МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе № 8

по дисциплине «Машинное обучение»

Тема: Классификация (линейный дискриминантный анализ, метод опорных векторов)

Студенты гр. 6304 _____ Григорьев И.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы

Ознакомиться с методами классификации модуля *Sklearn*.

Ход работы

Загрузка данных

Датасет загружен в датафрейм. Вид данных представлен на рис. 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

150 rows × 5 columns

Рисунок 1 – Исходные данные

Выделены данные и их метки, тексты меток преобразованы к числам. Выборка разбита на обучающую и тестовую *train_test_split*.

Линейный дискриминантный анализ

1. Проведена классификация наблюдений с помощью *LinearDiscriminantAnalysis*. Выявлено 3 неправильно классифицированных наблюдения. Параметры классификатора представлены в табл. 1. Атрибуты классификатора представлены в табл.2.

Таблица 1 – Параметры *LinearDiscriminantAnalysis*

Параметр	Описание
solver	• «svd»: Разложение по сингулярным числам. Не вычисляет ковариационную матрицу, поэтому рекомендуется для данных с большим количеством признаков. • «lsqr»: Решение наименьших квадратов, можно комбинировать с параметром *shrinkage*. • «eigen»: Разложение на собственные значения, можно комбинировать с параметром *shrinkage*.
shrinkage	• «auto»: Автоматическое сжатие по лемме Ледуа-Вольфа. • float from [0, 1]
priors	Класс априорных вероятностей. По умолчанию пропорции классов выводятся из данных обучения.
n_components	Количество компонентов (<= min (n_classes - 1, n_features)) для уменьшения размерности. Если None, будет установлено значение min (n_classes - 1, n_features). Этот параметр влияет только на метод преобразования *transform*.
store_covariance	Если True, явно вычислить взвешенную ковариационную матрицу внутри класса, когда решатель – «svd». Матрица всегда вычисляется и сохраняется для других решателей.

tol	Абсолютный порог для того, чтобы единичное значение X считалось значимым, используется для оценки ранга X. Измерения, единичные значения которых не значимы, отбрасываются. Используется только если решатель - «svd».

Таблица 2 – Атрибуты *LinearDiscriminantAnalysis*

Атрибут	Описание
coef_	Весовые вектора.
intercept_	Массив прерывания.
covariance_	Взвешенная внутриклассовая ковариационная матрица.
explained_variance_ratio_	Процент дисперсии, объясняемой каждым из выбранных компонентов. Если n_components не задано, то все компоненты сохраняются, а сумма объясненных дисперсий равна 1,0. Доступно только при использовании собственного решателя или «svd».
means_	Средние в классах.
priors_	Вероятности классов.
scalings_	Масштабирование объектов в пространстве, охватываемом центроидами классов. Доступно только для решателей «svd» и «eigen».
xbar_	Общее среднее. Присутствует, только если решатель - «svd».
classes_	Уникальные метки классов.

2. Точность классификации получена с помощью функции score() и составляет 98%.

3. Построен график зависимости неправильно классифицированных наблюдений и точности классификации от размера тестовой выборки. График представлен на рис. 2.

