МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №8
по дисциплине «Машинное обучение»
Тема: Классификация (линейный дискриминантный анализ, метод опорных векторов)

Студент гр. 1310 _____ Комаров Д. Е.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2025

Постановка задачи

Цель работы: ознакомление с методами классификации модуля Sklearn.

Выполнение лабораторной работы

Загрузка данных

Для выполнения лабораторной работы используем набор данных «Iris». Данный набор данных содержит 150 записей о трех классах цветов. Загрузим данные в датафрейм. Фрагмент результата представлен на рисунке 1.

	0	1	2	3	4
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
..	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

Рисунок 1 – Данные для классификации

Выделим из данных метки классов, преобразуем тексты меток к числам. Разобьём выборку на обучающую и тестовую. Получилось 75 записей в обучающей выборке и 75 записей в тестовой выборке.

Линейный дискриминантный анализ

Линейный дискриминантный анализ линейно преобразует исходное пространство признаков в подпространство меньшей размерности с целью минимизировать внутриклассовую дисперсию и максимизировать межклассовую.

Проведем классификацию наблюдений методом линейного дискриминантного анализа. Правильно было классифицировано 73 наблюдения, неправильно – 2. Точность классификации составила 97%.

Классификатор *LinearDiscriminantAnalysis* имеет такие параметры как *solver*, *shrinkage*, *priors*, *n_components*, *store_covariance*, *tol*. Параметр *solver* отвечает за метод вычисления преобразующих векторов. Параметр *shrinkage* отвечает за то, насколько выборочная ковариационная матрицы будет сжата к

диагонали. При значении 0 будет использоваться исходная ковариационная матрица, а при 1 ковариационная матрица будет состоять только из главной диагонали. Параметр *prior* нужен для задания априорных вероятностей классов. Параметр *n_components* отвечает за количество компонент при понижении размерности. Параметр *store_covariance* отвечает за то, будет ли сохранена ковариационная матрица после обучения классификатора. Параметр *tol* отвечает за порог сходимости алгоритмов поиска преобразующих векторов.

Атрибутами *LinearDiscriminantAnalysis* являются *coef_*, *intercept_*, *covariance_*, *explained_variance_ratio_*, *means_*, *priors_*, *scalings_*, *xbar_*, *classes_*, *n_features_in_*. Атрибут *coef_* хранит коэффициенты линейных дискриминантных функций. Атрибут *intercept_* хранит свободные члены линейных дискриминантных функций. Атрибут *covariance_* хранит ковариационную матрицу. Атрибут *means_* хранит средние значения всех классов. Атрибут *priors_* хранит априорные вероятности классов. Атрибут *explained_variance_ratio_* хранит процент объясненной дисперсии для каждой компоненты. Атрибут *xbar_* хранит средние значения для всех признаков. Атрибут *classes_* хранит имена всех классов. Атрибут *n_features_in_* хранит количество исходных признаков. Атрибут *scalings_* хранит преобразующие вектора.

Построим графики зависимости количества неправильно классифицированных наблюдений и точности классификации от соотношения обучающей и тестовой выборок. График зависимости количества неправильно классифицированных наблюдений от соотношения обучающей и тестовой выборок представлен на рисунке 2.

