Machine Learning Model for iris dataset classification

Table of Contents

Problem statement

Which is better: K-NN or SVM to solve the classification problem on the Iris dataset, and which one has high accuracy when training the data.

1. Introduction

In this report we used two of the supervised machine learning modules; K-Nearest Neighbors and Support Vector Machine to classify the Iris species in the Iris dataset.

1.1 K-Nearest Neighbors (K-NN)

K-NN is a supervised machine learning technique that solves both classification and regression problems. It predicts the value of any new data point based on who this point is close to the other known points of data sorted in the model (training data); it uses feature similarity measures to assign a data point to a class. K-NN decides to assign a new data point to a specific class by the majority voting of this point k closest neighbors and assigning it to the most common class among them. For example, if k=3 the new point class will be determined by the most common class of the nearest three data points to it. Nearest neighbors are determined by calculating the distance between the new data point and all stored data points [1].

If k is too large, there is a high potential of misclassifying the new data point as the nearest neighbors are far away from its neighborhood. At the same time, if k is too small, we will face an overfitting problem and lead to a high error rate on the validation set [2]. K could be calculated as the square root of the number of data points in the training data ‘Eq. 1’ [3]. It is better to keep k odd to avoid confusion between two data classes.

$k=sqrt(N)$ (1)

K-NN calculates the data distances using different methods such as Euclidean distance, Manhattan distance, Hamming distance, and Minkowski distance.

Euclidean distance is the most used function to calculate the data distance. It’s calculated as the square root of the sum of the squared differences between a new point (x) and an existing point (y) ‘Eq. 2’. The hamming distance is used for categorical data to calculate binary distance, If the (x) value equals (y) value they are the same, and the distance equals zero ‘Eq.3’. Otherwise, distance equals one [4].

$$d_{Euclidian}(x, y) = \sqrt{\sum_{i=1}^{k} (x_i - y_i)^2} \quad (2)$$

$$d_{Hamming} = \sum_{i=1}^{k} |x_i - y_i| \quad (3)$$

K-NN training is fast and costs zero. Also, it is simple to implement and can deal with multiclasses and noise data. On the other hand, it is slow to predict because it computes the distance of each query instance to all training data points. Furthermore, it is a lazy algorithm that needs large memory to store training data [2].

1.2 Support Vector Machine (SVM)

SVM is another supervised machine learning technique that was introduced in 1992 and used for solving classification, regression, and feature selection problems [2]. In classification, SVM finds the boundaries to separate different classes using the margin concept. SVM drew on determining the hyperplane that best divides a dataset into two classes. In which margin is the distance between the hyperplane and the closest points to it on either side; the maximum margin is better [5]. There are two cases when using SVM for classification: linearly separable case and non-linearly separable case. In a linearly separable case, with the condition that both classes are classified correctly, SVM tries to find the hyperplane that maximizes the margin. On the other hand, when SVM faces a highly disorganized pattern (non-linearly separable), it uses the soft margin and kernel trick. Soft margin allows the existence of some points on the incorrect side without violating the constraints or affecting results. Based on the position and distance of the incorrect points from the hyperplane, these points have a penalty. The soft margin goal is to minimize the incorrectly classified data points and classify other data points correctly with sufficient distance from the margin ‘Eq. 4’. Where c is the positive parameter; that controls the trade-off between the slack variable penalty and the size of the margin.

$$min \frac{1}{2} ||w||^2 + C \sum_{i=1}^{N} \xi_i \quad st.\ y_i(w^t x + b) + \xi i \geq 1 \ (4)$$

Too small C allows more data points to be located on the wrong side, and it may underfit the training data. Also, too large C may overfit the data which leads to poor generalization [2].

Kernels help in finding a non-linear boundary. It is used to nonlinearly map the training data from the input space to a higher dimensional feature space that refers to the collections of features that are used to characterize the training data by a nonlinear function. There are many types of kernels that we can choose from in Sklearn; linear ‘Eq. 5’, poly ‘Eq. 6’, radial basis function (RBF)‘Eq.