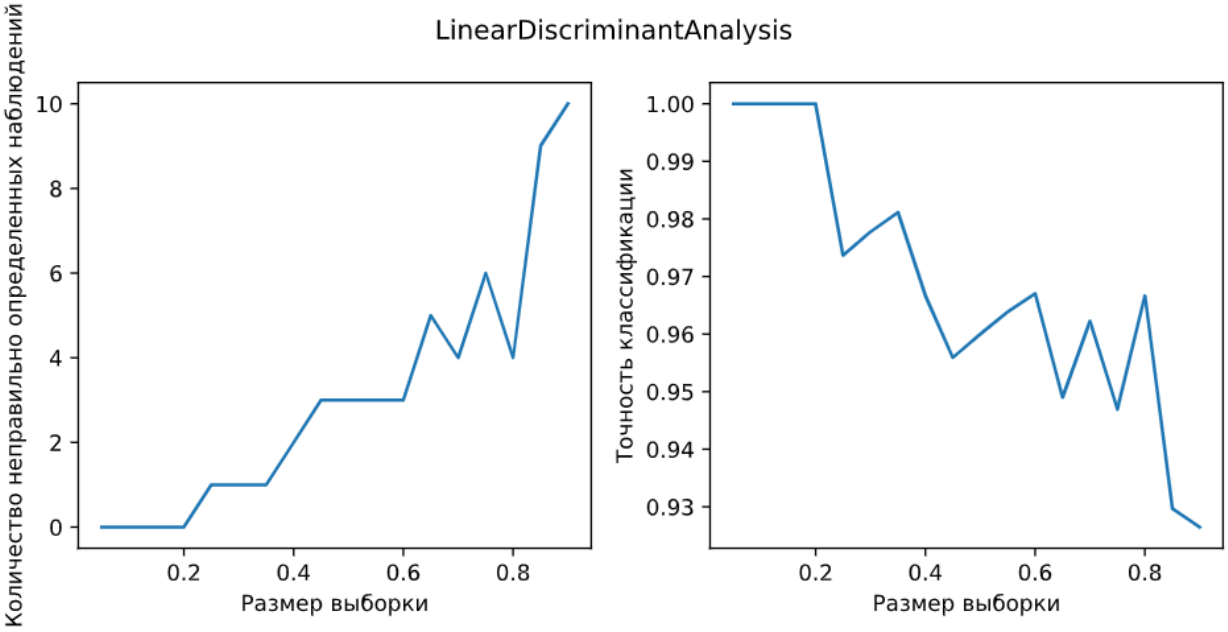

Рисунок 2 – Классификация *LinearDiscriminantAnalysis*

4. Функция transform проецирует данные для максимизации разбиения классов. LDA пытается определить атрибуты, на которые приходится наибольшая разница между классами. В частности, LDA, в отличие от PCA, является контролируемым методом, использующим известные метки классов.

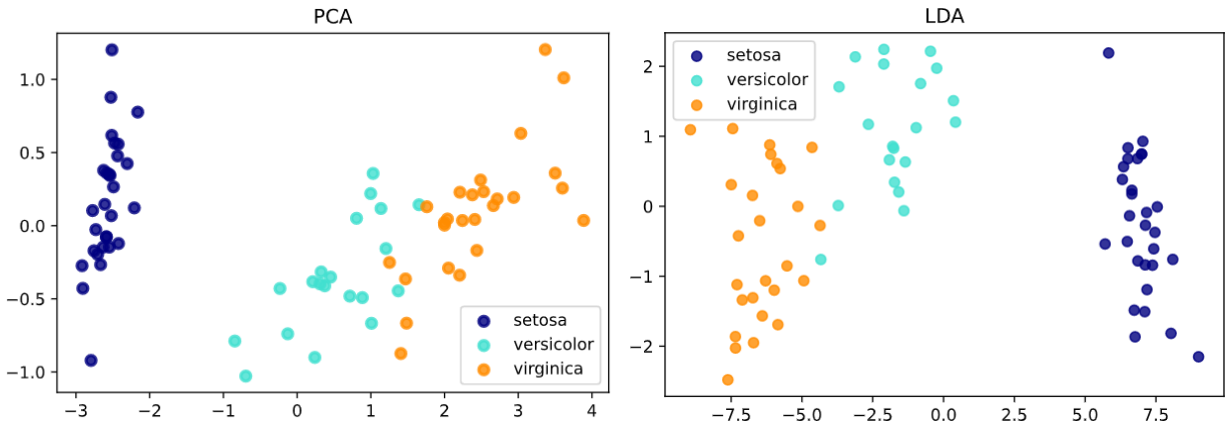

Рисунок 3 – Сравнение PCA и LDA

5. Работа классификатора исследована при различных параметрах solver, shrinkage. Результаты представлены на рис. 4-7.

