Московский Авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Курсовая работа по курсу «Объектно-ориентированное программирование»

Тема «Машинное обучение, классификация»

Выполнил: Маслихин Н.А.

Группа: М8О-205Б-19

Руководитель: Семенов А. С.

Введение

Машинное обучение (Machine Learning) — обширный подраздел искусственного интеллекта, изучающий методы построения алгоритмов, способных обучаться. Различают два типа обучения. Обучение по прецедентам, или индуктивное обучение, основано на выявлении общих закономерностей по частным эмпирическим данным. Дедуктивное обучение предполагает формализацию знаний экспертов и их перенос в компьютер в виде базы знаний. Дедуктивное обучение принято относить к области экспертных систем, поэтому термины машинное обучение и обучение по прецедентам можно считать синонимами.

Машинное обучение находится на стыке математической статистики, методов оптимизации и классических математических дисциплин, но имеет также и собственную специфику, связанную с проблемами вычислительной эффективности и переобучения. Многие методы индуктивного обучения разрабатывались как альтернатива классическим статистическим подходам. Многие методы тесно связаны с извлечением информации и интеллектуальным анализом данных (Data Mining).

Наиболее теоретические разделы машинного обучения объединены в отдельное направление, теорию вычислительного обучения (Computational Learning Theory, COLT).

Машинное обучение — не только математическая, но и практическая, инженерная дисциплина. Чистая теория, как правило, не приводит сразу к методам и алгоритмам, применимым на практике. Чтобы заставить их хорошо работать, приходится изобретать дополнительные эвристики, компенсирующие несоответствие сделанных в теории предположений условиям реальных задач. Практически ни одно исследование в машинном обучении не обходится без эксперимента на модельных или реальных данных, подтверждающего практическую работоспособность метода.

Дано конечное множество прецедентов (объектов, ситуаций), по каждому из которых собраны (измерены) некоторые данные. Данные о прецеденте называют также его описанием. Совокупность всех имеющихся описаний прецедентов называется обучающей выборкой. Требуется по этим частным данным выявить общие зависимости, закономерности, взаимосвязи, присущие не только этой конкретной выборке, но вообще всем прецедентам, в том числе тем, которые ещё не наблюдались. Говорят также о восстановлении зависимостей по эмпирическим данным — этот термин был введён в работах Вапника и Червоненкиса.

Наиболее распространённым способом описания прецедентов является признаковое описание. Фиксируется совокупность n показателей, измеряемых у всех прецедентов. Если все n показателей числовые, то признаковые описания представляют собой числовые векторы размерности n. Возможны и более сложные случаи, когда прецеденты описываются временными рядами или сигналами, изображениями, видеорядами, текстами, попарными отношениями сходства или интенсивности взаимодействия, и т. д.

Для решения задачи обучения по прецедентам в первую очередь фиксируется модель восстанавливаемой зависимости. Затем вводится функционал качества, значение которого показывает, насколько хорошо модель описывает наблюдаемые данные. Алгоритм обучения (learning algorithm) ищет такой набор параметров модели, при котором функционал качества на заданной обучающей выборке принимает оптимальное значение. Процесс настройки (fitting) модели по выборке данных в большинстве случаев сводится к применению численных методов оптимизации.

Задача классификации

Классификация — один из разделов машинного обучения, посвященный решению следующей задачи. Имеется множество объектов (ситуаций), разделённых некоторым образом на классы. Задано конечное множество объектов, для которых известно, к каким классам они относятся. Это множество называется обучающей выборкой. Классовая принадлежность остальных объектов не известна. Требуется построить алгоритм, способный классифицировать произвольный объект из исходного множества.

Классифицировать объект — значит, указать номер (или наименование класса), к которому относится данный объект.

Классификация объекта — номер или наименование класса, выдаваемый алгоритмом классификации в результате его применения к данному конкретному объекту.

В математической статистике задачи классификации называются также задачами дискриминантного анализа.

В машинном обучении задача классификации относится к разделу обучения с учителем. Существует также обучение без учителя, когда разделение объектов обучающей выборки на классы не задаётся, и требуется классифицировать объекты только на основе их сходства друг с другом. В этом случае принято говорить о задачах кластеризации или таксономии, и классы называть, соответственно, кластерами или таксонами.

Алгоритм работы программы

Программа имеет пользовательский интерфейс, где пользователь может выбрать свой набор входных данных в формате csv, выбрать алгоритм классификации и тип графиков для анализа полученных результатов.

1.Dataset

В курсовой работе исследуется набор данных iris из репозитория машинного обучения UCI. Это самая известная база данных, которую можно найти в репозитории по распознаванию образов.

Свойства выбранного набора данных:

1. 150 экземпляров с 4 атрибутами (длина чашелистика, ширина чашелистика, длина лепестка, ширина лепестка).
2. Сбалансированное распределение классов (классы делятся на setosa, virginica, versicolor)
3. Нет недостающих данных

2.Определение цели:

Есть два вопроса, на которые нужно получить ответ в результате работы программы:

1.Прогноз – насколько точно модель может предсказать классы(виды) новых данных?

2.Заключение – какие алгоритмы могут эффективно помочь с предсказаниями?

3.Импорт библиотек и загрузка набора данных:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from pandas.plotting import parallel_coordinates
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,
QuadraticDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
import PySimpleGUI as sg
import re

pip install pandas/sklearn/seabotn/matplotlib/PySimpleGui/numpy
```

pandas – библиотека для чтения набора данных

sklearn – библиотека библиотека машинного обучения

numpy – библиотека для поддержки многомерных массивов; поддержки высокоуровневых математических функций, предназначенных для работы с многомерными массивами.

Seaborn – библиотека для визуализации данных.

PySimpleGUI – библиотека для реализации пользовательского интерфейса программы.

5. Разбиение данных на тренировочные и тестовые

