Knowledge Databases Labs

5th Year

Lab-6- Flower Type Prediction with Classification Algorithms

Introduced By:

Eng. Khaled Kondakji *Eng. Heba Tadmouri*

Eng. Manar AL-Marai

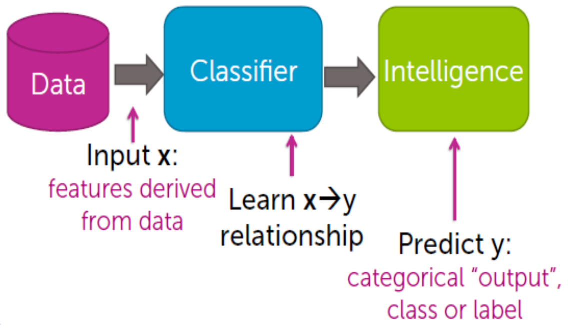
Supervised By:

Dr. Eng. Asmaa Shaar

Flower Type Prediction with Classification Algorithms

Introduction to classification:

- Given a collection of data tuples (training set). Each tuple is characterized by (x, y):
  - ✓ x (input) is the *attribute set*.
  - ✓ y (output) is the *class label attribute*.
- Task: based on the input dataset, learn a *classification model* (*Classifier*) that describes a predetermined set of data classes.



- Some common algorithms we're going to look at (decision tree, Naïve Bayes and SVM).

Decision Tree Classifier:

- A decision tree is a flowchart-like tree structure where:
  - ✓ Each internal node denotes a *test on an attribute*.
  - ✓ Each branch represents an *outcome of the test*.
  - ✓ And each leaf node holds a *class label*, the topmost node in the tree is the *root node*.
- Given a tuple, X, for which the class label is *unknown*,
  - ✓ The *attribute values* of the tuple X are *tested* against the decision tree.
  - ✓ Therefore, a path is traced from the root to a leaf node which holds the class prediction for that tuple.
- To build up DT we apply the following steps:
  - ✓ A greedy algorithm, where the tree is constructed in a *top-down recursive approach*.
  - ✓ At the start, *all* the training tuples are at the *root node*. Then training set is *recursively* partitioned into smaller subsets as the tree is being built.
  - ✓ training tuples are *partitioned* based on *test attributes* (which are selected based on a *heuristic or statistical measure*), like: (*Information Gain* or *Gain Ratio*).

Naïve Bayes Classifier:

- Bayesian Classifiers are statistical classifier, performs probabilistic prediction and are based on Bayes' theorem. They have also exhibited high accuracy and speed when applied to large databases.
- For classification problems, we want to determine $P(H|\mathbf{X})$. In other words, we are looking for the probability that tuple **X** belongs to class C, given that we know the attribute description of **X**.

Support Vector Machine (SVM) Classifier:

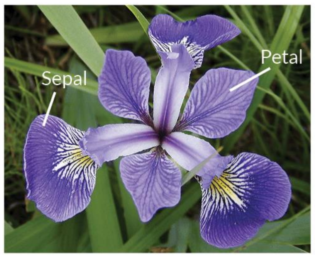
- A linear SVM searches for a hyperplane that maximizes the margin (maximal margin classifier).
- In `sklearn` we use `SVC` class which takes `kernel` as attribute which specifies the SVM kernel like (linear, polynomial, ….).

Flowers' type prediction:

We're going to use Iris dataset that contains information about pental and sepal length and width of different flowers. Our dataset contains the following attributes:

#	Attribute	Explanation
1	Sepal_length	sepal length in cm.
2	Sepal_width	sepal width in cm.
3	Pental_length	petal length in cm.
4	Pental_width	petal width in cm.
5	**species (class label attribute)**	**Type of flower (Setosa, Versicolor, Virginica).**

Each flower can be one of the following:



Iris Versicolor

Iris Setosa

Iris Virginica

1st step: Import required libraries and load our dataset:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
import graphviz
from mlxtend.plotting import plot_decision_regions
from sklearn.metrics import accuracy_score
#load dataset:
iris_df = pd.read_csv('IrisDataset.csv')
iris_df.head()
```

Our dataset looks like the following:

	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

2nd step: Exploratory Data Analysis (EDA):

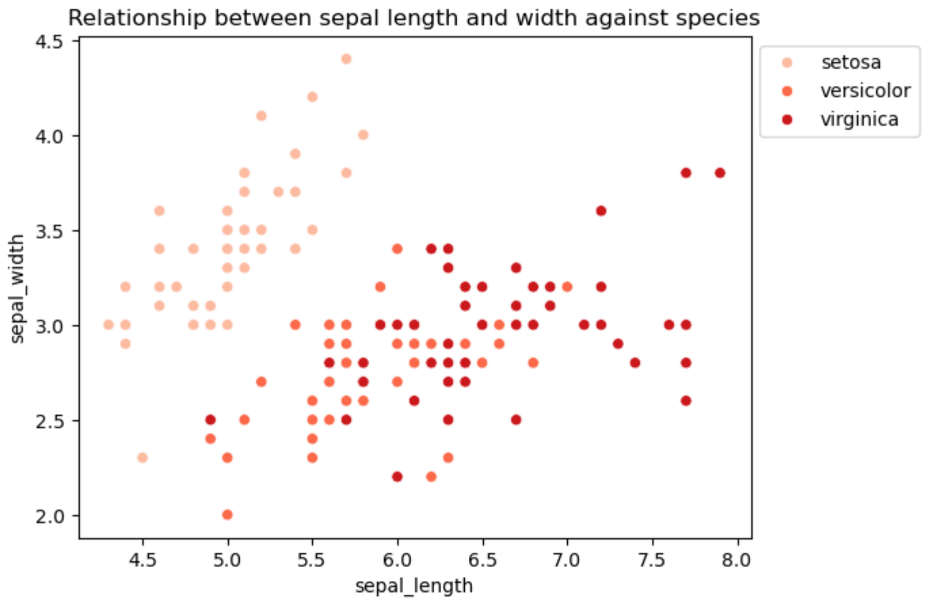
First at all we're going to explore our `species` attribute using `value_counts()` method:

```
iris_df['species'].value_counts()
```

```
species
setosa        50
versicolor    50
virginica     50
Name: count, dtype: int64
```

We can extract from here that our classification problem is balanced, meaning that we have quite a similar number of examples in both classes.
Now let's explore relationship between `sepal` properties and `species` attribute. For that we're going to use `seaborn` library:

```
sns.scatterplot(x='sepal_length', y='sepal_width', hue='species',
data=iris_df, palette='Reds')
# Placing Legend outside the Figure
plt.legend(bbox_to_anchor=(1, 1), loc=2)
plt.title('Relationship between sepal length and width against
species')
```

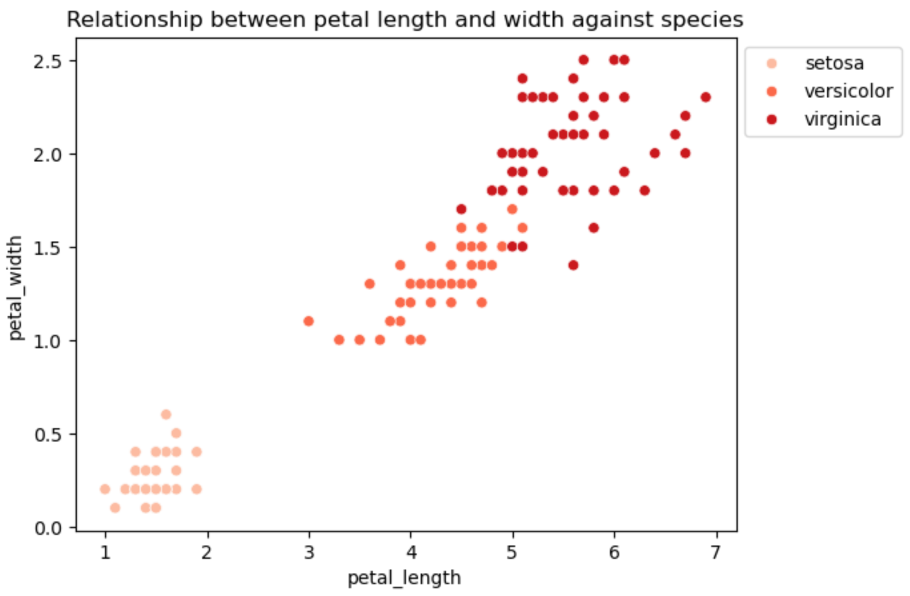


From the above plot, we can infer that:

- `Species Setosa` has smaller `sepal lengths` but larger `sepal widths`.
- `Versicolor Species` lies in the middle of the other two `species` in terms of `sepal length` and `width`.
- `Species Virginica` has larger `sepal lengths` but smaller `sepal widths`.

In the same way we're going to draw the relationship between `petal` properties and `species` attribute