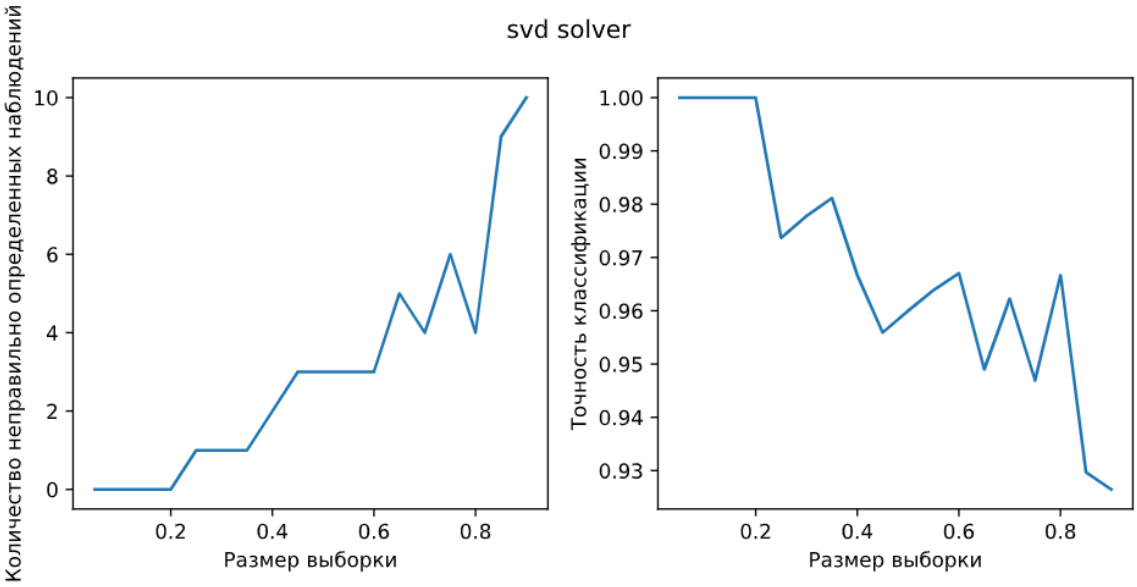

Рисунок 4 – Классификация *LinearDiscriminantAnalysis* с svd solver

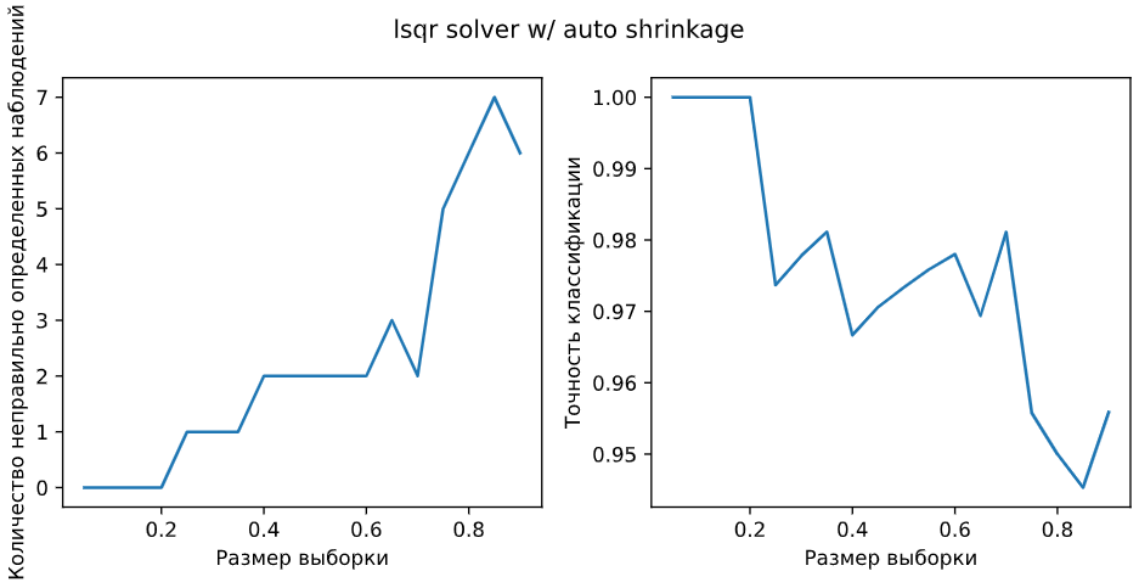

Рисунок 5 – Классификация *LinearDiscriminantAnalysis* с lsqr solver и shrinkage

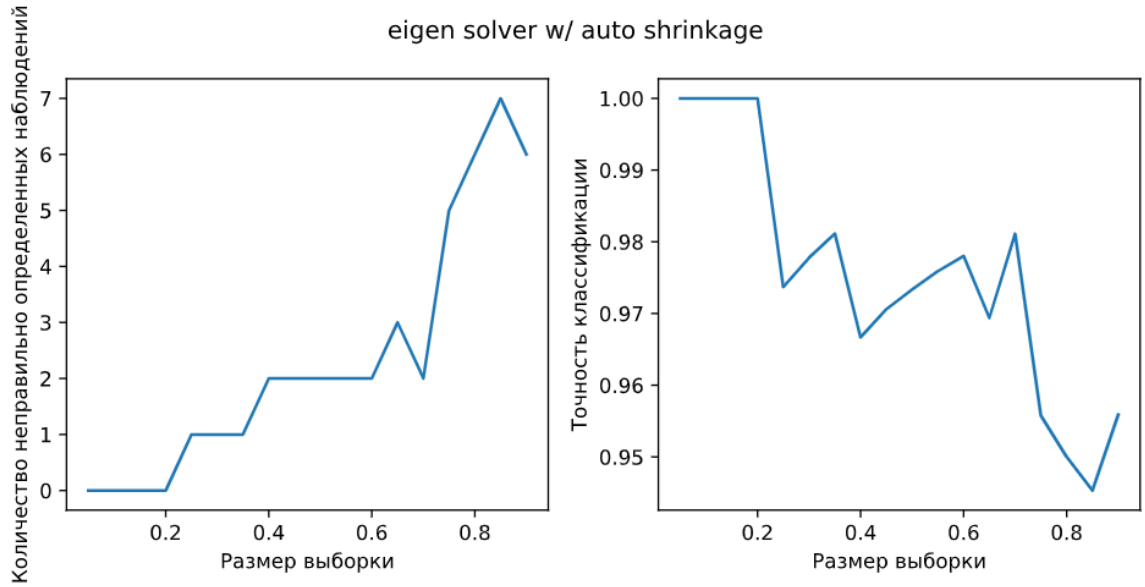

Рисунок 6 – Классификация *LinearDiscriminantAnalysis* с eigen solver и shrinkage