```
train, test = train_test_split(data, test_size = 0.4, stratify = data['species'],
random_state = 42)
```

6.Анализ данных:

После разбиения набора данных, переходим к анализу и изучению обучающих данных с помощью инструментов matplotlib и seaborn.

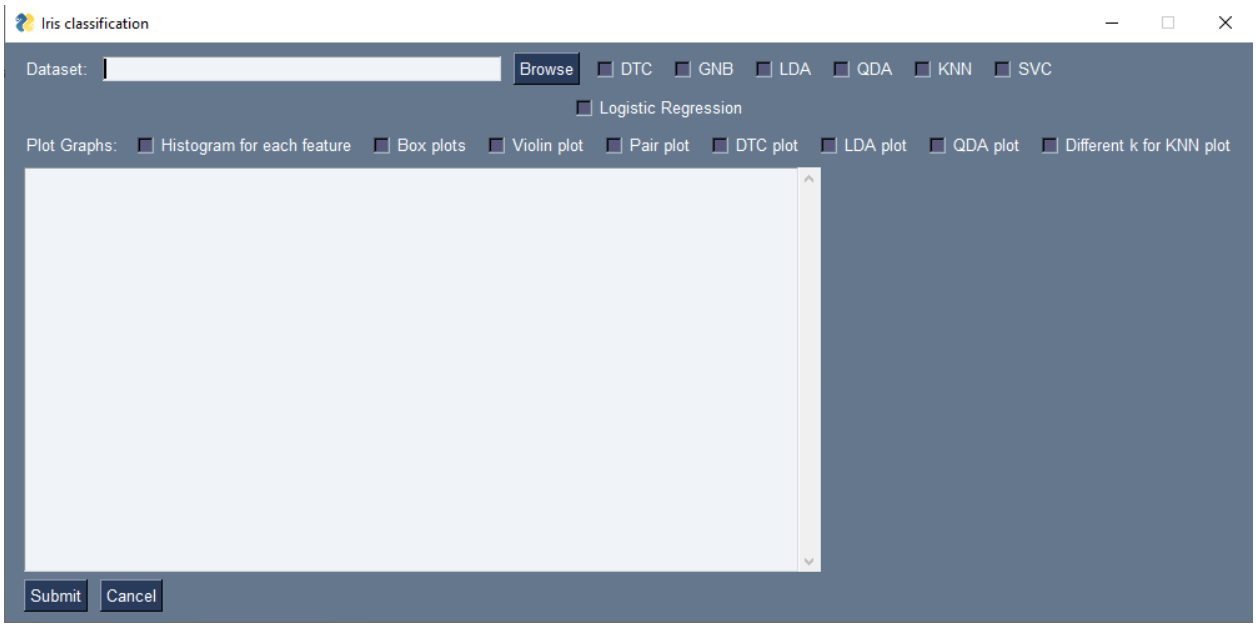
7.Построение классификаторов

- Classification Tree – дерево принятия решений
- Gaussian Naive Bayes Classifier - простой вероятностный классификатор, основанный на применении теоремы Байеса со строгими (наивными) предположениями о независимости.
- Linear Discriminant Analysis (LDA) - метод, используемый в статистике, распознавании образов и обучении машин для поиска линейной комбинации признаков, которая описывает или разделяет два или более классов или событий. Получившаяся комбинация может быть использована как линейный классификатор, или, более часто, для снижения размерности перед классификацией.
