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

ОТЧЕТ

по лабораторной работе №8

по дисциплине «Машинное обучение»

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

Студент гр. 8304 _____ Кирьянов Д.И.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Ознакомиться с методами классификации модуля Sklearn

Ход работы.

1. Загрузка данных

1.1. Произведена загрузка данных. Создан Python скрипт. Данные загружены в датафрейм.

	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

1.2. Выделены данные и их метки, преобразованы тексты меток к числам, разбита выборка на обучающую и тестовую.

2. Линейный дискриминантный анализ

2.1. Проведена классификация данных при помощи LDA. Количество найденных неправильно классифицированных наблюдений = 3.

Атрибуты:

coef_ - векторы веса.

intercept_ - срок перехвата.

covariance_ - взвешенная внутриклассовая матрица ковариаций.

explained_variance_ratio_ - процент отклонения для каждого выбранного компонента.

means_ - классовые средние.

priors_ - приоры класса.

scalings_ - масштабирование объектов в пространстве, охватываемом центрами классов.

xbar_ - общее среднее.

classes_ - уникальные лейблы класса.

n_features_in_ - количество видимых деталей во время посадки.

feature_names_in_ - названия видимых особенностей во время посадки.

```
Attributes
----------
coef_ : ndarray of shape (n_features,) or (n_classes, n_features)
    Weight vector(s).

intercept_ : ndarray of shape (n_classes,)
    Intercept term.

covariance_ : array-like of shape (n_features, n_features)
    Weighted within-class covariance matrix. It corresponds to
    `sum_k prior_k * C_k` where `C_k` is the covariance matrix of the
    samples in class `k`. The `C_k` are estimated using the (potentially
    shrunk) biased estimator of covariance. If solver is 'svd', only
    exists when `store_covariance` is True.

explained_variance_ratio_ : ndarray of shape (n_components,)
    Percentage of variance explained by each of the selected components.
    If ``n components`` is not set then all components are stored and the
    sum of explained variances is equal to 1.0. Only available when eigen
    or svd solver is used.

means_ : array-like of shape (n_classes, n_features)
    Class-wise means.

priors_ : array-like of shape (n_classes,)
    Class priors (sum to 1).
```

```
scalings_ : array-like of shape (rank, n_classes - 1)
    Scaling of the features in the space spanned by the class centroids.
    Only available for 'svd' and 'eigen' solvers.

xbar_ : array-like of shape (n_features,)
    Overall mean. Only present if solver is 'svd'.

classes_ : array-like of shape (n_classes,)
    Unique class labels.

n features in  : int
    Number of features seen during :term:`fit`.

    .. versionadded:: 0.24

feature names in  : ndarray of shape (`n features in `,)
    Names of features seen during :term:`fit`. Defined only when `X`
    has feature names that are all strings.
```

Параметры:

solver – используемый метод решения.

shrinkage – параметр усадки.

priors – класс априорных вероятностей.

n_components – количество компонентов для уменьшения размерности.

store_covariance – флаг для вычисления взвешенной ковариационной матрицы внутри класса.

tol – абсолютный порог, чтобы единичное значение X считалось значимым, используется для оценки ранга X.

covariance_estimator – используется для оценки ковариационных матриц вместо эмпирической оценки ковариации.