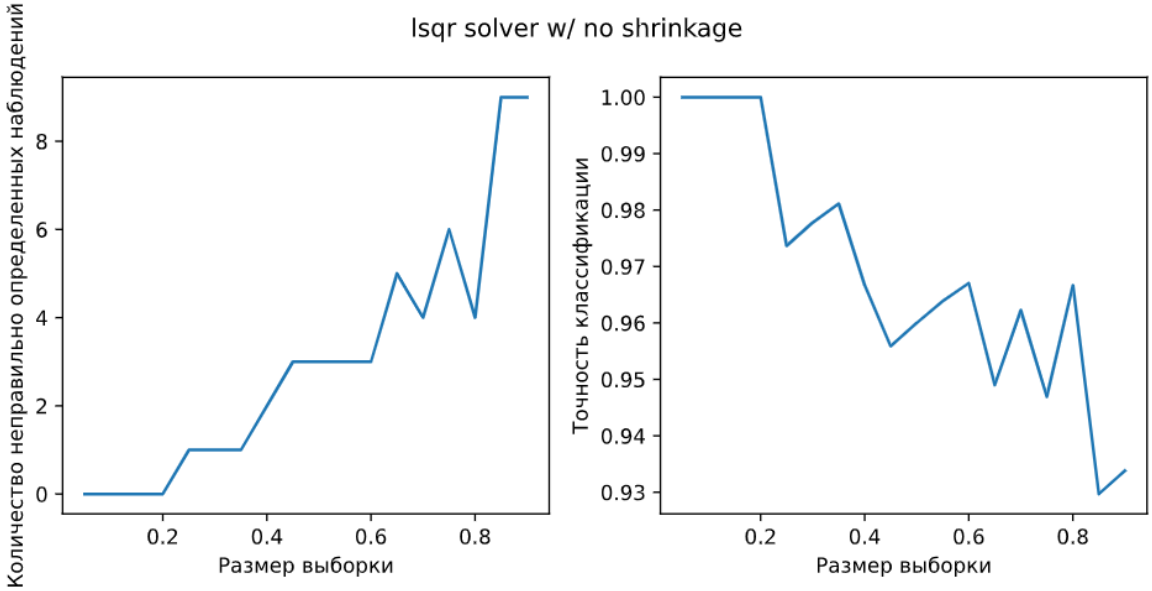

Рисунок 7 – Классификация *LinearDiscriminantAnalysis* с lsqr solver без shrinkage

6. Заданы собственные значения априорных вероятностей классов, результаты представлены в табл. 3 и на рис. 8.

Таблица 3 – Результаты классификации *LinearDiscriminantAnalysis*

Априорные вероятности классов	Количество неправильно определенных наблюдений	Точность классификации
[0.38666667, 0.26666667, 0.34666667]	3	0.987
[0.15, 0.7, 0.15]	5	0.987

