до раздела примерно уравнивается. При использовании же полных матриц видно, особенно на рис. 1, что кривая может просто «огибать» один из классов.

Что касается вида решающего правила, в линейном дискриминанте Фишера оно является прямой, тогда как в гауссовском классификаторе оно принимает вид кривой. Это можно объяснить тем, что в линейном дискриминанте у нас равны ковариационные матрицы.

Очевидно, что значения величин правдоподобия и точности зависимы (тем больше точность, тем меньше по модулю значение логарифма). Это наблюдается при разных размерностях.

При больших размерностях видно, что полная матрица может давать существенно низкий результат. Хотя при размерности 2 ее результаты были выше, чем у диагональной. Поэтому нельзя сказать, что какой-то вид матрицы лучше, следует принимать во внимание остальные обстоятельства.

СПИСОК ЛИТЕРАТУРЫ

1) **Анализ данных (Программная инженерия)** – http://wiki.cs.hse.ru/Анализ_данных_(Программная_инженерия)

ТЕКСТ ПРОГРАММЫ

```
__author__ = 'Lev Osipov'

import numpy as np
from scipy.stats import multivariate_normal as mn
import matplotlib.pyplot as plt
import pandas as pd


def objects_divide(x, y):
    objects1 = []
    objects2 = []
    for i in xrange(len(y)):
        if y[i] == 0:
            objects1.append(x[i])
        else:
            objects2.append(x[i])
    return np.array(objects1), np.array(objects2)

def fit_parameters(x, y, kind):
    objects1, objects2 = objects_divide(x, y)

    mean1 = np.mean(objects1, axis=0)
    cov_matrix1 = np.cov(objects1, rowvar=0)

    mean2 = np.mean(objects2, axis=0)
    cov_matrix2 = np.cov(objects2, rowvar=0)

    # If we need diagonal matrices
    if kind == "diag":
        cov_matrix1 = np.diag(np.diag(cov_matrix1))
        cov_matrix2 = np.diag(np.diag(cov_matrix2))

    return [(mean1, cov_matrix1), (mean2, cov_matrix2)]


def fit_lda(x, y, kind):
    objects1, objects2 = objects_divide(x, y)

    mean1 = np.mean(objects1, axis=0)

    mean2 = np.mean(objects2, axis=0)

    # General cov matrix
    cov_matrix = np.cov(x, rowvar=0)

    # If we need diagonal matrices
    if kind == "diag":
        cov_matrix = np.diag(np.diag(cov_matrix))

    return [(mean1, cov_matrix), (mean2, cov_matrix)]
```

```
def class_posterior(x, class_params):

    # Probability of first class
    prob1 = mn.pdf(x, class_params[0][0], class_params[0][1])

    # Probability of second class
    prob2 = mn.pdf(x, class_params[1][0], class_params[1][1])

    prob = np.zeros((len(x), 2))
    prob[:, 0] = prob1
    prob[:, 1] = prob2
    return np.array(prob)


def score(x, y, class_params):

    prob = class_posterior(x, class_params)
    prob[:, 0] *= (float(len(y[y == 0])) / len(y))
    prob[:, 1] *= (float(len(y[y == 1])) / len(y))
    correct = 0
    for i in xrange(len(y)):
        if prob[i][0] > prob[i][1]:
            prediction = 0
        else:
            prediction = 1
        if prediction == y[i]:
            correct += 1
    return float(correct) / len(y)


def loglikelihood(x, y, class_params):
    ll = 0
    for i in xrange(len(x)):
        ll += mn.logpdf(x[i], class_params[int(y[i])][0],
class_params[int(y[i])][1])
    return ll


def plot_decision_rule(x, y, class_params):

    x_min = x[:, 0].min() - 1
    x_max = x[:, 0].max() + 1
    y_min = x[:, 1].min() - 1
    y_max = x[:, 1].max() + 1
    delta = 0.01
    xs, ys = np.meshgrid(np.arange(x_min, x_max, delta),
np.arange(y_min, y_max, delta))

    grid = np.c_[xs.ravel(), ys.ravel()]
    prob = class_posterior(grid, class_params)
    z = prob[:, 0] * (float(len(y[y == 0])) / len(y)) - prob[:,
1] * (float(len(y[y == 1])) / len(y))
```

```
    z = z.reshape(xs.shape)
    plt.contourf(xs, ys, z, alpha=0.7)

    objects1, objects2 = objects_divide(x, y)
    plt.scatter(objects1[:, 0], objects1[:, 1], color='r',
marker='.')
    plt.scatter(objects2[:, 0], objects2[:, 1], color='b',
marker='.')

    plt.show()


# 2d

data = pd.read_csv('2d_train.csv', header=None).as_matrix()
train_features = data[:, 1:]
train_classes = data[:, 0]