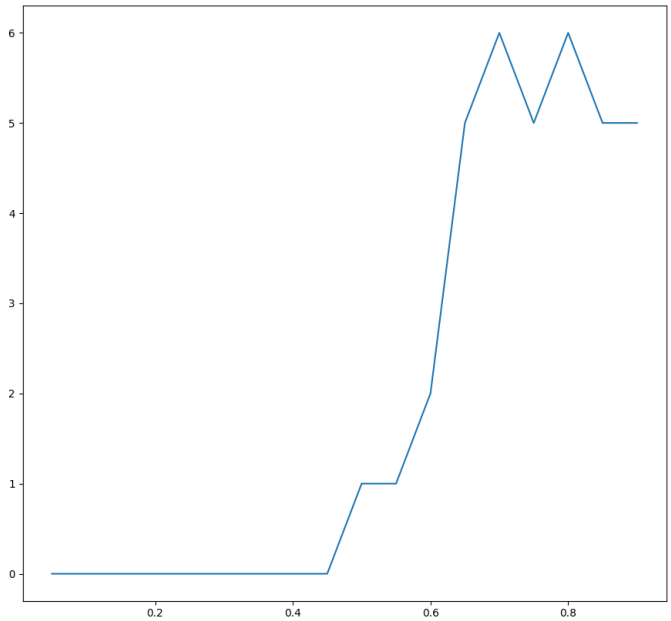
```
solver : {'svd', 'lsqr', 'eigen'}, default='svd'
    Solver to use, possible values:
      - 'svd': Singular value decomposition (default).
        Does not compute the covariance matrix, therefore this solver is
        recommended for data with a large number of features.
      - 'lsqr': Least squares solution.
        Can be combined with shrinkage or custom covariance estimator.
      - 'eigen': Eigenvalue decomposition.
        Can be combined with shrinkage or custom covariance estimator.

shrinkage : 'auto' or float, default=None
    Shrinkage parameter, possible values:
      - None: no shrinkage (default).
      - 'auto': automatic shrinkage using the Ledoit-Wolf lemma.
      - float between 0 and 1: fixed shrinkage parameter.

    This should be left to None if `covariance_estimator` is used.
    Note that shrinkage works only with 'lsqr' and 'eigen' solvers.

priors : array-like of shape (n_classes,), default=None
```

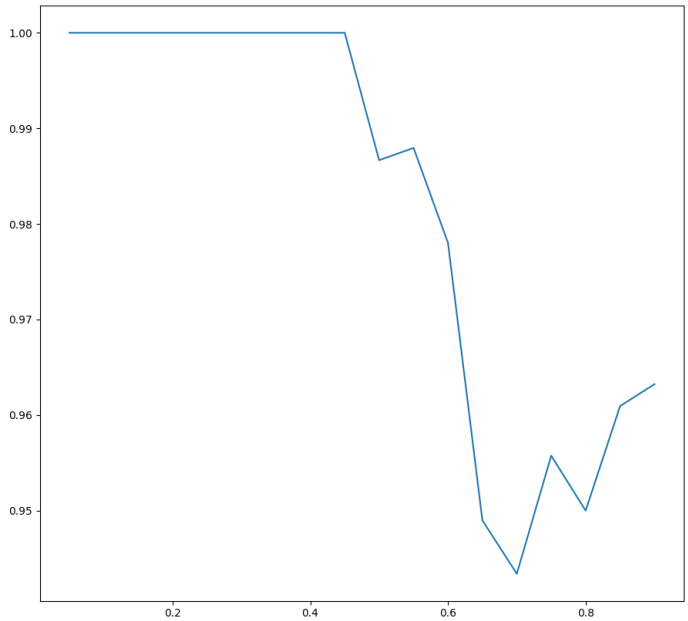
```
    The class prior probabilities. By default, the class proportions are
    inferred from the training data.

n_components : int, default=None
    Number of components (<= min(n_classes - 1, n_features)) for
    dimensionality reduction. If None, will be set to
    min(n_classes - 1, n_features). This parameter only affects the
    `transform` method.

store_covariance : bool, default=False
    If True, explicitly compute the weighted within-class covariance
    matrix when solver is 'svd'. The matrix is always computed
    and stored for the other solvers.

    .. versionadded:: 0.17

tol : float, default=1.0e-4
    Absolute threshold for a singular value of X to be considered
    significant, used to estimate the rank of X. Dimensions whose
    singular values are non-significant are discarded. Only used if
    solver is 'svd'.

    .. versionadded:: 0.17

covariance_estimator : covariance estimator, default=None
    If not None, `covariance_estimator` is used to estimate
    the covariance matrices instead of relying on the empirical
    covariance estimator (with potential shrinkage).
    The object should have a fit method and a ``covariance `` attribute
    like the estimators in :mod:`sklearn.covariance`.
    if None the shrinkage parameter drives the estimate.
```