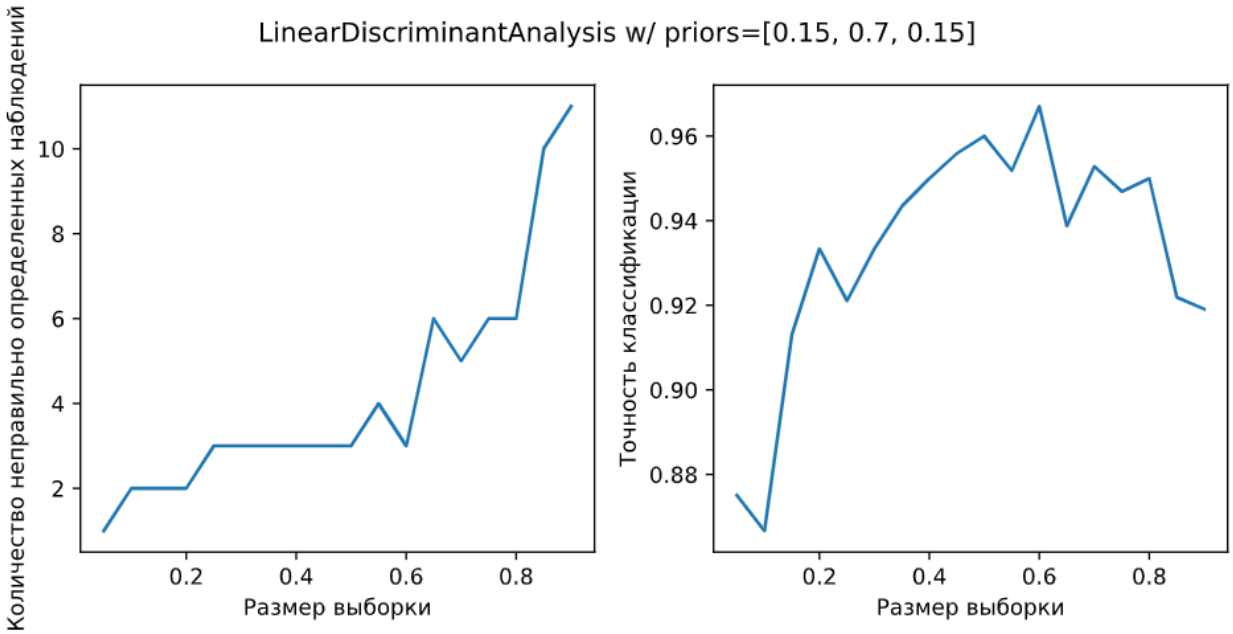

Рисунок 8 – Классификация *LinearDiscriminantAnalysis* с заданными априорными вероятностями

Метод опорных векторов

1. Проведена классификация наблюдений с помощью метода опорных векторов на тех же данных. Выявлено 4 неправильно классифицированных наблюдения.
2. Точность классификации получена с помощью функции score() и составляет 96%.

3. Атрибут *support_* хранит индексы опорных векторов, support_vectors_ – сами опорные вектора, n_support_ – количество опорных векторов для каждого класса.
4. Построен график зависимости неправильно классифицированных наблюдений и точности классификации от размера тестовой выборки. График представлен на рис. 9.

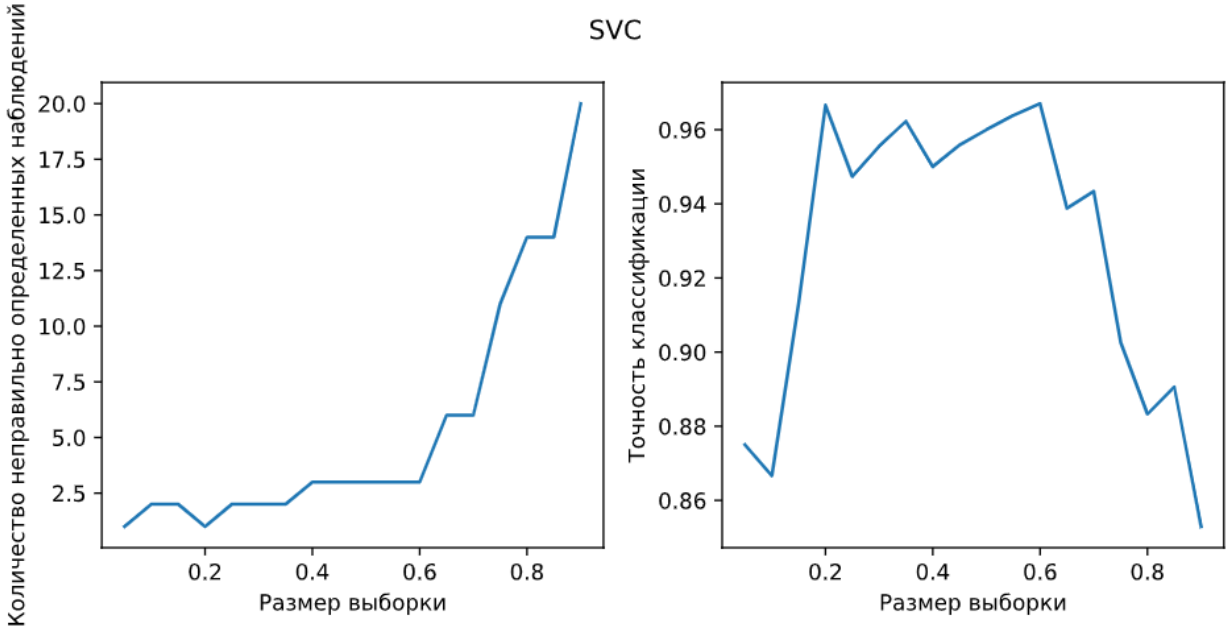

Рисунок 9 – Классификация *SVC*

5. Исследована работа метода опорных векторов при различных значениях параметров *kernel*, *degree*, *max_iter*. Результаты представлены в табл. 4-6.