7', sigmoid 'Eq. 8', or precomputed.

$$f_k(x, y) = x^T Y + C \quad (5)$$

$$f_k(x, y) = (x^T Y + \beta)^d \quad (6)$$

$$f_k(x, y) = exp(-\frac{||x-y||^2}{2\sigma^2}) \quad (7)$$

$$f_k(x, y) = tanh(\gamma(x^T \cdot Y) + \alpha) \quad (8)$$

The RBF and polynomial are the most used kernels; the linear kernel is only appropriate for linear problems, and the polynomial kernel has computational difficulties. RBF is a processor that measures the distance between all other dots to a specific dot/dots to generate new features. RBF is widely used neural network architecture in literature for regression problems [6].

SVM training is relatively easy, it is effective and scales well to the high dimensional data, it can deal with both categorical and continuous data, and you can control the trade-off between the model complexity and error easily. Because SVM is a non-parametric technique, no assumptions are required regarding the data structure. It captures the nonlinear relationships in the data with very high prediction accuracy. On the other hand, SVM does not work well with overlapped classes data, its results lack transparency as it is a non-parametric method. Also, it is tricky to choose the best kernel function, it requires a lot of time to process large data and could be computationally expensive [2].

2. Experiment Objective

In this project, we are working on the iris dataset. Our objective is to build two supervised machine learning models that can classify iris into three classes (Iris species), Iris-setosa, Iris-versicolor, and Iris-virginica, with high accuracy. These classes are determined based on four attributes: Sepal Length, Sepal Width, Petal Length, and Petal Width in centimeters. After developing the K-NN and SVM models we will determine which one of them is better based on their accuracy.

3. Why We Choose KNN And SVM:

3.1. KNN:

It used for both classification and regression problems. KNN is suitable for lower dimensional data as Iris data given. You can try it on high dimensional data (hundreds or thousands of input variables) but be aware that it may not perform as well as other techniques. KNN can benefit from feature selection that reduces the dimensionality of the input feature space. KNN is an algorithm that is useful for matching a point with its closest k neighbors in a multi-dimensional space.so, it can be used for data that are continuous, discrete like Iris data given.

3.2. SVM:

Because there was a clear margin of separation between classes, very helpful when there are more dimensions than samples, it can work with numerical continuous dataset very well and relatively effectively with memory. Normal SVM is not suitable for classification of large data sets, because the training complexity of SVM is very high, on the other hand our Iris dataset is small data given.

Note: another reason for choosing the above two models is due to the continuous features of Iris dataset; the other two models' decision trees and naive bayes work with high accuracy with categorical features.

4. Steps For Building ML Models:

Although different machine learning techniques use different methods to train the model, most models use the same fundamental procedures. To train algorithms properly, a lot of high-quality data must be available. To ensure that the model is as effective as possible, many of the phases deal with the preparation of this data. To ensure that a model meets the unique goals of the organization, the entire project must be carefully planned and controlled from the start. Therefore, setting the project's context inside the organization is the first stage.

The following are the six steps to creating a machine learning model:

1. Choose the data
2. Data preparation

3. Feature selection
4. Train model
5. Make prediction
6. Model evaluation

1. Choose The Data:

Iris has a tri class target variable and four numerical attributes. Both clustering and classification may be done with this dataset. Iris is the "hello world" of machine learning, according to data scientists. load and explore the well-known dataset of iris plant species. Iris dataset is continuous data and doesn't have null that can affect incorrect results.

2. Data Preparation:

Once choosing, the data needs to be prepared for the next step. Data preparation is the phase of placing data in an appropriate location and preparing it for use in machine learning training. To understand the nature of the data that we must work with, we need to understand the nature, format, and quality of the data. The more we understand our data the more we have accurate results

a. Import The Library:

```
In [131]: # Import libraries
          import numpy as np
          import pandas as pd
          import matplotlib.pyplot as plt
          import seaborn as sns
          from sklearn.neighbors import KNeighborsClassifier
          from sklearn.svm import SVC
          from sklearn.preprocessing import QuantileTransformer
          from sklearn import preprocessing
          from sklearn.model_selection import train_test_split
          from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
          from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
          from sklearn.model_selection import cross_val_score
```

Fig. 1. import library

In this step, as shown in (Fig. 1) we choose the library that will help us, numpy library is a Python package that is used to manipulate arrays. Additionally, it has matrices, Fourier transform, and functions for working in linear algebra. Panda library is capable of quickly loading data from various databases and data formats: can be utilized with a wide variety of data types. Matplotlib provides a variety of graphical displays, including Bar Graphs, Histograms, Line Graphs, Scatter