- Quadratic Discriminant Analysis (QDA) - обобщение метода LDA. QDA– многоклассный метод и он может использоваться для одновременной классификации нескольких классов
- K Nearest Neighbors (K-NN) - метрический алгоритм для автоматической классификации объектов или регрессии.
- Logistic regression - это статистическая модель, используемая для прогнозирования вероятности возникновения некоторого события путём его сравнения с логистической кривой. Эта регрессия выдаёт ответ в виде вероятности бинарного события (1 или 0).
- Support vector machine classifier(SVC) - набор схожих алгоритмов обучения с учителем, использующихся для задач классификации и регрессионного анализа. Принадлежит семейству линейных классификаторов. Особым свойством метода опорных векторов является непрерывное уменьшение эмпирической ошибки классификации и увеличение зазора, поэтому метод также известен как метод классификатора с максимальным зазором.

8. Анализ полученных результатов

Вывод программы

Окно программы:



Первые 5 элементов из набора данных:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

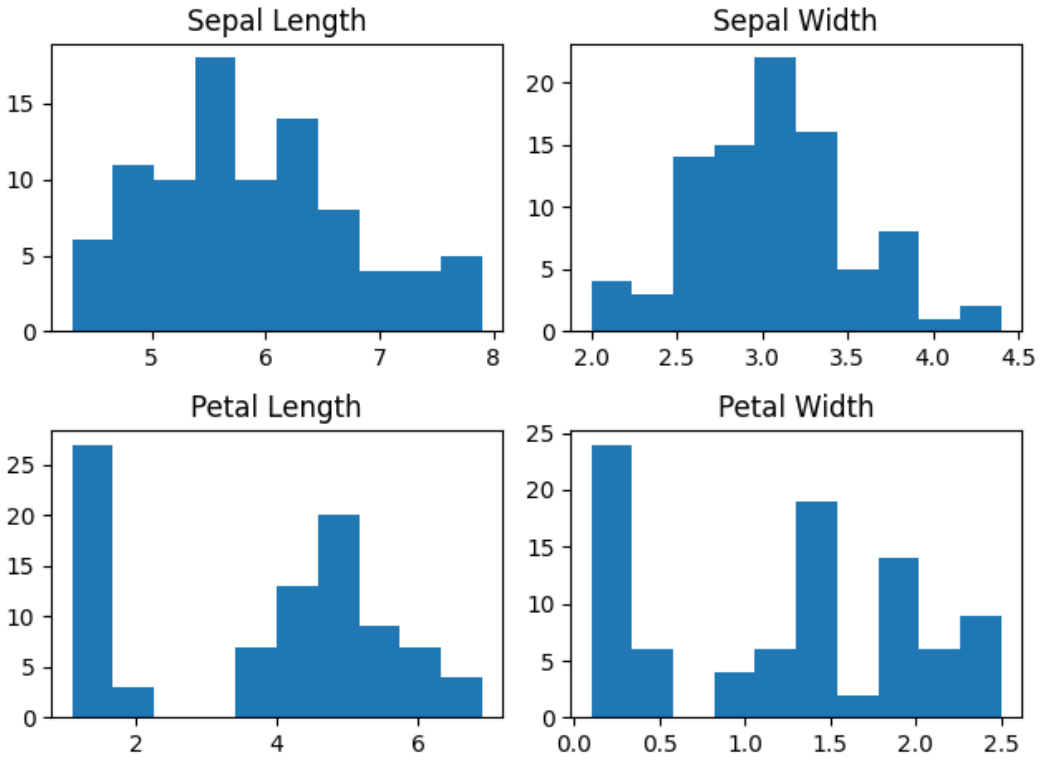
Вывод информации об атрибутах:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Проверка распределения классов:

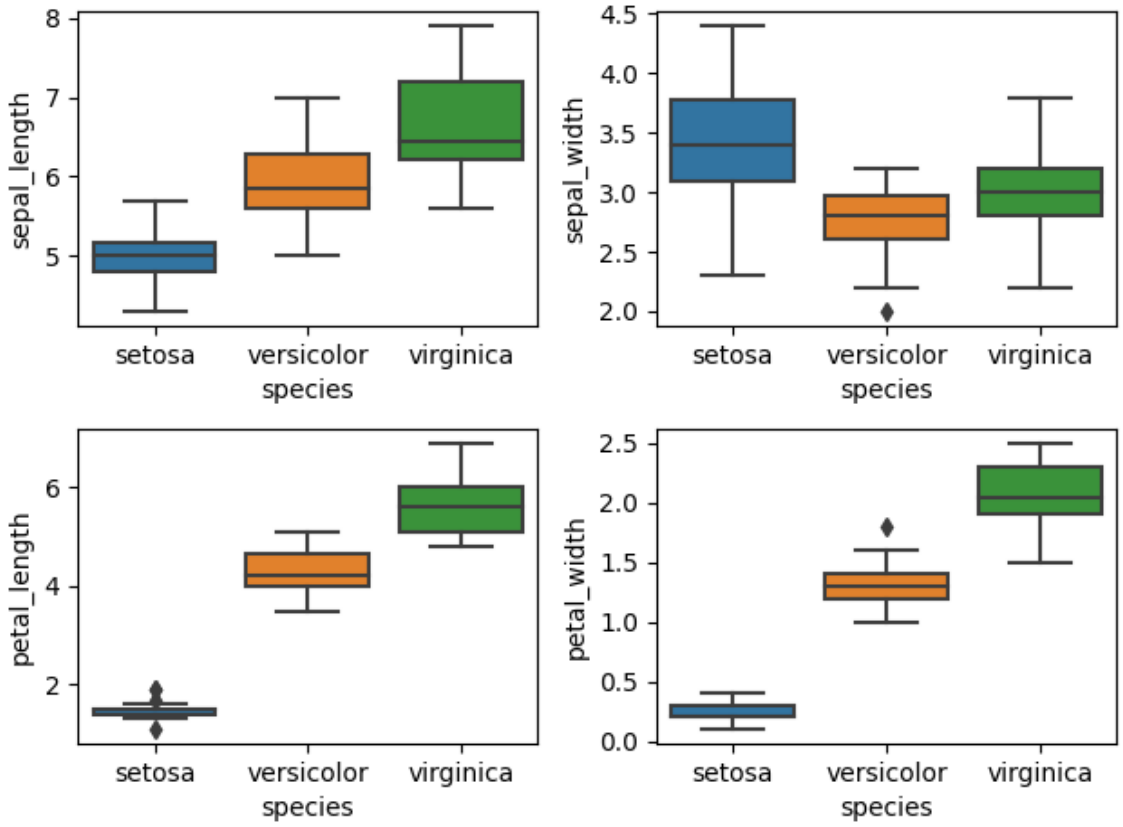
```
species
setosa       50
versicolor   50
virginica    50
dtype: int64
```

Вывод гистограммы для каждого атрибута:



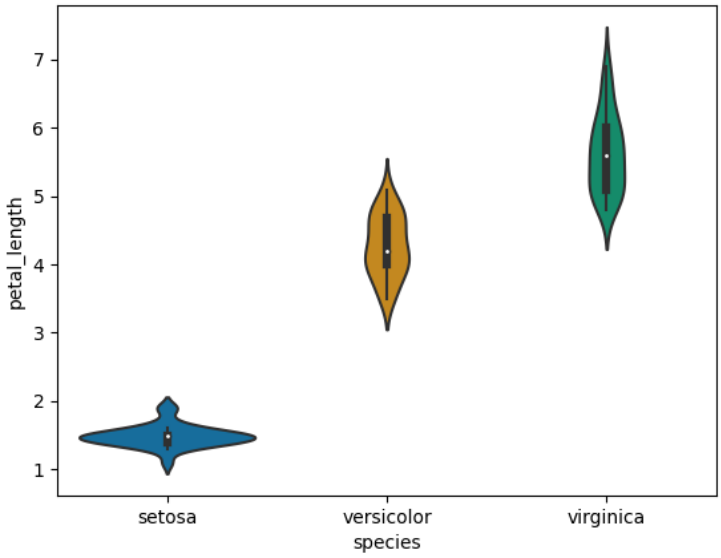
По данной диаграмме можно сделать вывод, что для petal_length и petal_width есть группа точек, которые имеют меньшие значения, чем другие. Соответственно можно предположить, что эти данные принадлежат разным классам.

Диаграмма размаха или side-by-side box plot:



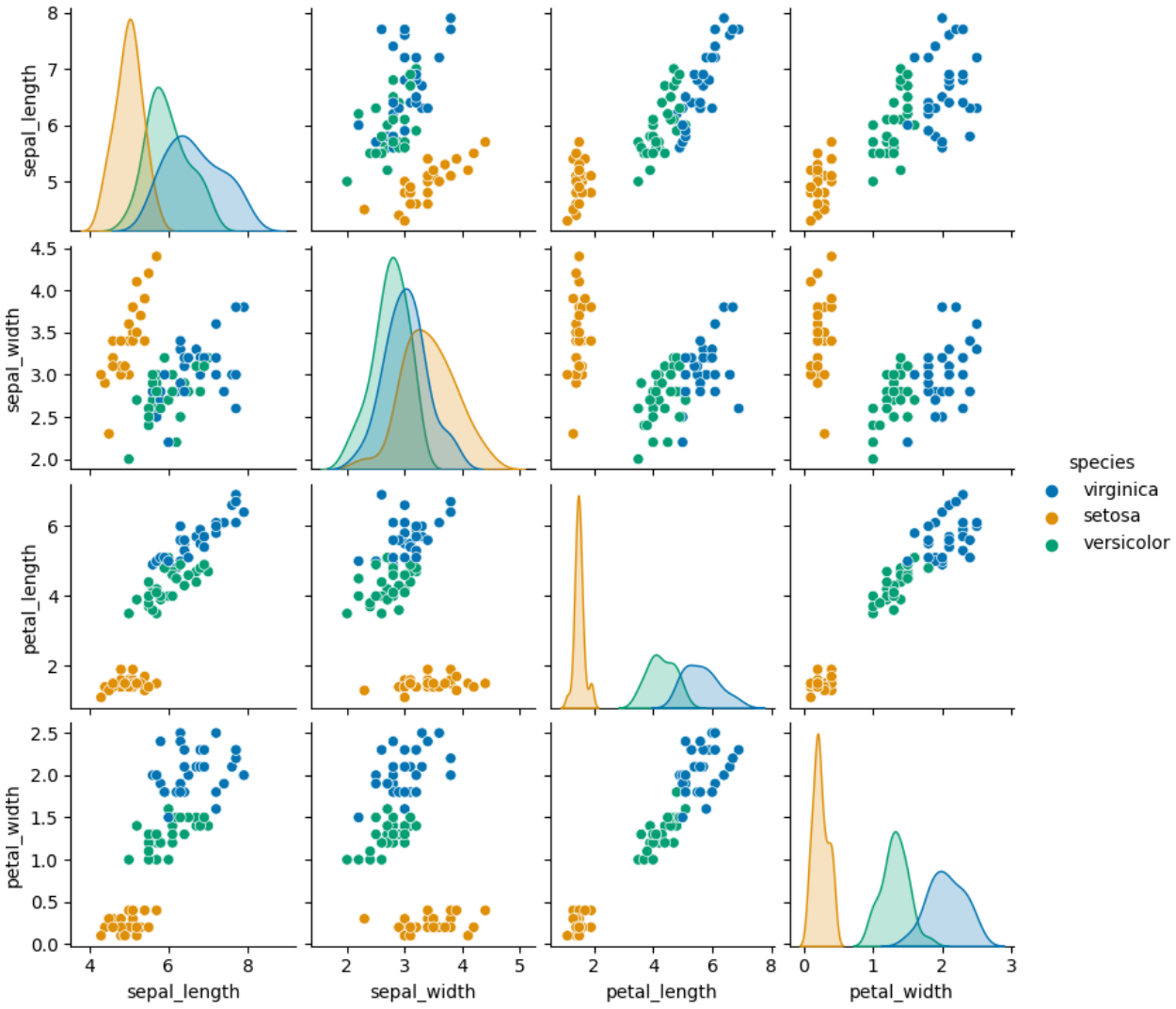