Таблица 4 – Результаты классификации *SVC*

Тип ядра	Количество неправильно определенных наблюдений	Точность классификации
linear	2	0.973
poly	6	0.987
rbf (default)	4	0.960
sigmoid	54	0.386

Таблица 5 – Результаты классификации *SVC*

Степень	Количество неправильно определенных наблюдений	Точность классификации
1	5	0.960
2	6	0.960
3	6	0.987
4	5	0.987
5	3	0.987

Таблица 6 – Результаты классификации *SVC*

Количество итераций	Количество неправильно определенных наблюдений	Точность классификации
Без ограничений	4	0.960
1	9	0.960
2	8	0.987
3	5	0.973
4	3	0.973
5	1	0.973
6	3	0.973

6. Классификация методами *NuSVC* и *LinearSVC* представлены на рис. 10-11.

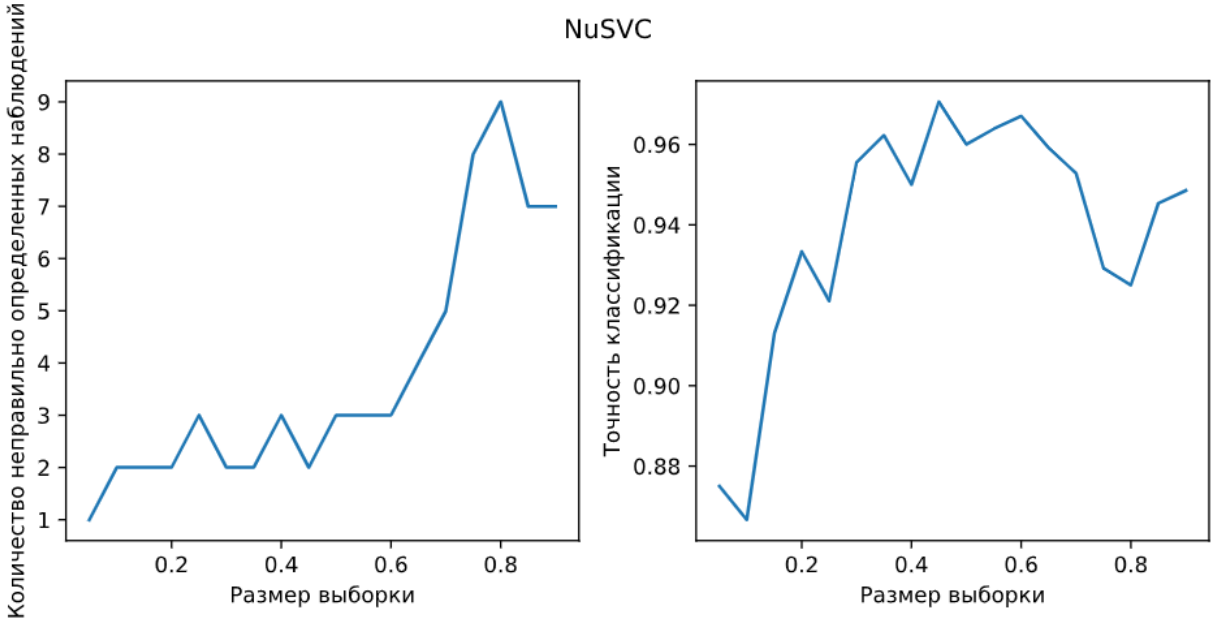

Рисунок 10 – Классификация *NuSVC*

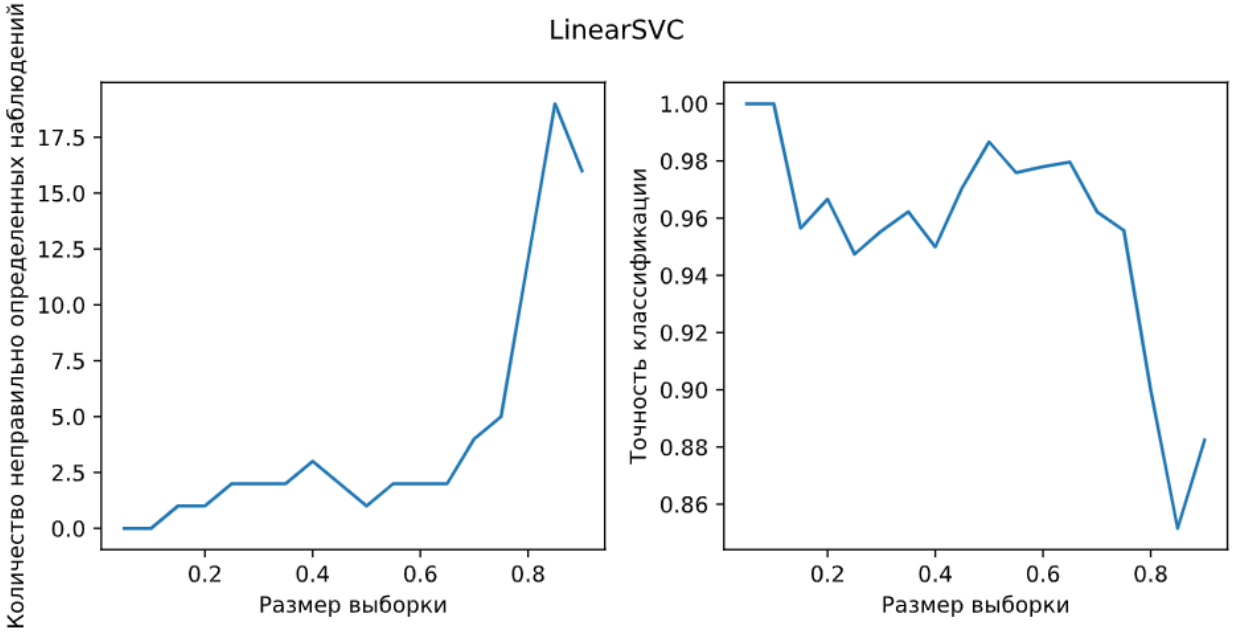

Рисунок 10 – Классификация *LinearSVC*

NuSVC подобен *SVC*, но использует параметр для управления количеством опорных векторов.

LinearSVC аналогично *SVC* с линейным ядром, но лучше масштабируется для большого числа выборок.

Выводы

В ходе лабораторной работы рассмотрены такие методы классификации модуля *Sklearn*, как *LinearDiscriminantAnalysis*, *SVC*, *NuSVC* и *LinearSVC*.