Plots, Stem Plots, etc. seaborn Clearly visualized our data on a plot. We can use this library to visualize our data without having to worry about the inside workings; all we must do is feed our data set or data inside the replot () function, and it will compute and place the value appropriately. sklearn it's a versatile, user-friendly tool that can create neuroimages and anticipate consumer behavior, among other things. It is cost-free and simple to use.

b. Reading The Data

```
In [132]: # Reading dataset
          DATA_PATH = "C:\\Users\\Abdelrahman Ahmed\\Downloads\\Data set 1 (5 KB) - iris.csv.csv"
          df = pd.read_csv(DATA_PATH)
```

Fig. 2. reading the data

As shown in (Fig. 2), in this step we load the csv file using the above code, access data from the csv file and retrieve data in the form of a data frame.

c. Check Some Rows of Data Frames:

```
In [133]: # check some rows of dataframes
          df.head(10)
```

Out[133]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

Fig. 3. check some rows of data frames

As shown in (Fig. 3), This code retrieves the first number of the rows.

d. Check Duplicates:

```
In [134]: # Check Duplicates
          df = df.drop_duplicates()
```

Fig. 4. check duplicates

As shown in (Fig. 4), in this code we check if there is duplicate data and order python to delete it, there is no duplicate in our data set.

e. Check Null Values:

```
In [135]: # Check Null Values
          df.isna().sum()

Out[135]: Id              0
          SepalLengthCm   0
          SepalWidthCm    0
          PetalLengthCm   0
          PetalWidthCm    0
          Species         0
```

Fig. 5. check null values

As shown in (Fig. 5), this code to see if there are empty values, there are no null values in our data set.

f. Data Types:

```
In [138]: # Data types
          df.info()

          <class 'pandas.core.frame.DataFrame'>
          Int64Index: 150 entries, 0 to 149
          Data columns (total 6 columns):
           #   Column         Non-Null Count  Dtype
          ---  ------         --------------  -----
           0   Id             150 non-null    int64
           1   SepalLengthCm  150 non-null    float64
           2   SepalWidthCm   150 non-null    float64
           3   PetalLengthCm  150 non-null    float64
           4   PetalWidthCm   150 non-null    float64
           5   Species        150 non-null    object
          dtypes: float64(4), int64(1), object(1)
          memory usage: 8.2+ KB
```

Fig. 6. data types

As shown in (Fig. 6), in this code we retrieve the info about all the data in the csv file, check data types, we found the species (label column) data type object and it should be category. So, we will change species data type.

g. Changing Data Types:

```
In [139]: # Changing data types
          df["Species"] = df["Species"].astype("category")
```

Fig. 7. changing data types

As shown in (Fig. 7), In this code we change species from object to categories.

h. Check Data Types:

```
In [140]: # Check data types
          df.info()

          <class 'pandas.core.frame.DataFrame'>
          Int64Index: 150 entries, 0 to 149
          Data columns (total 6 columns):
           #   Column         Non-Null Count  Dtype
          ---  ------         --------------  -----
           0   Id             150 non-null    int64
           1   SepalLengthCm  150 non-null    float64
           2   SepalWidthCm   150 non-null    float64
           3   PetalLengthCm  150 non-null    float64
           4   PetalWidthCm   150 non-null    float64
           5   Species        150 non-null    category
          dtypes: category(1), float64(4), int64(1)
          memory usage: 7.3 KB
```

Fig. 8. check data types

As shown in (Fig. 8), in this code we want to make sure that the species types changed.

i. Some Statistics

```
In [141]: # Some statistics
          df[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm"]].describe()
```

Out[141]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
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

Fig. 9. some statics

As shown in (Fig. 9), in this code we made Summary statistics of the data provided. This includes mean, count, std deviation, percentiles, and min-max values of all the features.

3. Feature Selection:

Checking data distribution; checking the normality distribution of our features as our models work well with the normal distribution data without outliers.

checking the correlation pair plot; correlated features with high correlated accuracy increase the variance of our model that will reduce the test accuracy of our model.

Correlation heatmap; another plot with annotation of correlated accuracy number to ensure our feature selection.