По двум нижним графикам можно сделать вывод, что группа точек данных, которую мы видели на гистограмме, относится к setosa. Размеры их лепестков меньше и у их размера меньше разброс по значениям, чем у двух других видов. Сравнивая два других вида, versicolor в среднем имеет более низкие значения, чем virginica.

Диаграмма Violin plot:



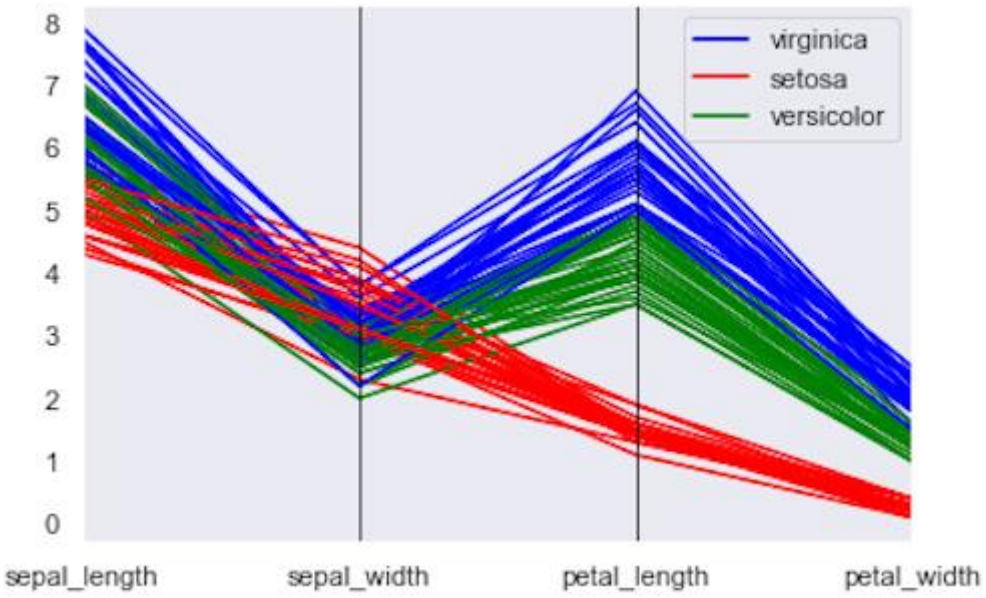
Эта диаграмма объединяет в себе преимущества двух предыдущих

Вывод диаграммы рассеяния всех парных атрибутов:



Можно заметить, что некоторые переменные сильно коррелированы, например petal_length и petal_width. Кроме того, размеры лепестков определяют разные классы лучше, чем размеры чашелистиков.

Вывод графика параллельных координат, на котором каждая строка представлена в виде линии:



Как мы видели ранее, размер лепестков может определять классы лучше, чем чашелистики.

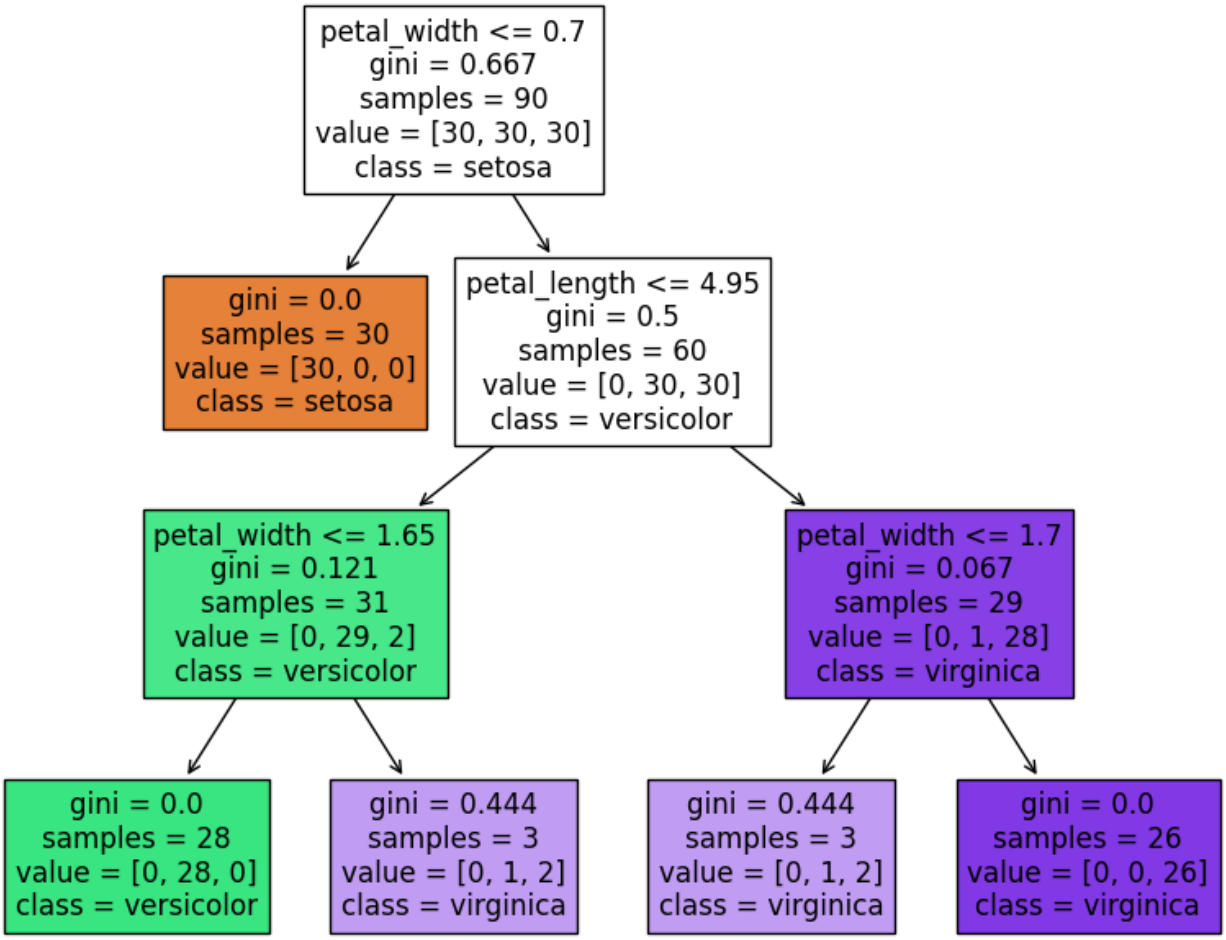
Точность алгоритмов:

1.Decision Tree:

```
The accuracy of the Decision Tree is 0.983.
Importance of each predictor [sepal length,sepal width, petal length, petal
width] :

[0.         0.         0.42430866 0.57569134]
```

Точность этого алгоритма 98.3%, также можно увидеть важность каждого предиктора. Из выходных данных мы знаем, что первые два атрибута (размеры чашелистиков) не имеют значения, и только свойства лепестков используются для построения дерева.



На этом графике представлены правила классификации дерева решений на основе тестовых данных.

Помимо каждого правила (например, первый критерий - petal_width ≤ 0,7), мы также можем видеть индекс Джини для каждого разбиения, назначенный класс и т.д. Во всех конечных узлах индекс джини равен нулю, за исключением двух светло-пурпурных, следовательно, мы можем менее уверенно относиться к случаям в этих двух категориях.

2.Gaussian Naive Bayes Classifier:

```
The accuracy of the Guassian Naive Bayes Classifier on test data is 0.933
```

Точность этого алгоритма по четырём предикторам составляет 93.3%

```
The accuracy of the Guassian Naive Bayes Classifier with 2 predictors on test
data is 0.950
```

Использование только двух предикторов приводит к более правильной классификации точек. Точность составляет 95%.

3. Linear Discriminant Analysis (LDA)

```
The accuracy of the LDA Classifier on test data is 0.983
The accuracy of the LDA Classifier with two predictors on test data is 0.933
```

Использование всех четырёх предикторов увеличивает точность LDA.

Чтобы визуализировать границу принятия решения в 2D, мы можем использовать нашу модель LDA только с лепестками, а также построить тестовые данные:

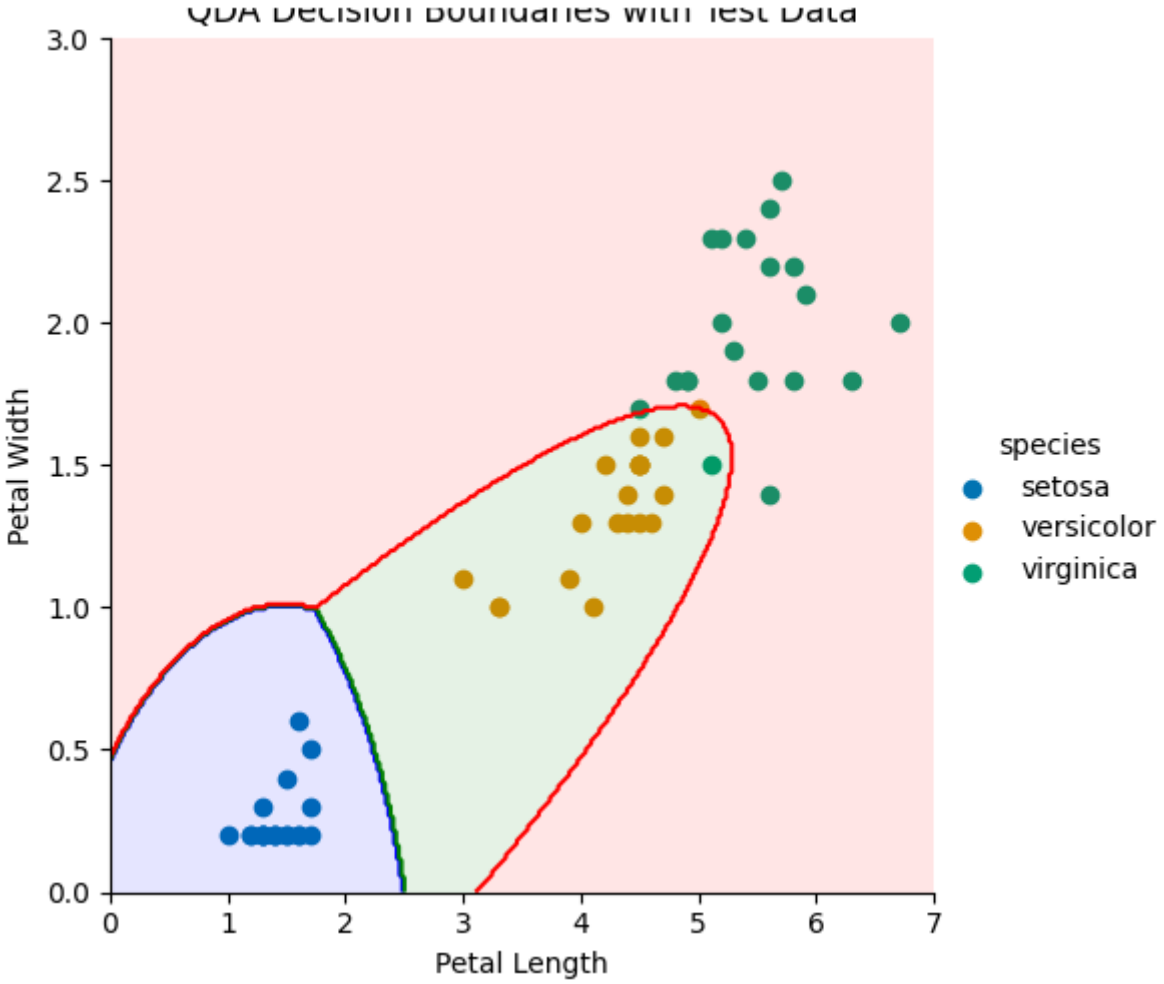
Четыре контрольных точки неправильно классифицированы - три virginica и одна versicolor.

4. Quadratic Discriminant Analysis (QDA)

```
The accuracy of the QDA Classifier is 0.983
The accuracy of the QDA Classifier with two predictors is 0.967
```

QDA имеет ту же точность при использовании всех предикторов, что и LDA, но имеет лучшую точность, когда используются только два.

Аналогичным образом строится граница принятия решений для QDA:

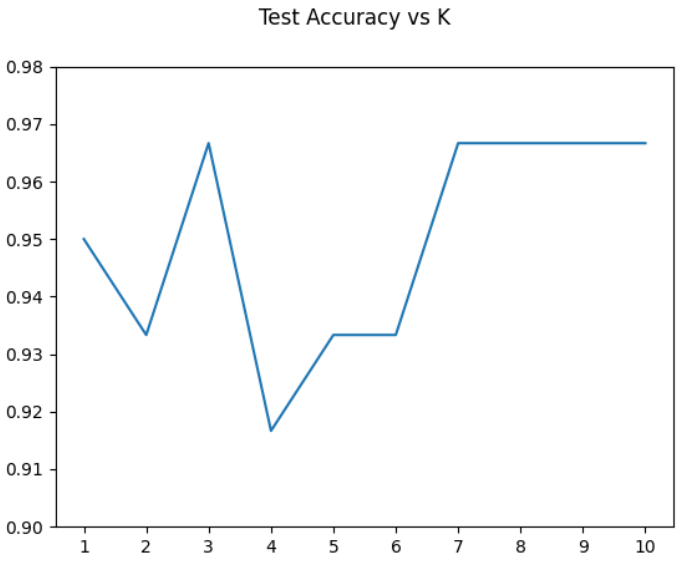


Здесь граница представляет собой квадратичную функцию, в отличие от LDA.

5. K Nearest Neighbors (K-NN)

```
The accuracy of the KNN Classifier is 0.933
```

График зависимости точности от разных значений K:



Можно заметить, что наилучшая точность достигается при K=3 или K от 7 до 10. Support vector machine classifier(SVC)