2.2. Используя функцию score() выведена точность классификации

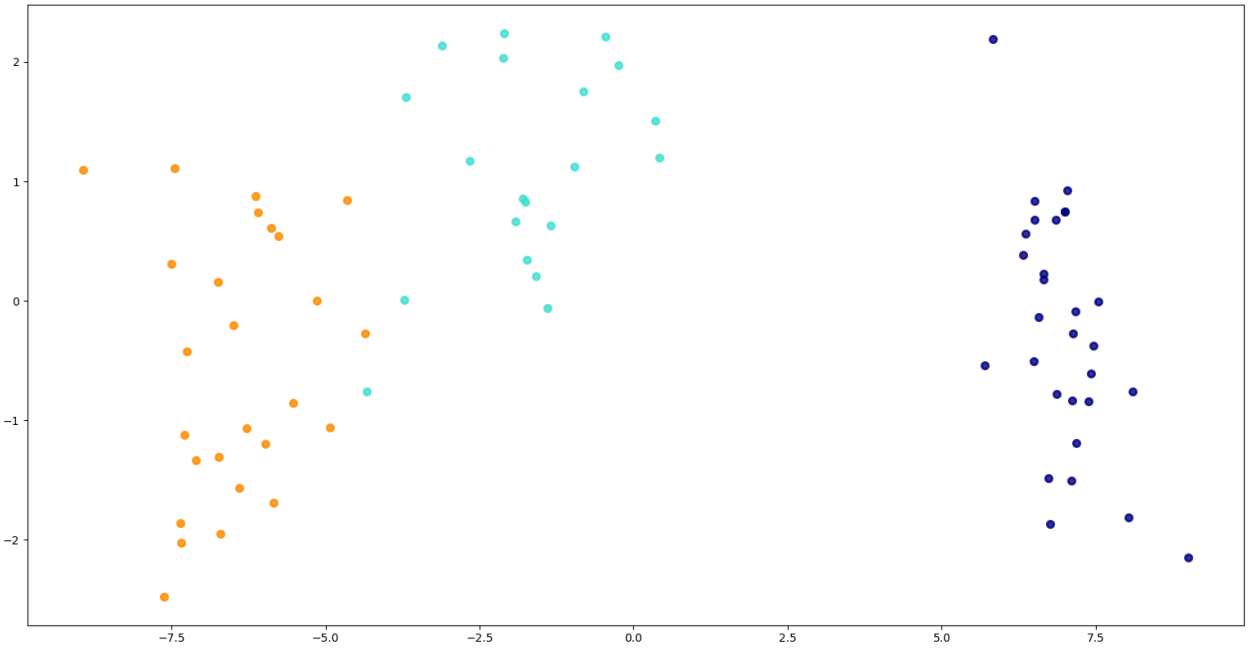
Score = 0.9866666666666667

2.3. Построен график зависимости неправильно классифицированных наблюдений и точности классификации от размера тестовой выборки.





2.4. Применена функция transform и визуализированы ее результаты.



Функция transform применяется для уменьшения размерности данных.

2.5. Исследована работа классификатора при различных параметрах solver, shrinkage.

solver:

svd - разложение по сингулярным значениям (по умолчанию). Не вычисляет ковариационную матрицу, поэтому этот метод решения рекомендуется для данных с большим количеством функций.

lsqr - решение методом наименьших квадратов. Можно комбинировать с оценкой усадки или настраиваемой ковариационной оценкой.

eigen - разложение по собственным значениям. Можно комбинировать с оценкой усадки или настраиваемой ковариационной оценкой.