Checking imbalance classes for only classification; to prevent the model being biased towards one of the classification categories.

a. Checking Data Distribution:

```
In [13]: # Checking data distribution
import matplotlib.pyplot as plt
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(
                                    ncols=2,
                                    nrows=2,
                                    figsize=(15, 6))

ax1.hist(df["SepalLengthCm"])
ax1.set_title("SepalLengthCm Distribution", fontsize=15)

ax2.hist(df["SepalWidthCm"])
ax2.set_title("SepalWidthCm Distribution", fontsize=15)

ax3.hist(df["PetalLengthCm"])
ax3.set_title("PetalLengthCm Distribution", fontsize=15)

ax4.hist(df["PetalWidthCm"])
ax4.set_title("PetalWidthCm Distribution", fontsize=15)
plt.tight_layout()
```

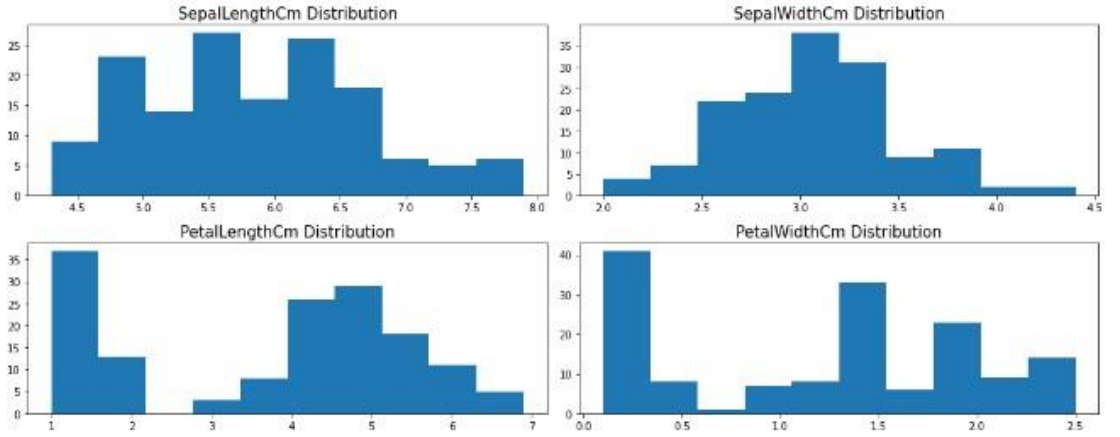

Fig. 10. checking data distribution

Use the Python module Matplotlib to draw a histogram, as shown in (Fig. 10), to know the distribution of each column.

b. Checking The Correlation Pair Plot:

Fig. 11. check the correlation pair plot

Using the python seaborn library to visualize, as shown in (Fig. 11), if there is correlation between the variables, our data set has a correlation between two features; pathlengths and petal wdith cm, and this will affect on the model accuracy when we build the model, so we will drop one of them before building our ML model.

c. Correlation Heatmap:

```
In [144]: # Correlation heatmap
corr_df = df[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm",
              "PetalWidthCm"]].corr()
dataplot = sns.heatmap(corr_df, cmap="YlGnBu", annot=True)
```

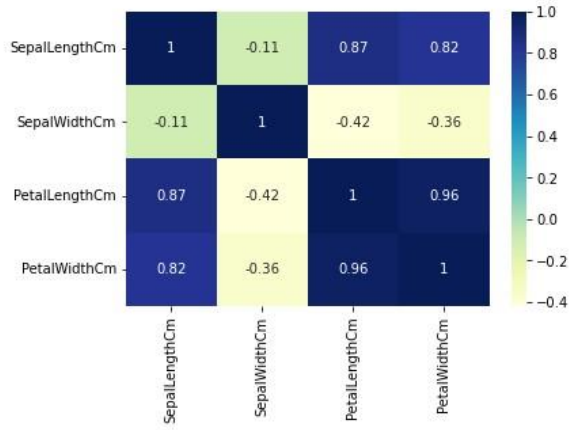

Fig. 12. correlation heatmap

We utilized it to determine how various features relate to one another and which features are most useful for developing ML models, as shown in (Fig. 12). where lesser amounts are represented by lighter hues, and bigger amounts by darker shades, the heat map also confirms the result we have from the previous step (Fig. 11).

d. Checking Imbalance Classes For Only Classification:

```
In [145]: # Checking imbalance classes for only classification
class_df = df.groupby(by=["Species"]).count().reset_index()
plt.bar(class_df["Species"], class_df["Id"])

print("Balanced dataset, no need to over/undersampling")
```

```
Balanced dataset, no need to over/undersampling
```

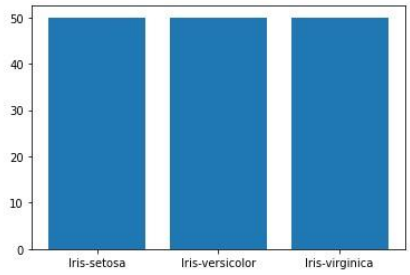

Fig. 13. checking imbalance classes

most machine learning algorithms presumptively distribute data equally. Due to this bias towards the dominant class, the machine learning classifier performs poorly when there is a class imbalance, misclassifying the minority class. Our dataset has balanced classes, as shown in (Fig.