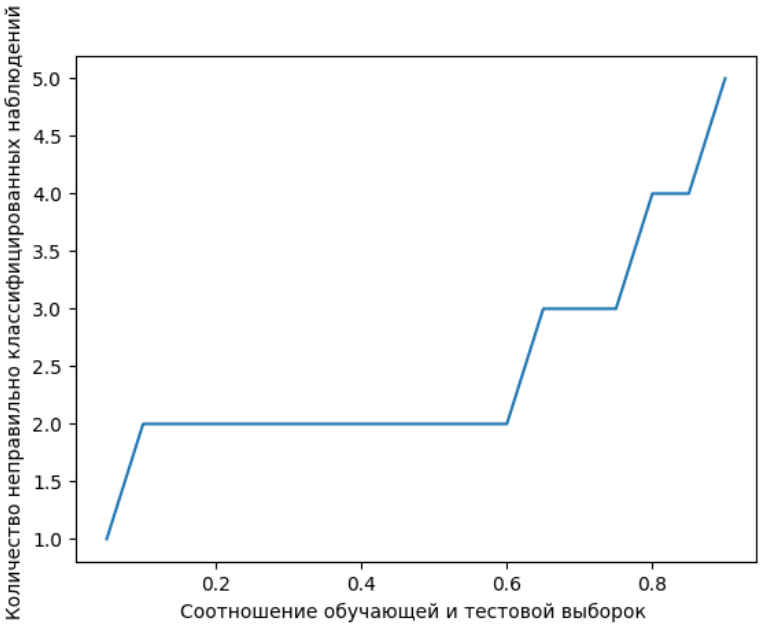

Рисунок 2 – График зависимости количества неправильно классифицированных наблюдений от соотношения обучающей и тестовой выборок для линейного дискриминантного анализа

Как можно увидеть из рисунка 2, при увеличении размера тестовой выборки количество неправильно классифицированных наблюдений также растет, что логично, так как в большей выборке находится больше наблюдений, которые трудно классифицировать. На рисунке 3 представлен график зависимости точности классификации от соотношения обучающей и тестовой выборок.

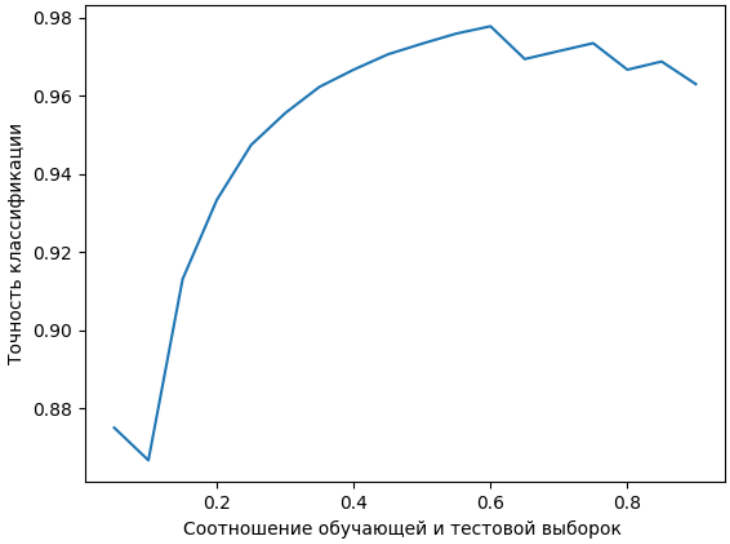

Рисунок 3 – График зависимости точности классификации от соотношения обучающей и тестовой выборок для линейного дискриминантного анализа

Из рисунка 3 можно заметить, что при увеличении соотношения обучающей и тестовой выборки точность также растет, однако после достижения некоторого порога, точность начинает медленно падать, что объясняется переобучением классификатора.

Метод *transform* преобразует данные в подпространство, в котором происходит классификация. Результат применения данного метода представлен на рисунке 4.