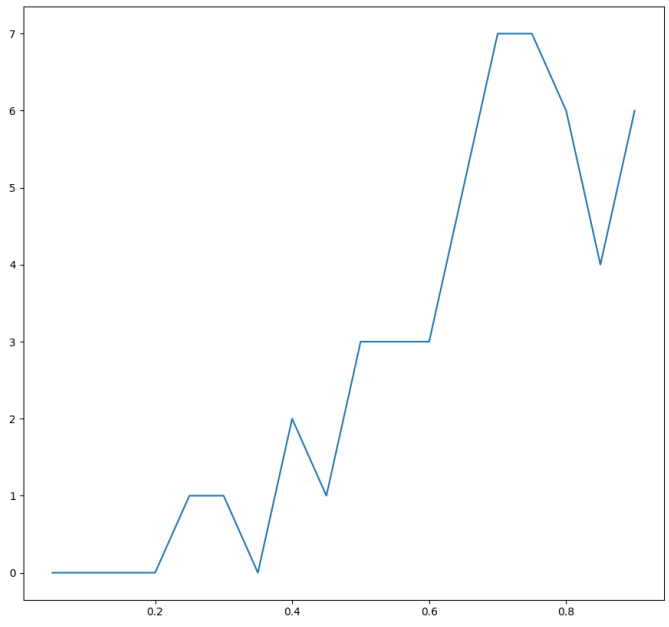
shrinkage:

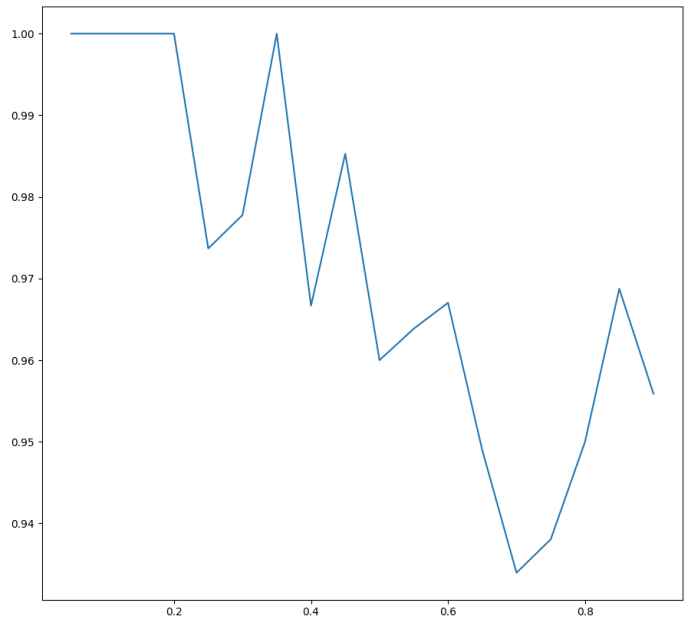
None – без усадки. (по умолчанию)

auto - автоматическая усадка с использованием леммы Ледуа-Вольфа.

float – значение между 0 и 1. фиксированный параметр усадки.

2.6. Задана априорную вероятность класса с номером 1 равная 0.7, остальным классам заданы равные априорные вероятности.





3. Метод опорных векторов

3.1. Проведена классификация при помощи SVM на тех же данных. Количество найденных неправильно классифицированных наблюдений = 4.

3.2. Используя функцию score() выведена точность классификации Score = 0.9533333333333334