13).

4. Train The Model:

Through the dataset, we choose different algorithms to train the data to have the result we want.

a. Selecting The Final Features Labels:

```
In [146]: # Selecting the final features, labels
          df = df.drop(columns=["Id"])

          features = df[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm"]]
          labels = df["Species"]
```

Fig. 14. selecting the final feature

While selecting the final features we found the id column is not important. So, we gave the order to delete this column, and to drop the petal width cm column because this column highly correlated to another feature (petal length cm) as shown (Fig. 11) and (Fig. 12), and we gave the order to use the rest of the columns, as shown in (Fig. 14).

b. Getting Features Normal Distributed And Remove Outliers:

```
In [147]: # Getting features normal distribuated and remove outliers
          quantile = QuantileTransformer(output_distribution='normal')
          features = quantile.fit_transform(X=features)

          C:\Anaconda3\lib\site-packages\sklearn\preprocessing\_data.py:2612: UserWarning: n_quantiles (1000) is greater than the total n
          umber of samples (150). n_quantiles is set to n_samples.
            warnings.warn("n_quantiles (%s) is greater than the total number "
```

Fig. 15. remove outlier

Removal of outliers gives some variables a normal distribution and improves the effectiveness of transformations for the other variables, as shown in (Fig. 15). Encoding is used for any categorical column in my dataset:

```
In [148]: # Encoding is used for any categorical column in my dataset
          label_encoder = preprocessing.LabelEncoder()
          labels = label_encoder.fit_transform(labels)
```

Fig. 16. encode dataset

Ordinal encoding created by converting categorical label column to encoded column, as (Fig. 16) shows.

D. Train Test Split:

```
In [149]: # Train test split
training_features, X_test , training_y, y_test = train_test_split(
    features, labels, test_size=.15, random_state=1, shuffle=True)

X_train, X_valid , y_train, y_valid = train_test_split(
    training_features, training_y, test_size=.2, random_state=1, shuffle=True)
```

```
In [150]: print("The shape of X_train : ", X_train.shape)
print("The shape of X_valid : ", X_valid.shape)
print("The shape of X_test :  ", X_test.shape)

The shape of X_train :  (101, 3)
The shape of X_valid :  (26, 3)
The shape of X_test :   (23, 3)
```

Fig. 17. train test and rows for each train, valid and test

Training data, this kind of data is used to develop the machine learning algorithm. The data scientist provides input data to the algorithm, which correlates to an expected outcome. The model assesses the data regularly to learn more about its behavior and then adapts itself to fit its intended function. During training, validation data introduces fresh data into the model that it hasn't previously analyzed.

Validation data provides, the first test against previously unknown data, allowing data scientists to assess how well the model predicts based on the new data Validation data is not used by all data scientists, but it might give some useful information for optimizing hyperparameters, which impact how the model evaluates data.

Test data, testing data ensures that the model can generate correct predictions when it is developed. If the training and validation data contain labels to monitor the model's performance metrics, the testing data should remain unlabeled. Test data is a final, real-world verification of an unknown dataset to ensure that the ML algorithm was properly trained.
Aa shown in (Fig. 17), we choose in range (15%- 20%) of validation and test data.:

1- To make sure the model will be trained on enough training data, so we did not take more than 20%.
2- We did not take below 15% to make sure that the model will be tested on all different cases.

So, the numbers of rows for the training data will be 101 row , validation data rows will be 26 and test data rows number will be 23, as shown in (Fig. 17).