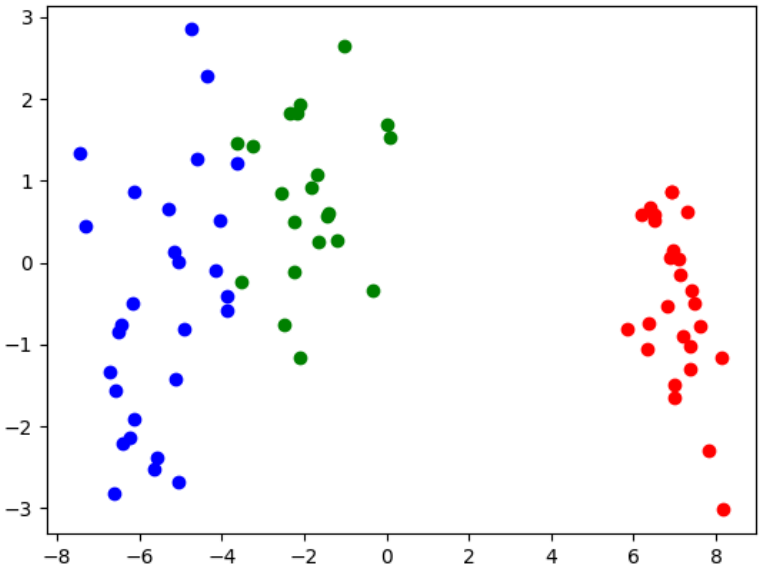

Рисунок 4 – Данные после применения метода *transform*

Как можно увидеть из рисунка 4, после применения метода *transform* данные стали хорошо разделимы по классам по компоненте, отображенной на горизонтальной оси.

Проведем классификацию при различных параметрах *solver*. По умолчанию используется *svd* – сингулярное разложение. Изменим значение на *eigen* – вычисление собственных чисел. Точность классификации составила 97%, неправильно было классифицировано 2 наблюдения. Изменим значение на *lsqr* –метод наименьших квадратов. Точность классификации составила 97%, неправильно было классифицировано 2 наблюдения. Как можно заметить данный параметр не влияет на точность классификации.

Проведем классификацию при различных параметрах *shrinkage*. Данный параметр доступен только при значениях solver, равных *eigen* или *lsqr*. По

умолчанию используется значение *None*, при котором используется исходная ковариационная матрица. Изменим значение на 0.5. Все наблюдения были правильно классифицированы, точность составила 100%. Изменим значение на 1. Точность составила 93%, неправильно было классифицировано 5 наблюдений.

Зададим априорную вероятность классу 1, равную 0.7, классу 2 0.15, классу 3 0.15. Точность составила 97%, неправильно было классифицировано 2 наблюдения.

Метод опорных векторов

Метод опорных векторов разделяет классы при помощи гиперплоскости. Гиперплоскость выбирается таким образом, чтобы расстояние от нее до ближайших к границе разделения точек, называемых опорными векторами, было максимально.

Проведем классификацию методом опорных векторов на тех же данных. Точность классификации составила 97%, неправильно было классифицировано 2 наблюдения.

Атрибут *support_vectors_* хранит опорные вектора, атрибут *supports_* хранит индексы точек, к которым эти опорные вектора принадлежат, атрибут *n_support_* хранит количество опорных векторов для каждого класса.

Построим графики зависимости количества неправильно классифицированных наблюдений и точности классификации от соотношения обучающей и тестовой выборок. График зависимости количества неправильно классифицированных наблюдений от соотношения обучающей и тестовой выборок представлен на рисунке 5.

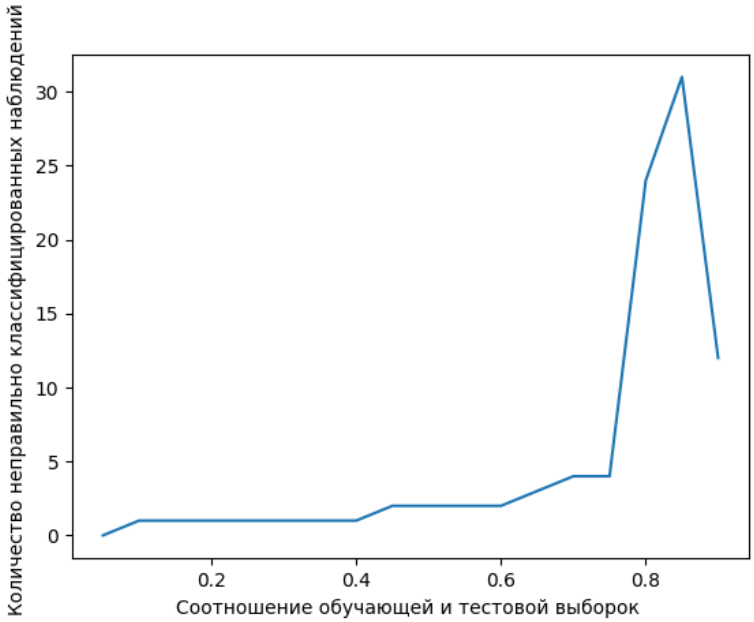

Рисунок 5 – График зависимости количества неправильно классифицированных наблюдений от соотношения обучающей и тестовой выборок для метода опорных векторов

