ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

**Московский институт электроники и математики им. А.Н. Тихонова**

Годяев Дмитрий Владиславович , группа БИВ181

**ОТЧЕТ**

# ПО ЛАБОРАТОРНОЙ РАБОТЕ

# «Машинное обучение»

по дисциплине

«Математический компьютерный практикум»

Вариант 4

Дата сдачи отчета 27.05.2020

Москва 2020 г.

**Исходные данные варианта**

Файл svmdata4.txt для обучения

Файл svmdata4test.txt для тестирования

**Задача**

Классифицировать данные не менее чем 3-мя разными методами.

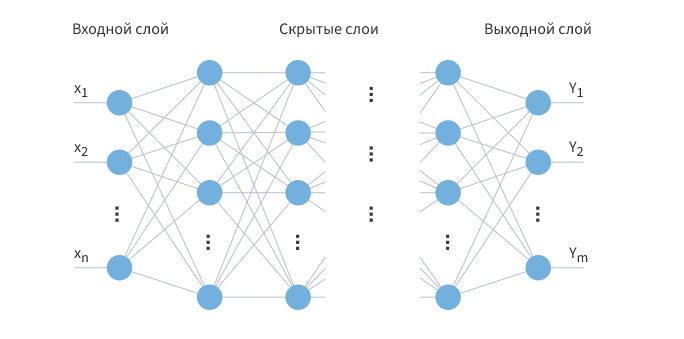
**Теория**

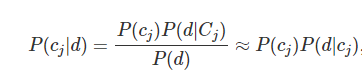
Т.к. в наших выборках существует 3 параметра, а именно X1, X2 и цвет, то будем использовать данные X1 и X2, как параметры, а цвет, как результат. Значит, нам предстоит решить задачу бинарной классификации, т.к. цветов – всего 2 ( красный и зелёный)

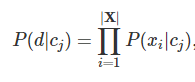
Для решения данной задачи воспользуемся следующими методами:

1. Нейронная сеть, типа многослойный перцептрон
2. Байесовский наивный классификатор
3. Линейный дискриминантный алгоритм
4. Квадратичный дискриминантный алгоритм

**Теоретическое описание использованных методов:**

1. Многослойный перцептрон – тип нейросети, в котором слои нейронов связаны последовательно (i-ый слой с i+1-вым слоем). Где каждый нейрон имеет собственную функцию активации и имеются скрытие слои.
2. Байесовский наивный классификатор – основан на гипотезе о максимальной вероятности, т.е. объект d принадлежит классу объектов если достигается наибольшая апостериорная вероятность.

По формуле Байеса

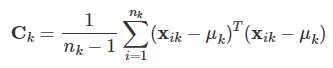
Если сделать “наивное” предположение, что все признаки, описывающие классифицируемые объекты, совершенно равноправны между собой и не связаны друг c другом, то P(d|cj) можно вычислить как произведение вероятностей встретить признак xi ( xi∈X ) среди объектов класса cj:

При умножении очень малых условных вероятностей может наблюдаться потеря значащих разрядов, в связи с чем вместо самих оценок вероятностей P(xi|cj) применяют логарифмы этих вероятностей. Поскольку логарифм - монотонно возрастающая функция, то класс cj с наибольшим значением логарифма вероятности останется наиболее вероятным.

1. Линейный дискриминантный анализ – реализован на двух статических процедурах:

Интерпретация межгрупповых различий - умение сформировать разделяющую поверхность для объектов обучающей выборки и выбор наиболее информативных переменных.

Классификация – предсказание значения группировочного фактора экзаменуемой группы наблюдений.

В основе дискриминантного анализа лежит предположение о том, что описания объектов каждого k -го класса представляют собой реализации многомерной случайной величины, распределенной по нормальному закону Nm(μk;Σk) со средними μk и ковариационной матрицей   
Для случая двух классов. Пусть дискриминантные переменные x - оси m-мерного евклидова пространства. Каждый объект (наблюдение) является точкой этого пространства с координатами, представляющими собой фиксируемые значения каждой переменной. Если оба класса отличаются друг от друга по наблюдаемым переменным, их можно представить как скопления точек в разных областях пространства, которые могут частично перекрываться.

Задача дискриминантного анализа заключается в проведении дополнительной оси, которая проходит через облако точек таким образом, что проекции на нее обеспечивают наилучшую разделимость на два класса. Ее положение задается линейной дискриминантной функцией (linear discriminant, LD)

1. Квадратичный дискриминантный алгоритм – нелинейное обобщение линейного дискриминантного метода, где вместо линейной дискриминантной функции применяется квадратичная функция.

**Алгоритм работы программы**

1. На вход передаем заданные по условию файлы
2. Читаем файлы, разбиваем данные построчно и разделяем каждую строку на элементы массива
3. Нормализуем выборки, используя правило z-масшабирования (разделим разницу между значением и дисперсией на среднее)
4. Создадим нейронную сеть.
5. Воспользуемся всеми описанными алгоритмами и нейронной сетью
6. Сравним полученные результаты

**Листинг программы**

import pandas as pd

import numpy as np

import re

from keras.layers import Dense,GaussianNoise,LeakyReLU,Activation

from keras.models import Sequential

import matplotlib.pyplot as plt

from sklearn.naive\_bayes import GaussianNB,BernoulliNB,MultinomialNB,CategoricalNB

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.discriminant\_analysis import QuadraticDiscriminantAnalysis

train=open('svmdata4.txt','r') # читаем файл с тренировочной выборкой

train = train.read().splitlines() # разделяем текст на строки

test=open('svmdata4test.txt','r') # читаем файл с тестовой выборкой

test=test.read().splitlines()