3.3. Выведена следующая информация.

```
[[4.5 2.3 1.3 0.3]
 [5.4 3.9 1.7 0.4]
 [5.1 3.3 1.7 0.5]
 [5.  3.  1.6 0.2]
 [5.1 2.5 3.  1.1]
 [6.2 2.2 4.5 1.5]
 [5.7 2.9 4.2 1.3]
 [5.7 2.8 4.5 1.3]
 [6.6 3.  4.4 1.4]
 [6.4 2.9 4.3 1.3]
 [4.9 2.4 3.3 1. ]
 [6.7 3.1 4.4 1.4]
 [5.7 2.6 3.5 1. ]
 [6.3 2.5 4.9 1.5]
 [6.7 3.  5.  1.7]
 [5.5 2.4 3.7 1. ]
 [6.6 2.9 4.6 1.3]
 [5.6 3.  4.1 1.3]
 [5.9 3.2 4.8 1.8]
 [6.3 2.3 4.4 1.3]
 [5.9 3.  5.1 1.8]
 [6.4 2.8 5.6 2.1]
 [6.5 3.2 5.1 2. ]
 [6.2 3.4 5.4 2.3]
 [5.7 2.5 5.  2. ]
 [6.9 3.1 5.4 2.1]
 [7.2 3.  5.8 1.6]
 [7.9 3.8 6.4 2. ]
 [6.  3.  4.8 1.8]
 [6.4 3.2 5.3 2.3]
 [6.7 3.  5.2 2.3]
 [5.8 2.7 5.1 1.9]
 [6.3 2.9 5.6 1.8]]
[16 26 36 59  2  4  6 33 34 37 40 42 54 57 58 60 64 65 66 67  1 11 14 17
 19 20 23 41 44 55 56 62 71]
[ 4 16 13]
```

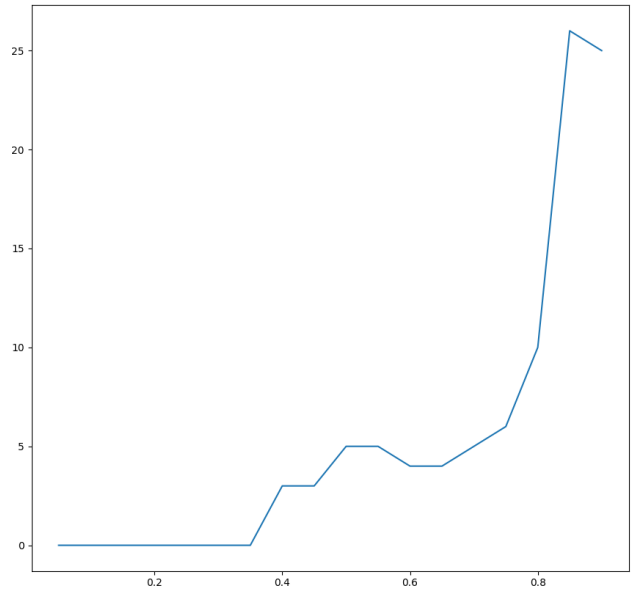
Это информация об опорных векторах.

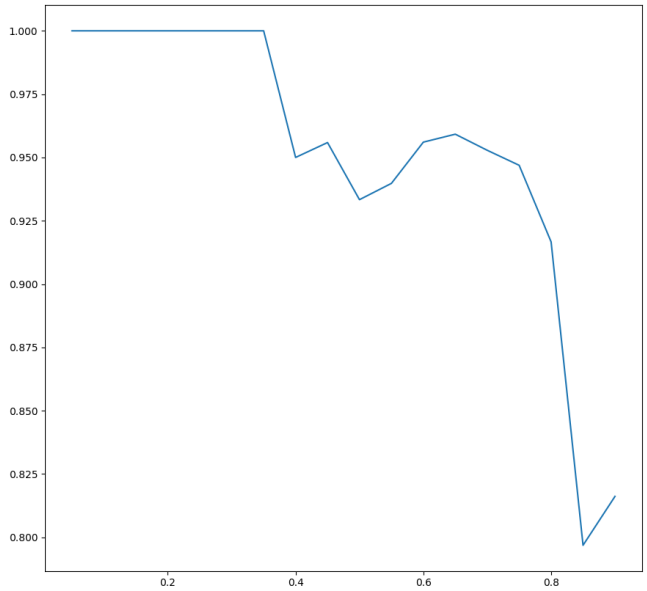
support_ хранит индексы опорных векторов.

support_vectors_ хранит сами опорные вектора.

n_support_ хранит количество опорных векторов для каждого класса.