Приложение А

Код программы на python

```
# To add a new cell, type '# %%'
# To add a new markdown cell, type '# %% [markdown]'
# %%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing, svm
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.decomposition import PCA

# %%
data = pd.read_csv('iris.data',header=None)
data

# %%
X = data.iloc[:,:4].to_numpy()
labels = data.iloc[:,4].to_numpy()

# %%
le = preprocessing.LabelEncoder()
Y = le.fit_transform(labels)
Y

# %%
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.5, random_state
=0)

# %%
clf = LinearDiscriminantAnalysis()
y_pred = clf.fit(X_train, y_train).predict(X_test)
print((y_test != y_pred).sum()) #количество наблюдений, который были неправильно опре
делены
clf.priors_

# %%
clf.score(X_train, y_train)

# %%
def plot_clf(clf, title=""):
    test_sizes = np.arange(0.05, 0.95, 0.05)
    wrong_results = []
    scores = []

    for test_size in test_sizes:
        X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size
, random_state=630405)
        y_pred = clf.fit(X_train, y_train).predict(X_test)
        wrong_results.append((y_test != y_pred).sum())
        scores.append(clf.score(X_test, y_test))

    fig, axs = plt.subplots(1, 2, figsize=(8, 4))
    axs[0].plot(test_sizes, wrong_results)
    axs[1].plot(test_sizes, scores)
    axs[0].set_ylabel('Количество неправильно определенных наблюдений')
    axs[1].set_ylabel('Точность классификации')
    axs[0].set_xlabel('Размер выборки')
    axs[1].set_xlabel('Размер выборки')
```

```
    fig.suptitle(title)
    plt.tight_layout()
    plt.show()

# %%
plot_clf(LinearDiscriminantAnalysis(), 'LinearDiscriminantAnalysis')

# %%
target_names = ['setosa', 'versicolor', 'virginica']
y = y_train

pca = PCA(n_components=2)
X_r = pca.fit(X_train).transform(X_train)

X_r2 = clf.transform(X_train)

plt.figure()
colors = ['navy', 'turquoise', 'darkorange']
lw = 2

for color, i, target_name in zip(colors, [0, 1, 2], target_names):
    plt.scatter(X_r[y == i, 0], X_r[y == i, 1], color=color, alpha=.8, lw=lw,
                label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('PCA')

plt.figure()
for color, i, target_name in zip(colors, [0, 1, 2], target_names):
    plt.scatter(X_r2[y == i, 0], X_r2[y == i, 1], alpha=.8, color=color,
                label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('LDA')

plt.show()

# %%
test_sizes = np.arange(0.05, 0.95, 0.05)
wrong_results_1 = []
scores_1 = []
wrong_results_2 = []
scores_2 = []
wrong_results_3 = []
scores_3 = []
wrong_results_4 = []
scores_4 = []

for test_size in test_sizes:
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, ra
ndom_state=630405)

    clf_1 = LinearDiscriminantAnalysis(solver='svd', shrinkage=None)
    y_pred = clf_1.fit(X_train, y_train).predict(X_test)
    wrong_results_1.append((y_test != y_pred).sum())
    scores_1.append(clf_1.score(X_test, y_test))

    clf_2 = LinearDiscriminantAnalysis(solver='lsqr', shrinkage='auto')
    y_pred = clf_2.fit(X_train, y_train).predict(X_test)
    wrong_results_2.append((y_test != y_pred).sum())
    scores_2.append(clf_2.score(X_test, y_test))

    clf_3 = LinearDiscriminantAnalysis(solver='eigen', shrinkage='auto')
    y_pred = clf_3.fit(X_train, y_train).predict(X_test)
```

```
    wrong_results_3.append((y_test != y_pred).sum())
    scores_3.append(clf_3.score(X_test, y_test))

    clf_4 = LinearDiscriminantAnalysis(solver='lsqr', shrinkage=None)