Как можно увидеть из рисунка 5, при увеличении размера тестовой выборки количество неправильно классифицированных наблюдений постоянно до того момента, пока классификатор не достигает переобучения. На рисунке 6 представлен график зависимости точности классификации от соотношения обучающей и тестовой выборок.

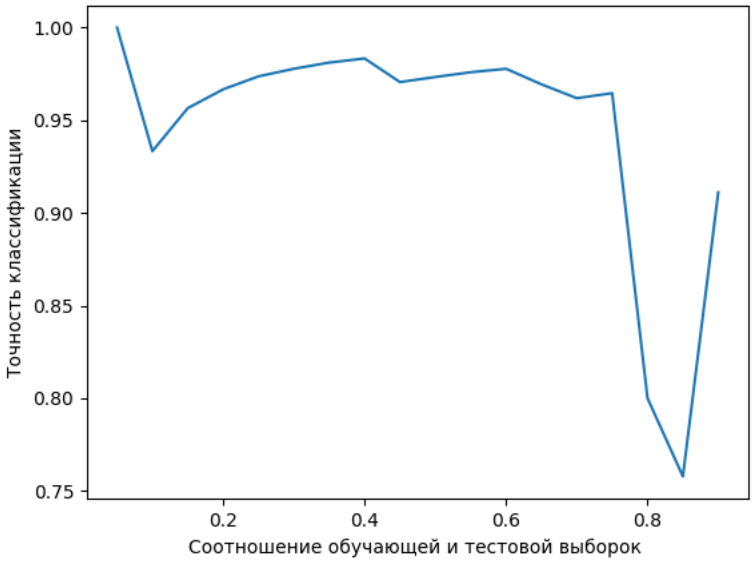

Рисунок 6 – График зависимости точности классификации от соотношения обучающей и тестовой выборок для метода опорных векторов

Из рисунка 6 можно заметить, что при увеличении соотношения обучающей и тестовой выборки точность постоянна до того момента, пока классификатор не достигает переобучения.

Метод опорных векторов может также работать для данных после ядерных преобразований. По умолчанию параметр *kernel* равен *rbf*. Проведём классификацию с параметром *kernel* равным *sigmoid*. Неправильно было классифицировано 53 наблюдения, точность классификации составила 29%. Проведём классификацию с параметром *kernel* равным *linear*. Неправильно было классифицировано 1 наблюдение, точность классификации составила 99%.

Проведём классификацию с параметром *kernel* равным *poly*. Данное ядро имеет параметр *degree*. Зададим его равным 1 и проведем классификацию. Неправильно было классифицировано 2 наблюдения, точность классификации составила 97%. Изменим значение *degree* на 5 и проведем классификацию. Неправильно было классифицировано 1 наблюдение, точность классификации составила 99%.

Как можно заметить, ядра *linear* и *poly* дали наилучший результат.

Параметр *max_iter* отвечает за количество итераций оптимизатора. По умолчанию количество итераций не ограничено. Изменим значение на 1 и проведем классификацию. Неправильно было классифицировано 9 наблюдений, точность классификации составила 88%. Изменим значение на 10 и проведем классификацию. Неправильно было классифицировано 2 наблюдения, точность классификации составила 97%. Как можно заметить при увеличении количества итераций точность классификации растет.

Метод NuSVC отличается от обычного метода опорных векторов (SVC) тем, что SVC в качестве параметра использует штраф за ошибки, а NuSVC в качестве параметра задает долю опорных векторов. Классифицируем те же данные методом NuSVC при параметрах по умолчанию. Неправильно было классифицировано 2 наблюдения, точность классификации составила 97%.