```
The accuracy of the linear SVC is 1.000

The accuracy of the Poly SVC is 0.933
```

11. Logistic Regression

```
The accuracy of the Logistic Regression is 0.950
```

Приложение

Ссылки: https://archive.ics.uci.edu/ml/datasets/iris

Код программы:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from pandas.plotting import parallel_coordinates
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,
QuadraticDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
import PySimpleGUI as sg
import re

layout = [
    [sg.Text('Dataset:'), sg.InputText(), sg.FileBrowse(),
sg.Checkbox('DTC'), sg.Checkbox('GNB'), sg.Checkbox('LDA'),
sg.Checkbox('QDA'), sg.Checkbox('KNN'), sg.Checkbox('SVC')],
    [sg.Text('\t\t\t\t\t\t\t\t        '), sg.Checkbox('Logistic Regression')],
    [sg.Text('Plot Graphs:'), sg.Checkbox('Histogram for each feature'),
sg.Checkbox('Box plots'),sg.Checkbox('Violin plot'),sg.Checkbox('Pair
plot'),sg.Checkbox('DTC plot'),sg.Checkbox('LDA plot'),sg.Checkbox('QDA
plot'),sg.Checkbox('Different k for KNN plot')],
    [sg.Output(size=(88, 20))],
    [sg.Submit(), sg.Cancel()]
]
window = sg.Window('Iris classification', layout)
while True:
    event, values = window.read()
    if event in (None, 'Exit', 'Cancel'):
        break
    if event == 'Submit':
        file1 =  isitago = None
        print(values[0])
        if values[0]:
            file1 = re.findall('.+:\/.+\.+.', values[0])
            isitago = 1
```

```
            if not file1 and file1 is not None:
                print('Error: File path not valid.')
                isitago = 0
            k=0;
            for i in values:
                if values[i]==True : k=1
            if k == 0 : print("Choose smth")
            elif isitago == 1:
                print('Info: Filepath correctly defined.')
                filepaths = [] #files
                filepaths.append(values[0])
                data = pd.read_csv(values[0])
                print(data.head(5))
                print(data.describe())
                print(data.groupby('species').size())


                train, test = train_test_split(data, test_size = 0.4,
stratify = data['species'], random_state = 42)
                X_train =
train[['sepal_length','sepal_width','petal_length','petal_width']]
                y_train = train.species
                X_test =
test[['sepal_length','sepal_width','petal_length','petal_width']]
                y_test = test.species
                y_train_en =
y_train.replace({'setosa':0,'versicolor':1,'virginica':2}).copy()

                n_bins = 10
                print("\n\n")
                selected_predictors = ["petal_length", "petal_width"]
                algos = [] #algos to compare
                if values[1] == True: algos.append('DTC')
                if values[2] == True: algos.append('GNB')
                if values[3] == True: algos.append('LDA')
                if values[4] == True: algos.append('QDA')
                if values[5] == True: algos.append('KNN')
                if values[6] == True: algos.append('SVC')
                if values[7] == True: algos.append('Logistic Regression')
                if values[8] == True: algos.append('Histogram for each
feature')
                if values[9] == True: algos.append('Box plots')
                if values[10] == True: algos.append('Violin plot')
                if values[11] == True: algos.append('Pair plot')
                if values[12] == True: algos.append('DTC plot')
                if values[13] == True: algos.append('LDA plot')
                if values[14] == True: algos.append('QDA plot')
                if values[15] == True: algos.append('Different k for KNN
plot')
                fn = ["sepal_length", "sepal_width", "petal_length",
"petal_width"] #for plots
                cn = ['setosa', 'versicolor', 'virginica']
                DTC = 0
                LDA = 0
                for algo in algos:
                    if algo == 'DTC':
                        print("DTC Classification:\n")
                        DTC = 1
                        mod_dt = DecisionTreeClassifier(max_depth = 3,
random_state = 1)
                        mod_dt.fit(X_train,y_train)
                        prediction=mod_dt.predict(X_test)
                        print('The accuracy of the Decision Tree
is',"{:.3f}".format(metrics.accuracy_score(prediction,y_test)))
```

```
                        print("Importance of each predictor [sepal
length,sepal width, petal length, petal width] :")
                        print(mod_dt.feature_importances_)
                        print("\n\n");
                    if algo == 'GNB':
                        # Guassian Naive Bayes Classifier
                        mod_gnb_all = GaussianNB()
                        GNB = 1
                        y_pred = mod_gnb_all.fit(X_train,
y_train).predict(X_test)
                        print('The accuracy of the Guassian Naive Bayes
Classifier on test data
is',"{:.3f}".format(metrics.accuracy_score(y_pred,y_test)))

                        print("\n");
                        # Guassian Naive Bayes Classifier with two predictors
                        mod_gnb = GaussianNB()
                        y_pred = mod_gnb.fit(X_train[selected_predictors],
y_train).predict(X_test[selected_predictors])
                        print('The accuracy of the Guassian Naive Bayes
Classifier with 2 predictors on test data
is',"{:.3f}".format(metrics.accuracy_score(y_pred,y_test)))
                        print("\n\n");
                    if algo == 'LDA':
                        # LDA Classifier
                        LDA = 1
                        mod_lda_all = LinearDiscriminantAnalysis()
                        y_pred = mod_lda_all.fit(X_train,
y_train).predict(X_test)
                        print('The accuracy of the LDA Classifier on test
data is',"{:.3f}".format(metrics.accuracy_score(y_pred,y_test)))
                        print("\n");
                        # LDA Classifier with two predictors
                        mod_lda = LinearDiscriminantAnalysis()
                        y_pred = mod_lda.fit(X_train[selected_predictors],
y_train).predict(X_test[selected_predictors])
                        print('The accuracy of the LDA Classifier with two
predictors on test data
is',"{:.3f}".format(metrics.accuracy_score(y_pred,y_test)))

                        print("\n\n");
                    if algo == 'QDA':
                        # QDA Classifier
                        QDA = 1
                        mod_qda_all = QuadraticDiscriminantAnalysis()
                        y_pred = mod_qda_all.fit(X_train,
y_train).predict(X_test)
                        print('The accuracy of the QDA Classifier
is',"{:.3f}".format(metrics.accuracy_score(y_pred,y_test)))

                        print("\n");

                        # QDA Classifier with two predictors
                        mod_qda = QuadraticDiscriminantAnalysis()
                        y_pred = mod_qda.fit(X_train[selected_predictors],
y_train).predict(X_test[selected_predictors])
                        print('The accuracy of the QDA Classifier with two
predictors is',"{:.3f}".format(metrics.accuracy_score(y_pred,y_test)))