data = pd.read_csv('2d_test.csv', header=None).as_matrix()
test_features = data[:, 1:]
test_classes = data[:, 0]

params = fit_parameters(train_features, train_classes, "full")
print "Score Gauss Full (train) 2D: " +
str(score(train_features, train_classes, params))
print "Log Gauss Full (train) 2D: " +
str(loglikelihood(train_features, train_classes, params))
print "Score Gauss Full (test) 2D: " + str(score(test_features,
test_classes, params))
print "Log Gauss Full (test) 2D: " +
str(loglikelihood(test_features, test_classes, params))
plot_decision_rule(test_features, test_classes, params)

params = fit_parameters(train_features, train_classes, "diag")
print "Score Gauss Diag (train) 2D: " +
str(score(train_features, train_classes, params))
print "Log Gauss Diag (train) 2D: " +
str(loglikelihood(train_features, train_classes, params))
print "Score Gauss Diag (test) 2D: " + str(score(test_features,
test_classes, params))
print "Log Gauss Diag (test) 2D: " +
str(loglikelihood(test_features, test_classes, params))
plot_decision_rule(test_features, test_classes, params)

params = fit_lda(train_features, train_classes, "full")
print "Score Fisher Full (train) 2D: " +
str(score(train_features, train_classes, params))
print "Log Fisher Full (train) 2D: " +
str(loglikelihood(train_features, train_classes, params))
print "Score Fisher Full (test) 2D: " + str(score(test_features,
test_classes, params))
print "Log Fisher Full (test) 2D: " +
str(loglikelihood(test_features, test_classes, params))
```

```
plot_decision_rule(test_features, test_classes, params)

params = fit_lda(train_features, train_classes, "diag")
print "Score Fisher Diag (train) 2D: " +
str(score(train_features, train_classes, params))
print "Log Fisher Diag (train) 2D: " +
str(loglikelihood(train_features, train_classes, params))
print "Score Fisher Diag (test) 2D: " + str(score(test_features,
test_classes, params))
print "Log Fisher Diag (test) 2D: " +
str(loglikelihood(test_features, test_classes, params))
plot_decision_rule(test_features, test_classes, params)

# 200d

data = pd.read_csv('200d_train.csv', header=None).as_matrix()
train_features = data[:, 1:]
train_classes = data[:, 0]

data = pd.read_csv('200d_test.csv', header=None).as_matrix()
test_features = data[:, 1:]
test_classes = data[:, 0]

params = fit_parameters(train_features, train_classes, "full")
print "Score Gauss Full (train) 200D: " +
str(score(train_features, train_classes, params))
print "Log Gauss Full (train) 200D: " +
str(loglikelihood(train_features, train_classes, params))
print "Score Gauss Full (test) 200D: " +
str(score(test_features, test_classes, params))
print "Log Gauss Full (test) 200D: " +
str(loglikelihood(test_features, test_classes, params))

params = fit_parameters(train_features, train_classes, "diag")
print "Score Gauss Diag (train) 200D: " +
str(score(train_features, train_classes, params))
print "Log Gauss Diag (train) 200D: " +
str(loglikelihood(train_features, train_classes, params))
print "Score Gauss Diag (test) 200D: " +
str(score(test_features, test_classes, params))
print "Log Gauss Diag (test) 200D: " +
str(loglikelihood(test_features, test_classes, params))

params = fit_lda(train_features, train_classes, "full")
print "Score Fisher Full (train) 200D: " +
str(score(train_features, train_classes, params))
print "Log Fisher Full (train) 200D: " +
str(loglikelihood(train_features, train_classes, params))
print "Score Fisher Full (test) 200D: " +
str(score(test_features, test_classes, params))
print "Log Fisher Full (test) 200D: " +
str(loglikelihood(test_features, test_classes, params))
```

```
params = fit_lda(train_features, train_classes, "diag")
print "Score Fisher Diag (train) 200D: " +
str(score(train_features, train_classes, params))
print "Log Fisher Diag (train) 200D: " +
str(loglikelihood(train_features, train_classes, params))
print "Score Fisher Diag (test) 200D: " +
str(score(test_features, test_classes, params))
print "Log Fisher Diag (test) 200D: " +
str(loglikelihood(test_features, test_classes, params))
```