Метод LinearSVC является оптимизированной версией метода опорных векторов для случая линейного ядра. Классифицируем те же данные методом LinearSVC при параметрах по умолчанию. Неправильно было классифицировано 5 наблюдений, точность классификации составила 93%.

Выводы

В ходе выполнения лабораторной работы было проведено ознакомление с методами классификации модуля Sklearn.

Был изучен метод линейного дискриминантного анализа. Данный метод преобразует исходное пространство признаков в подпространство меньшей размерности с целью минимизировать внутриклассовую дисперсию и максимизировать межклассовую. Метод был протестирован при различных параметрах. Также были построены графики зависимости неправильно классифицированных наблюдений и точности классификации от соотношения обучающей и тестовой выборок.

Был изучен метод опорных векторов. Данные метод разделяет классы при помощи гиперплоскости, выбираемой таким образом, чтобы расстояние от нее до ближайших к границе разделения точек, называемых опорными векторами, было максимально. Метод был протестирован при различных параметрах и различных ядерных преобразованиях. Также были построены графики зависимости неправильно классифицированных наблюдений и точности классификации от соотношения обучающей и тестовой выборок.

Код программы, написанной для выполнения лабораторной работы представлен в приложении А.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn import model_selection
from sklearn import discriminant_analysis
from sklearn import svm
import matplotlib.pyplot as plt

data = pd.read_csv('iris.data',header=None)
print(data)
X = data.iloc[:,:4].to_numpy()
labels = data.iloc[:,4].to_numpy()
le = preprocessing.LabelEncoder()
Y = le.fit_transform(labels)

X_train, X_test, y_train, y_test = model_selection.train_test_split(X, Y,
test_size=0.5,random_state=5)
lda = discriminant_analysis.LinearDiscriminantAnalysis()
y_pred = lda.fit(X_train, y_train).predict(X_test)
print("LDA:",(y_test != y_pred).sum(),lda.score(X_test,y_test))

test_size=[s/100 for s in range(5,95,5)]
nc=[]
sc=[]
for ts in test_size:
    lda=discriminant_analysis.LinearDiscriminantAnalysis()
    X_train, X_test, y_train, y_test = model_selection.train_test_split(X, Y,
test_size=ts,random_state=5)
    y_pred = lda.fit(X_train, y_train).predict(X_test)
    nc.append((y_test != y_pred).sum())
    sc.append(lda.score(X_test,y_test))
plt.plot(test_size,nc)
plt.xlabel("Соотношение обучающей и тестовой выборок ")
plt.ylabel("Количество неправильно классифицированных наблюдений ")
plt.show()
plt.plot(test_size,sc)
plt.xlabel("Соотношение обучающей и тестовой выборок ")
plt.ylabel("Точность классификации")
plt.show()

X_train, X_test, y_train, y_test = model_selection.train_test_split(X, Y,
test_size=0.5,random_state=5)
lda = discriminant_analysis.LinearDiscriminantAnalysis()
y_pred = lda.fit(X_train, y_train).predict(X_test)
X_n_test=lda.transform(X_test)

colors = ['red', 'green', 'blue']
for i, color in zip([0, 1,2], colors):
    plt.scatter(
        X_n_test[y_test == i, 0],
        X_n_test[y_test == i, 1],
        c=color
    )
plt.show()

lda = discriminant_analysis.LinearDiscriminantAnalysis(solver='eigen')
y_pred = lda.fit(X_train, y_train).predict(X_test)
print("LDA + solver=eigen:",(y_test != y_pred).sum(),lda.score(X_test,y_test))
```