3.4. Построен график зависимости неправильно классифицированных наблюдений и точности классификации от размера тестовой выборки.





3.5. Исследована работа метода опорных векторов при различных значениях kernel, degree, max_iter.

kernel – тип ядра, который будет использоваться внутри алгоритма.

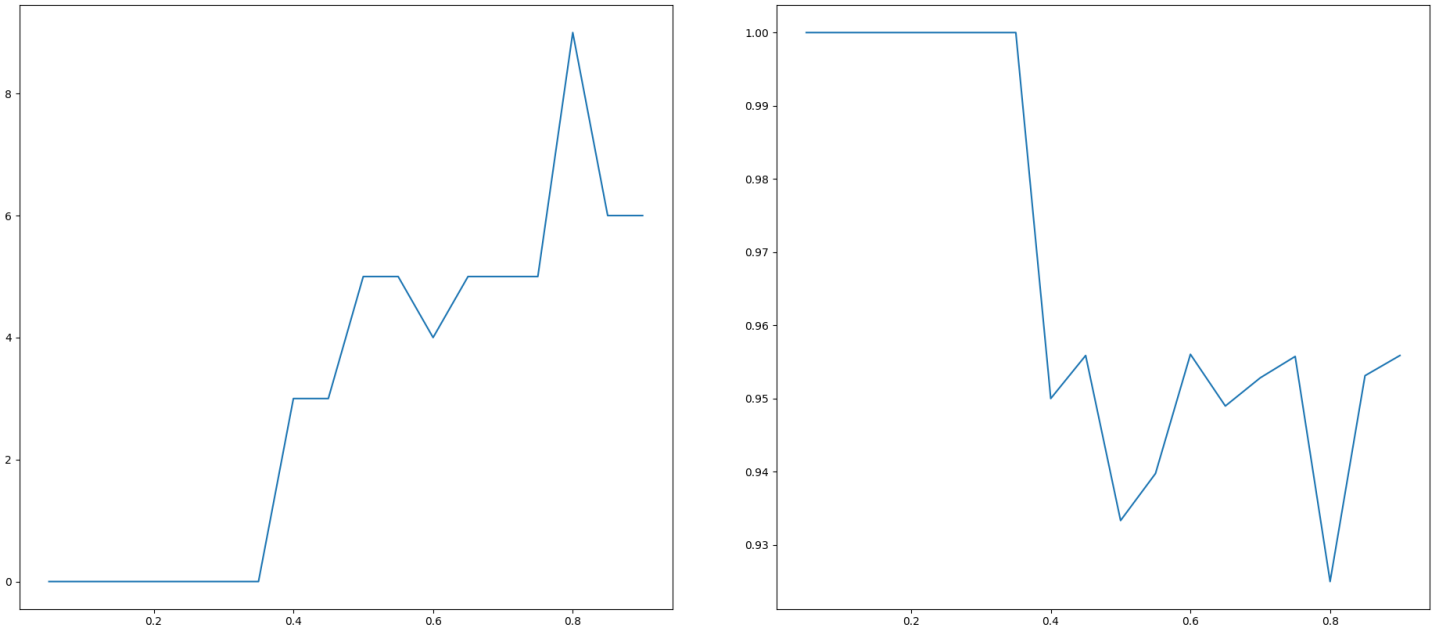
kernel: {'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}

degree – степень полиномиальной функции ядра. Только при kernel = 'poly'.

max_iter – ограничение на количество итераций. При -1 неограниченно.