```
sns.scatterplot(x='petal_length', y='petal_width', hue='species', data=iris_df,
palette='Reds')
# Placing Legend outside the Figure
plt.legend(bbox_to_anchor=(1, 1), loc=2)
plt.title('Relationship between petal length and width against species')
```

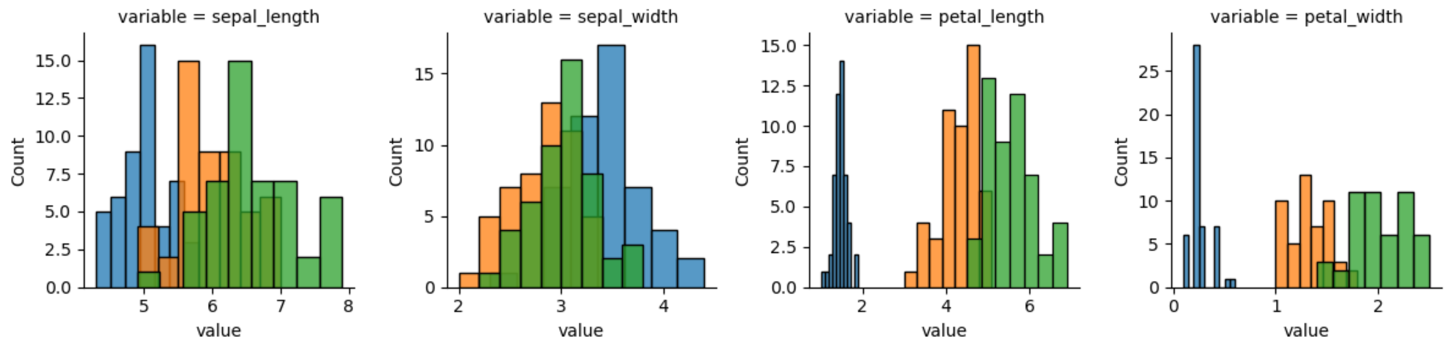


From the above plot, we can infer that:

- Species Setosa has smaller petal lengths and widths.
- Versicolor Species lies in the middle of the other two species in terms of petal length and width.
- Species Virginica has the largest of petal lengths and widths.

Another way to get insights about data attributes is by using histplot() which is used basically for the univariant set of observations and visualizes it through a histogram i.e. only one observation and hence we choose one particular column of the dataset.