```
lda = discriminant_analysis.LinearDiscriminantAnalysis(solver='lsqr')
y_pred = lda.fit(X_train, y_train).predict(X_test)
print("LDA + solver=lsqr:",(y_test != y_pred).sum(),lda.score(X_test,y_test))

lda                                                                    =
discriminant_analysis.LinearDiscriminantAnalysis(shrinkage=0.5,solver='eigen'
)
y_pred = lda.fit(X_train, y_train).predict(X_test)
print("LDA              +              shrinkage=0.5:",(y_test              !=
y_pred).sum(),lda.score(X_test,y_test))

lda                                                                    =
discriminant_analysis.LinearDiscriminantAnalysis(shrinkage=1,solver='eigen')
y_pred = lda.fit(X_train, y_train).predict(X_test)
print("LDA + shrinkage=1:",(y_test != y_pred).sum(),lda.score(X_test,y_test))

lda = discriminant_analysis.LinearDiscriminantAnalysis(priors=[0.7,0.15,0.15])
y_pred = lda.fit(X_train, y_train).predict(X_test)
print("LDA + prior:",(y_test != y_pred).sum(),lda.score(X_test,y_test))

X_train, X_test, y_train, y_test = model_selection.train_test_split(X, Y,
test_size=0.5,random_state=5)
clf = svm.SVC()
y_pred = clf.fit(X_train, y_train).predict(X_test)
print("SVC:",(y_test != y_pred).sum(),clf.score(X_test,y_test))

print(clf.support_vectors_)
print(clf.support_)
print(clf.n_support_)

test_size=[s/100 for s in range(5,95,5)]
nc=[]
sc=[]
for ts in test_size:
    clf=svm.SVC()
    X_train, X_test, y_train, y_test = model_selection.train_test_split(X, Y,
test_size=ts,random_state=5)
    y_pred = clf.fit(X_train, y_train).predict(X_test)
    nc.append((y_test != y_pred).sum())
    sc.append(clf.score(X_test,y_test))
plt.plot(test_size,nc)
plt.xlabel("Соотношение обучающей и тестовой выборок ")
plt.ylabel("Количество неправильно классифицированных наблюдений ")
plt.show()
plt.plot(test_size,sc)
plt.xlabel("Соотношение обучающей и тестовой выборок ")
plt.ylabel("Точность классификации")
plt.show()

X_train, X_test, y_train, y_test = model_selection.train_test_split(X, Y,
test_size=0.5,random_state=5)
clf = svm.SVC(kernel='sigmoid')
y_pred = clf.fit(X_train, y_train).predict(X_test)
print("SVC+kernel=sigmoid:",(y_test != y_pred).sum(),clf.score(X_test,y_test))

clf = svm.SVC(kernel='linear')
y_pred = clf.fit(X_train, y_train).predict(X_test)
print("SVC+kernel=linear:",(y_test != y_pred).sum(),clf.score(X_test,y_test))

clf = svm.SVC(kernel='poly',degree=1)
y_pred = clf.fit(X_train, y_train).predict(X_test)
print("SVC+kernel=poly                degree=1:",(y_test                !=
y_pred).sum(),clf.score(X_test,y_test))
```

```
clf = svm.SVC(kernel='poly',degree=5)
y_pred = clf.fit(X_train, y_train).predict(X_test)
print("SVC+kernel=poly                  degree=5:",(y_test                  !=
y_pred).sum(),clf.score(X_test,y_test))

clf = svm.SVC(max_iter=1)
y_pred = clf.fit(X_train, y_train).predict(X_test)
print("SVC+max_iter=1:",(y_test != y_pred).sum(),clf.score(X_test,y_test))

clf = svm.SVC(max_iter=10)
y_pred = clf.fit(X_train, y_train).predict(X_test)
print("SVC+max_iter=10:",(y_test != y_pred).sum(),clf.score(X_test,y_test))

clf = svm.NuSVC()
y_pred = clf.fit(X_train, y_train).predict(X_test)
print("NuSVC:",(y_test != y_pred).sum(),clf.score(X_test,y_test))

clf = svm.LinearSVC()
y_pred = clf.fit(X_train, y_train).predict(X_test)
print("LinearSVC:",(y_test != y_pred).sum(),clf.score(X_test,y_test))
```