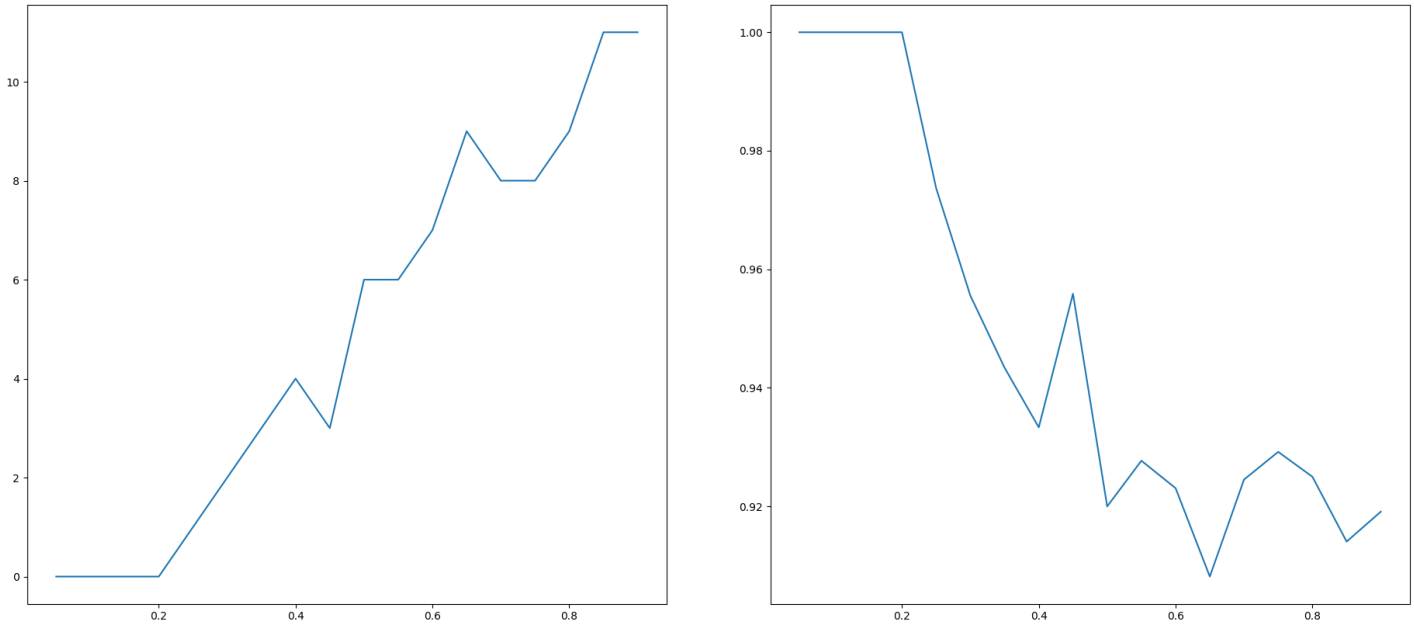
3.6. Проведено исследование для методов NuSVC и LinearSVC.

NuSVC имеет параметр для управления количеством опорных векторов.



NuSVC

LinearSVC аналогичен SVC при kernel = linear, но лучше масштабируется.



LinearSVC

Выводы

В ходе выполнения лабораторной работы было произведено знакомство с классификацией методами GaussianNB, MultinominalNB, ComplementNB, BernoulliNB и DecisionTreeClassifier модуля Sklearn.

ПРИЛОЖЕНИЕ А

Исходный код программы

```
import pandas as pd
import numpy as np
from sklearn import preprocessing, svm
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
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 = train_test_split(X, Y, test_size=0.5)
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())
print(clf.score(X_train, y_train))

def grafics(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=830406)
        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)
    fig.suptitle(title)
    plt.tight_layout()
    plt.show()


grafics(LinearDiscriminantAnalysis(), 'LinearDiscriminantAnalysis')

target_names = ['setosa', 'versicolor', 'virginica']
y = y_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_r2[y == i, 0], X_r2[y == i, 1], color=color, alpha=.8,
lw=lw,
                label=target_name)
```

```
plt.show()

grafics(LinearDiscriminantAnalysis(priors=[0.15, 0.7, 0.15]), 'priors=[0.15,
0.7, 0.15]')

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

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

grafics(svm.SVC(), 'SVC')
grafics(svm.NuSVC(), 'NuSVC')
grafics(svm.LinearSVC(), 'LinearSVC')
```