```
g = sns.FacetGrid(iris_df.melt(id_vars='species'), col="variable", sharex=False,
sharey=False, hue="species")
g.map(sns.histplot,"value")
```

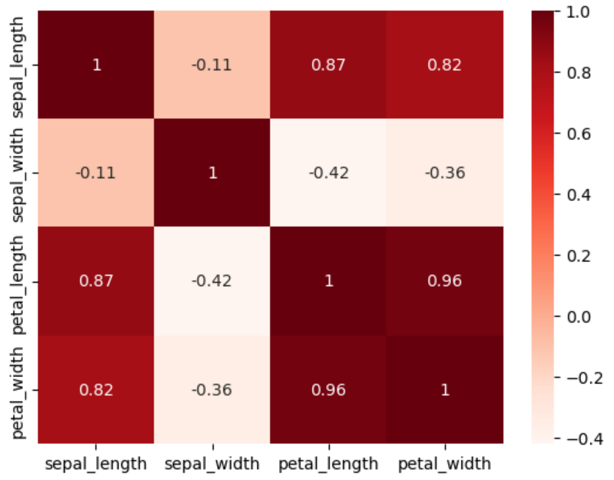


Refer to this link for information. From the above plots, we can see that:

- In the case of Sepal Length, there is a huge amount of overlapping.
- In the case of Sepal Width also, there is a huge amount of overlapping.
- In the case of Petal Length, there is a very little amount of overlapping.
- In the case of Petal Width also, there is a very little amount of overlapping.

Finally, we're going to print the correlation matrix between attributes:

```
sns.heatmap(iris_df.corr(numeric_only=True),annot = True, cmap='Reds');
plt.show()
```



3rd step: Prepare our dataset for classification task:

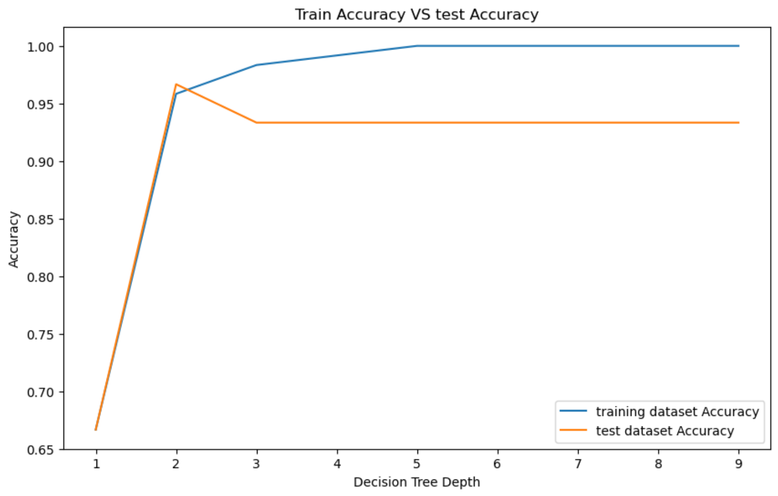
```
# shuffle our data:
iris_df = iris_df.sample(frac=1) #frac = 1 mean returns all rows.

# extract input and output label:
X = iris_df.drop(labels=["species"],axis = 1)
y = iris_df['species']

#setting a seed and split our data into train and test sets:
np.random.seed(42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

4th step: build and tune our classification models:

Build and evaluate Decision Tree Model:

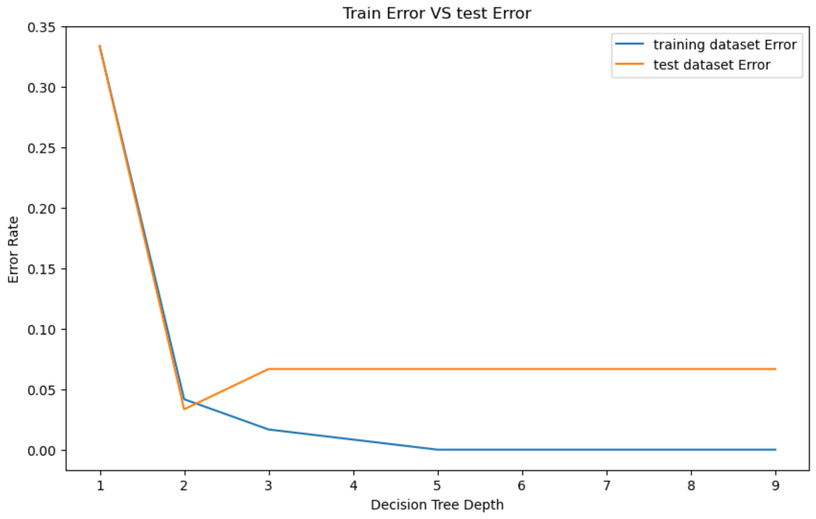
We're going to use `DecisionTreeClassifier` class provided by `sklearn` in order to learn a model based on the input data. Our class takes some important parameters such as criterion and `max_depth`.

```
train_acc = [] #training set accuracy list
test_acc = [] #test set accuracy list
train_error = [] #training set error list
test_error= [] #test set error list
depths = list(range(1,10))
for i in range (1,10):
    dt = DecisionTreeClassifier(criterion='entropy',max_depth=i)
    dt.fit(X_train,y_train)
    train_acc.append(accuracy_score(y_train,dt.predict(X_train)))
    test_acc.append(accuracy_score(y_test,dt.predict(X_test)))
    train_error.append(1-(accuracy_score(y_train,dt.predict(X_train))))
    test_error.append(1-(accuracy_score(y_test,dt.predict(X_test))))
plt.figure(figsize=(10,6))
plt.plot(depths, train_acc,'-',depths,test_acc,'-')
plt.title("Train Accuracy VS test Accuracy")
plt.xlabel("Decision Tree Depth")
plt.ylabel("Accuracy")
plt.legend(["training dataset Accuracy", "test dataset Accuracy"])
```



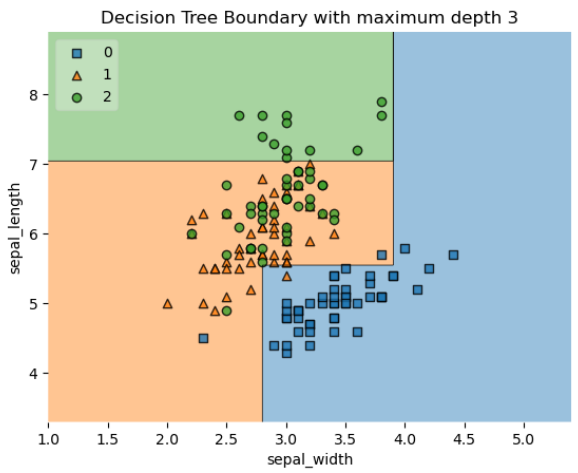
- We can notice that if we use the *training set* to measure the classifier's accuracy, this estimate would likely be *optimistic* because the classifier tends to *overfit the data*.
- Because of that, we tend to estimate classifier accuracy using a list of tuples called *test dataset* which is *independent* of the training dataset.
- The *accuracy* of a classifier, M, on a given test set is the "percentage of test set tuples that are *correctly* classified by the classifier".
- We can evaluate the same concepts based on classification error which is given by:
  - `Error = 1- Accuracy.`

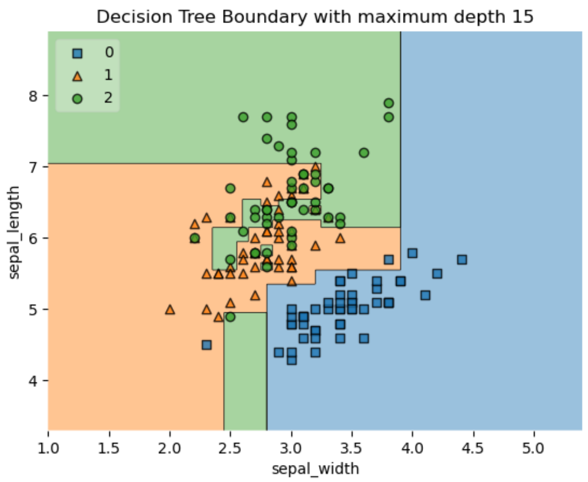
```
plt.figure(figsize=(10,6))
plt.plot(depths, train_error,'-',depths,test_error,'-')
plt.title("Train Error VS test Error")
plt.xlabel("Decision Tree Depth")
plt.ylabel("Error Rate")
plt.legend(["training dataset Error", "test dataset Error"])
```