# Удаляем первую строку, содрежащую индексы

train.pop(0)

test.pop(0)

train\_data=np.array(train)

test\_data=np.array(test)

width=2 # ширина двумерного массива, т.к. мы имеем два числа, определющих выборку

train\_length=train\_data.shape[0] # длина массива тренировочных данных

test\_length=test\_data.shape[0] # длнна массива тестовых данных

# подготавливаем массивы необходимых размеров

x\_train=np.ones((train\_length,width))

y\_train=np.ones((train\_length,1))

x\_test=np.ones((test\_length,width))

y\_test=np.ones((test\_length,1))

for i,line in enumerate(train\_data): # заполняем тренировочный массив

data=re.split(r'\t',line)

# разделяем каждую строку по символу табуляции

x\_train[i][0]=float(data[1]) # заполняем массив координат

x\_train[i][1]=float(data[2])

if data[3]=='red': # заполняем массив y

y\_train[i]=0

else:

y\_train[i]=1

for i,line in enumerate(test\_data): # по аналогии заполняем тестовый массив

data=re.split(r'\t',line)

x\_test[i][0]=float(data[1])

x\_test[i][1]=float(data[2])

if data[3]=='red':

y\_test[i]=0

else:

y\_test[i]=1

print('Первая строка до нормализации',x\_train[0])

means = x\_train.mean(axis=0) # определим среднее

stds = x\_train.std(axis=0) # определим дисперсию

# нормализовать будем следующим образом:

# разделим разницу между значением каждой ячейки массива и дисперсией на среднее

for i in range(x\_train.shape[0]):

for j in range(x\_train.shape[1]):

x\_train[i][j]=(x\_train[i][j]-means[j])/stds[j]

# воспользуемся тем же методом нормализации

for i in range(x\_test.shape[0]):

for j in range(x\_test.shape[1]):

x\_test[i][j]=(x\_test[i][j]-means[j])/stds[j]

# сформируем простую нейронную сеть

# из пакета keras возьмем модель Sequential (с последовательно связанными слоями)

model=Sequential()

#добавим нормального(гауссого) шума ,чтобы избежать переобучения

model.add(GaussianNoise(stddev=0.03,input\_shape=(2,)))

# добавим первый слой нейронов

model.add(Dense(256))

# добавим функцию активации LeakyReLU,

# которая отсекает у функции ReLu нулевые значения,

# изменяя их на a\*x, где в нашем случае a = 0.2

#(https://www.tensorflow.org/api\_docs/python/tf/keras/layers/LeakyReLU

model.add(LeakyReLU(0.2))

# по аналогии создадим второй слой, уменьшив количество нейронов в 2 раза

model.add(Dense(128))

model.add(LeakyReLU(0.2))

# т.к. наша задача - задача бинарной классификации,

# то на выходе нам достаточо 1 нейрона

model.add(Dense(1))

# и функцию активации - sigmoid

model.add(Activation('sigmoid'))

batch=2 # количество элементов единовременно используемых для обучения

epochs=150 # количество циклов обучения

validation\_split=0.25 # количество данных, используемых для валидации непосредственно во время обучения

# для задачи бинарной классификации воспользуемся функцией потерь binary\_crossentropy

model.compile(loss='binary\_crossentropy',optimizer='RMSprop',metrics=['accuracy'])

history=model.fit(x\_train,y\_train,batch\_size=batch,epochs=epochs,verbose=True,validation\_split=validation\_split)

score=model.evaluate(x\_test,y\_test)

print("\nTest score:", score[0])

print('Test accuracy:', score[1])

NB = GaussianNB()

NB.fit(x\_train, y\_train)

BNB=BernoulliNB()

BNB.fit(x\_train, y\_train)

y\_predict\_nb = NB.predict(x\_test)

y2\_predict\_bnb=BNB.predict(x\_test)

print("Accuracy Normal NB: {:.2f}".format(NB.score(x\_test, y\_test)))

print("Accuracy Bernulli NB: {:.2f}".format(BNB.score(x\_test, y\_test)))

LDA = LinearDiscriminantAnalysis()

LDA.fit(x\_train, y\_train)

score\_lda = LDA.score(x\_test, y\_test)

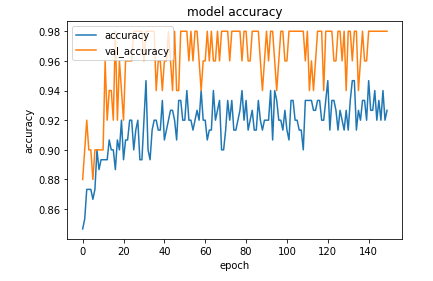
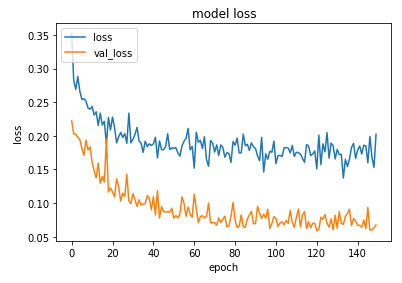
print("Linear Discriminant Analysis: ", score\_lda)

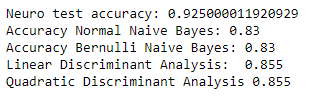
score\_qda = QDA.score(x\_test, y\_test)

print("Quadratic Discriminant Analysis", score\_qda)

**Результат**

Графики точности и потерь для перцептрона(на обучающей выборке):





По полученным данным можно сказать, что многослойный-перцептрон показал наилучший результат, в то время как одно- и много- Байесовский классификатор – наихудший.