                        print("\n\n");
                    if algo == 'KNN':
                        # KNN, first try 5
                        KNN = 1
                        mod_5nn=KNeighborsClassifier(n_neighbors=5)
```

```
                        mod_5nn.fit(X_train,y_train)
                        prediction=mod_5nn.predict(X_test)
                        print('The accuracy of the 5NN Classifier
is',"{:.3f}".format(metrics.accuracy_score(prediction,y_test)))

                        print("\n\n");
                    if algo == 'SVC':
                        # SVC with linear kernel
                        # for SVC, may be impractical beyond tens of
thousands of samples
                        SVC1 = 1
                        linear_svc = SVC(kernel='linear').fit(X_train,
y_train)
                        prediction=linear_svc.predict(X_test)
                        print('The accuracy of the linear SVC
is',"{:.3f}".format(metrics.accuracy_score(prediction,y_test)))
                        print("\n");

                        # SVC with polynomial kernel
                        poly_svc = SVC(kernel='poly', degree =
4).fit(X_train, y_train)
                        prediction=poly_svc.predict(X_test)
                        print('The accuracy of the Poly SVC
is',"{:.3f}".format(metrics.accuracy_score(prediction,y_test)))
                        print("\n\n");
                    if algo == 'Logistic Regression':
                        LR = 1
                        mod_lr = LogisticRegression(solver = 'newton-
cg').fit(X_train, y_train)
                        prediction=mod_lr.predict(X_test)
                        print('The accuracy of the Logistic Regression
is',"{:.3f}".format(metrics.accuracy_score(prediction,y_test)))
                        print("\n\n");
                    if algo == 'Histogram for each feature':
                        fig, axs = plt.subplots(2, 2)
                        axs[0,0].hist(train['sepal_length'], bins = n_bins);
                        axs[0,0].set_title('Sepal Length');
                        axs[0,1].hist(train['sepal_width'], bins = n_bins);
                        axs[0,1].set_title('Sepal Width');
                        axs[1,0].hist(train['petal_length'], bins = n_bins);
                        axs[1,0].set_title('Petal Length');
                        axs[1,1].hist(train['petal_width'], bins = n_bins);
                        axs[1,1].set_title('Petal Width');
                        fig.tight_layout(pad=1.0);

                    if algo == 'Box plots':

                        fig, axs = plt.subplots(2, 2)
                        sns.boxplot(x = 'species', y = 'sepal_length', data =
train, order = cn, ax = axs[0,0])
                        sns.boxplot(x = 'species', y = 'sepal_width', data =
train, order = cn, ax = axs[0,1])
                        sns.boxplot(x = 'species', y = 'petal_length', data =
train, order = cn, ax = axs[1,0])
                        sns.boxplot(x = 'species', y = 'petal_width', data =
train,  order = cn, ax = axs[1,1])
                        fig.tight_layout(pad=1.0)
                    if algo == 'Violin plot':
                        plt.figure()
                        sns.violinplot(x="species", y="petal_length",
data=train, size=5, order = cn, palette = 'colorblind')
                    if algo == 'Pair plot':
                        sns.pairplot(train, hue="species", height = 2,
palette = 'colorblind')
```

```
                    if algo == 'DTC plot':
                        if DTC == 0:
                           print("Choose DTC Algorythm for DTC Plot!!!")
                           break
                        plt.figure(figsize = (10,8))
                        plot_tree(mod_dt, feature_names = fn, class_names =
cn, filled = True)
                    if algo == 'LDA plot':
                        mod_lda_1 = LinearDiscriminantAnalysis()
                        y_pred = mod_lda_1.fit(X_train[selected_predictors],
y_train_en).predict(X_test[selected_predictors])
                        N = 300
                        X = np.linspace(0, 7, N)
                        Y = np.linspace(0, 3, N)
                        X, Y = np.meshgrid(X, Y)
                        g = sns.FacetGrid(test, hue="species", height=5,
palette = 'colorblind').map(plt.scatter,"petal_length", "petal_width",
).add_legend()
                        my_ax = g.ax
                        zz = np.array([mod_lda_1.predict(np.array([[xx,yy]]))
for xx, yy in zip(np.ravel(X), np.ravel(Y)) ] )
                        Z = zz.reshape(X.shape)
                        #Plot the filled and boundary contours
                        my_ax.contourf( X, Y, Z, 2, alpha = .1, colors =
('blue','green','red'))
                        my_ax.contour( X, Y, Z, 2, alpha = 1, colors =
('blue','green','red'))
                        # Add axis and title
                        my_ax.set_xlabel('Petal Length')
                        my_ax.set_ylabel('Petal Width')
                        my_ax.set_title('LDA Decision Boundaries with Test
Data');
                    if algo == 'QDA plot':
                        mod_qda_1 = QuadraticDiscriminantAnalysis()
                        y_pred = mod_qda_1.fit(X_train.iloc[:,2:4],
y_train_en).predict(X_test.iloc[:,2:4])

                        N = 300
                        X = np.linspace(0, 7, N)
                        Y = np.linspace(0, 3, N)
                        X, Y = np.meshgrid(X, Y)

                        g = sns.FacetGrid(test, hue="species", height=5,
palette = 'colorblind').map(plt.scatter,"petal_length", "petal_width",
).add_legend()
                        my_ax = g.ax

                        zz = np.array([mod_qda_1.predict(np.array([[xx,yy]]))
for xx, yy in zip(np.ravel(X), np.ravel(Y)) ] )
                        Z = zz.reshape(X.shape)

                        #Plot the filled and boundary contours
                        my_ax.contourf( X, Y, Z, 2, alpha = .1, colors =
('blue','green','red'))
                        my_ax.contour( X, Y, Z, 2, alpha = 1, colors =
('blue','green','red'))

                        # Addd axis and title
                        my_ax.set_xlabel('Petal Length')
                        my_ax.set_ylabel('Petal Width')
                        my_ax.set_title('QDA Decision Boundaries with Test
Data');
```

```
                        plt.figure()
                    if algo == 'Different k for KNN plot':
                        acc_s = pd.Series(dtype = 'float')
                        for i in list(range(1,11)):
                            mod_knn=KNeighborsClassifier(n_neighbors=i)
                            mod_knn.fit(X_train,y_train)
                            prediction=mod_knn.predict(X_test)
                            acc_s =
acc_s.append(pd.Series(metrics.accuracy_score(prediction,y_test)))

                        plt.plot(list(range(1,11)), acc_s)
                        plt.suptitle("Test Accuracy vs K")
                        plt.xticks(list(range(1,11)))
                        plt.ylim(0.9,0.98)
                plt.show()
        else:
            print('Please choose dataset.')




# try different k (KNN)

window.close()
```