    y_pred = clf_4.fit(X_train, y_train).predict(X_test)
    wrong_results_4.append((y_test != y_pred).sum())
    scores_4.append(clf_4.score(X_test, y_test))

fig, axs = plt.subplots(1, 2, figsize=(8, 4))
axs[0].plot(test_sizes, wrong_results_1)
axs[1].plot(test_sizes, scores_1)
axs[0].set_ylabel('Количество неправильно определенных наблюдений')
axs[1].set_ylabel('Точность классификации')
axs[0].set_xlabel('Размер выборки')
axs[1].set_xlabel('Размер выборки')
fig.suptitle('svd solver')
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(1, 2, figsize=(8, 4))
axs[0].plot(test_sizes, wrong_results_2)
axs[1].plot(test_sizes, scores_2)
axs[0].set_ylabel('Количество неправильно определенных наблюдений')
axs[1].set_ylabel('Точность классификации')
axs[0].set_xlabel('Размер выборки')
axs[1].set_xlabel('Размер выборки')
fig.suptitle('lsqr solver w/ auto shrinkage')
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(1, 2, figsize=(8, 4))
axs[0].plot(test_sizes, wrong_results_3)
axs[1].plot(test_sizes, scores_3)
axs[0].set_ylabel('Количество неправильно определенных наблюдений')
axs[1].set_ylabel('Точность классификации')
axs[0].set_xlabel('Размер выборки')
axs[1].set_xlabel('Размер выборки')
fig.suptitle('eigen solver w/ auto shrinkage')
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(1, 2, figsize=(8, 4))
axs[0].plot(test_sizes, wrong_results_4)
axs[1].plot(test_sizes, scores_4)
axs[0].set_ylabel('Количество неправильно определенных наблюдений')
axs[1].set_ylabel('Точность классификации')
axs[0].set_xlabel('Размер выборки')
axs[1].set_xlabel('Размер выборки')
fig.suptitle('lsqr solver w/ no shrinkage')
plt.tight_layout()
plt.show()

# %%
plot_clf(LinearDiscriminantAnalysis(priors=[0.15, 0.7, 0.15]), 'LinearDiscriminantAna
lysis w/ priors=[0.15, 0.7, 0.15]')

# %%
clf = LinearDiscriminantAnalysis(priors=[0.15, 0.7, 0.15])
y_pred = clf.fit(X_train, y_train).predict(X_test)
print((y_test != y_pred).sum()) #количество наблюдений, который были неправильно опре
делены
clf.priors_
```

```
# %%
clf.score(X_train, y_train)

# %%
clf = svm.SVC()
y_pred = clf.fit(X_train, y_train).predict(X_test)
print((y_test != y_pred).sum())
print(clf.score(X_train, y_train))

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

# %%
clf = svm.SVC(kernel='sigmoid')
y_pred = clf.fit(X_train, y_train).predict(X_test)
print((y_test != y_pred).sum())
print(clf.score(X_train, y_train))

# %%
clf = svm.SVC(kernel='poly')
y_pred = clf.fit(X_train, y_train).predict(X_test)
print((y_test != y_pred).sum())
print(clf.score(X_train, y_train))

# %%
clf = svm.SVC(kernel='poly')
y_pred = clf.fit(X_train, y_train).predict(X_test)
print((y_test != y_pred).sum())
print(clf.score(X_train, y_train))

# %%
clf = svm.SVC(max_iter=6)
y_pred = clf.fit(X_train, y_train).predict(X_test)
print((y_test != y_pred).sum())
print(clf.score(X_train, y_train))

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

# %%
plot_clf(svm.SVC(), 'SVC')

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

# %%
clf = svm.LinearSVC()
y_pred = clf.fit(X_train, y_train).predict(X_test)
print((y_test != y_pred).sum())
print(clf.score(X_train, y_train))

# %%
plot_clf(svm.NuSVC(), 'NuSVC')
```

```
# %%
plot_clf(svm.LinearSVC(), 'LinearSVC')

# %%
plot_clf(svm.SVC(kernel='linear', max_iter=1), 'SVC kernel=linear')

# %%
plot_clf(svm.LinearSVC(max_iter=1), 'SVC kernel=linear')

# %%
```