- To select the best `max_depth` value for a decision tree, we should notice the effect of getting higher depth trees VS lower depth trees.
- Now, we're going to build a new `DecisionTreeClassifier` based only on two attributes (`sepal width`, `sepal length`), in order to notice the `max_depth` parameter effect for 2 values (3 and 15).

```
x1 = iris_df[["sepal_width","sepal_length"]].values
y = iris_df.species.replace({'setosa':0,'versicolor':1,'virginica':2}).values
x1_train, x1_test, y1_train, y1_test = train_test_split(x1,y,test_size= .2)
for i in [3,15]:
    dt1 = DecisionTreeClassifier(criterion='entropy',max_depth=i)
    dt1.fit(x1_train,y1_train)
    plot_decision_regions(np.array(x1, ), np.array(y), clf=dt1, legend=2,)
    plt.xlabel('sepal_width')
    plt.ylabel('sepal_length')
    title = 'Decision Tree Boundary with maximum depth '+ str(i)
    plt.title(title)
    plt.show()
```

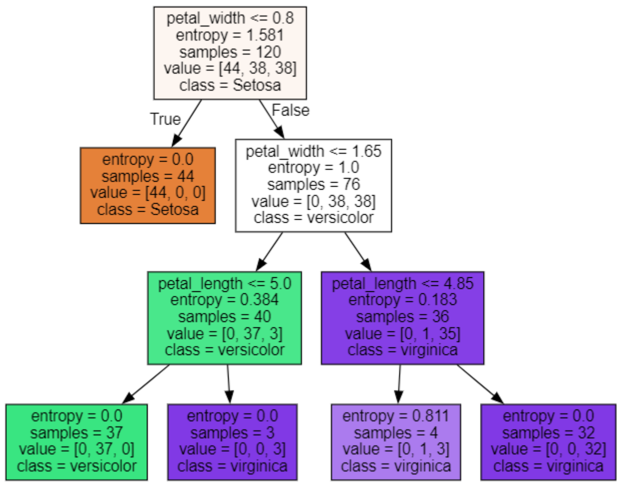




When a decision tree is built, many of the branches will reflect anomalies in the training data due to *noise* or *outliers*. Pruned trees:

- ✓ Tend to be smaller and *less complex*, and thus *easier* to comprehend. Usually faster and *better* at *classifying test tuples* than unpruned trees. This code will draw the fitted tree:

```
dt = DecisionTreeClassifier(criterion='entropy', max_depth=3)
dt.fit(X_train,y_train)
dot_data=tree.export_graphviz(dt,out_file=None,feature_names=list(X.column
s),class_names=['Setosa', 'versicolor', 'virginica'], filled = True)
tr = graphviz.Source(dot_data, format ="png")
tr
```



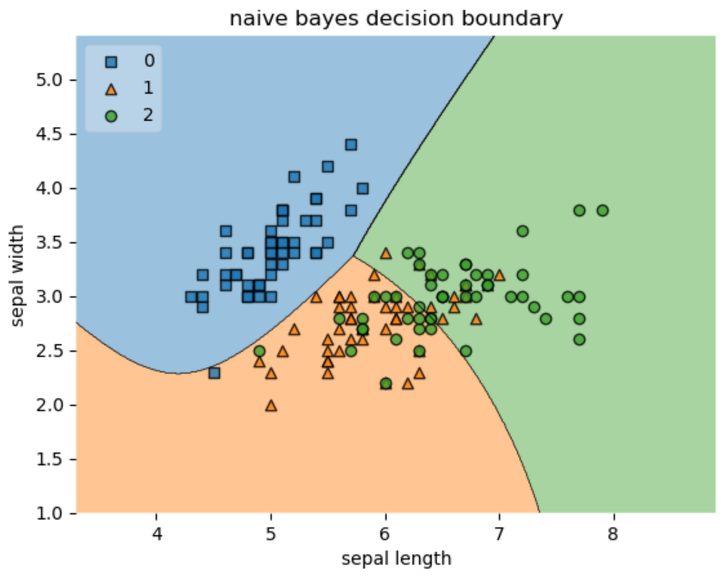
Build model using naïve bayes model:

`SKLearn` provides many classes that implements naïve bayes classifier like `GaussianNB`, `MultinominalNB`, … which can be used for different purposes.

```
classifier = GaussianNB() #create object from GaussianNB class
classifier.fit(X_train, y_train) #fit model to dataset
y_pred = classifier.predict(X_test) #predict test set using fitted model
print ("Accuracy: ", accuracy_score(y_test, y_pred)) #measure model accuracy
```

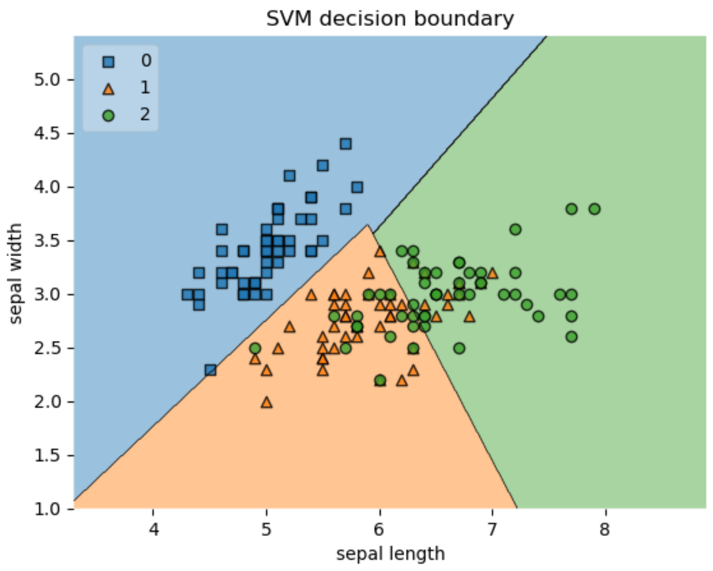
again, we're going to use only sepal length and width for fit a gaussian naïve bayes model. We also replace our classes values with numeric values to draw the boundary like follows:

```
X_data = iris_df.iloc[:,0:2].values
y_labels =
iris_df.species.replace({'setosa':0,'versicolor':1,'virginica':2}).copy().values
nb = GaussianNB().fit(X_data,y_labels)
fig = plot_decision_regions(X=X_data, y=y_labels,clf=nb, legend=2)
plt.title('naive bayes decision boundary')
plt.xlabel('sepal length')
plt.ylabel('sepal width')
```



Support Vector Machine (SVM) Model:

```
model = SVC(kernel='linear')
model.fit(X_train,y_train)
print ("Accuracy : ", accuracy_score(y_test, model.predict(X_test)))
#draw boundaries
model1 = SVC(kernel='linear')
model1.fit(X_data  ,y_labels)
fig = plot_decision_regions(X=X_data, y=y_labels,clf=model1, legend=2)
plt.title('SVM decision boundary')
plt.xlabel('sepal length')
plt.ylabel('sepal width')
```



References:

1- https://www.geeksforgeeks.org/exploratory-data-analysis-on-iris-dataset/
2- Mlxtend docs: http://rasbt.github.io/mlxtend/
3- https://towardsdatascience.com/machine-learning-basics-naive-bayes-classification-964af6f2a965
4- https://www.pycodemates.com/2022/10/classification-of-iris-dataset-using-SVM-in-python.html
5- Pandas docs https://pandas.pydata.org/docs/ .
6- NumPy docs https://numpy.org/doc/ .
7- Matplotlib docs https://matplotlib.org/3.3.3/contents.html .
8- SciKit-learn docs https://scikit-learn.org/0.21/documentation.html .
9- http://rasbt.github.io/mlxtend/user_guide/plotting/plot_decision_regions/#example-1-decision-regions-in-2d