5. Make Prediction:

After we finish training our model, we test the model. Check the model's accuracy by providing it with a test dataset.

a. Fitting The Data:

```
In [152]: # Hyperparameters Tuning
          # Choosing these two models compaared to the others because the other models
          # needs categorical features to get best accuracy
          svc_model = SVC(C=1.5, kernel="rbf", gamma=1.2)
          knn_model = KNeighborsClassifier(n_neighbors=6)

          # Fitting the data
          svc_model.fit(X_train, y_train)
          knn_model.fit(X_train, y_train)

Out[152]: KNeighborsClassifier(n_neighbors=6)
```

Fig. 18. fitting the data

As shown in (Fig. 18), to obtain greater accuracy, the fit function modifies the weights in accordance with the data values. The model can be applied to predictions after training. The results your model generates won't be accurate enough to be useful for making practical decisions if it doesn't fit your data appropriately. We chose this specific parameter for each model to get high accuracy after we tried a lot some cases.

b. Evaluation To Get The Best Model By The Accuracy:

```
In [172]: # Getting the predictions labels of validation
          # ROC AUC Percsion Recall Accuracy Crosss validation F1 score- Confusion Metrics
          svc_train_pred = svc_model.predict(X_train)
          knn_train_pred = knn_model.predict(X_train)
          svc_valid_pred = svc_model.predict(X_valid)
          knn_valid_pred = knn_model.predict(X_valid)

          # Evalution to get the best model
          print("The accuracy on training data")
          print("svc training accuracy : ", accuracy_score(y_true=y_train,
                                                           y_pred=svc_train_pred))
          print("knn training accuracy : ", accuracy_score(y_true=y_train,
                                                           y_pred=knn_train_pred))

          print()
          print("Cross validation")
          print("Cross vaildation of svc", cross_val_score(svc_model, X_train, y_train, cv=6).mean())
          print("Cross vaildation of knn", cross_val_score(knn_model, X_train, y_train, cv=6).mean())

          The accuracy on training data
          svc training accuracy :  0.9702970297029703
          knn training accuracy :  0.9702970297029703

          Cross validation
          Cross vaildation of svc 0.9607843137254902
          Cross vaildation of knn 0.9117647058823529
```

Fig. 19. evaluation to get best model by the accuracy

As shown in (Fig. 19), the two models KNN and SVC have almost the same result on the training accuracy, but SVC has high accuracy in different cases of cross validation.

6. Model Evaluation:

It's the last step in ML. If the model we used produces an accurate result as per our requirement with acceptable speed, then we deploy the model in the real system. However, before implementing a project, we will use the available data to see if it improves its performance or not.
The model evaluation phase is like making the final report for a project.

5. Comparison And Result:

5.1. Accuracy Comparison:

```
The accuracy on training data
svc training accuracy :  0.9702970297029703
knn training accuracy :  0.9702970297029703

Cross validation
Cross vaildation of svc 0.9607843137254902
Cross vaildation of knn 0.9117647058823529
```

Fig. 20. Accuracy comparison between K-NN and SVM

5.2. Accuracy:

The accuracy on training data shows that SVM accuracy equals KNN accuracy equal to 0.97, the accuracy of validation of SVM = 0.96 and the accuracy of validation of KNN=0.91 so SVM is more accurate than KNN in validation, as shown in (Fig. 20).

5.3. Confusion Matrix:

A confusion matrix is a table that is used to define the performance of a classification algorithm. A confusion matrix visualizes and summarizes the performance of a classification algorithm. We implement code to draw the confusion matrix to show how good our model is.

```python
svc_test_pred = svc_model.predict(X_test)
knn_test_pred = knn_model.predict(X_test)

# Plotting svc confusion matrix
plt.figure(figsize=(15,7))
plt.subplot(1,2,1)
conf_mat = confusion_matrix(y_true=y_test, y_pred=svc_test_pred)
labels = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
my_plot = sns.heatmap(conf_mat, annot=True, xticklabels=labels,
                      yticklabels=labels,)
my_plot.set_xlabel("Predicted", fontdict={"size":15})
my_plot.set_ylabel("True", fontdict={"size":15})
plt.title("confusion matrix of svc", fontdict={"size": 15})

# Plotting knn confusion matrix
plt.subplot(1,2,2)
conf_mat = confusion_matrix(y_true=y_test, y_pred=knn_test_pred)
labels = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
my_plot = sns.heatmap(conf_mat, annot=True, xticklabels=labels,
                      yticklabels=labels,)
my_plot.set_xlabel("Predicted", fontdict={"size":15})
my_plot.set_ylabel("True", fontdict={"size":15})
plt.title("confusion matrix of knn", fontdict={"size": 15})
```

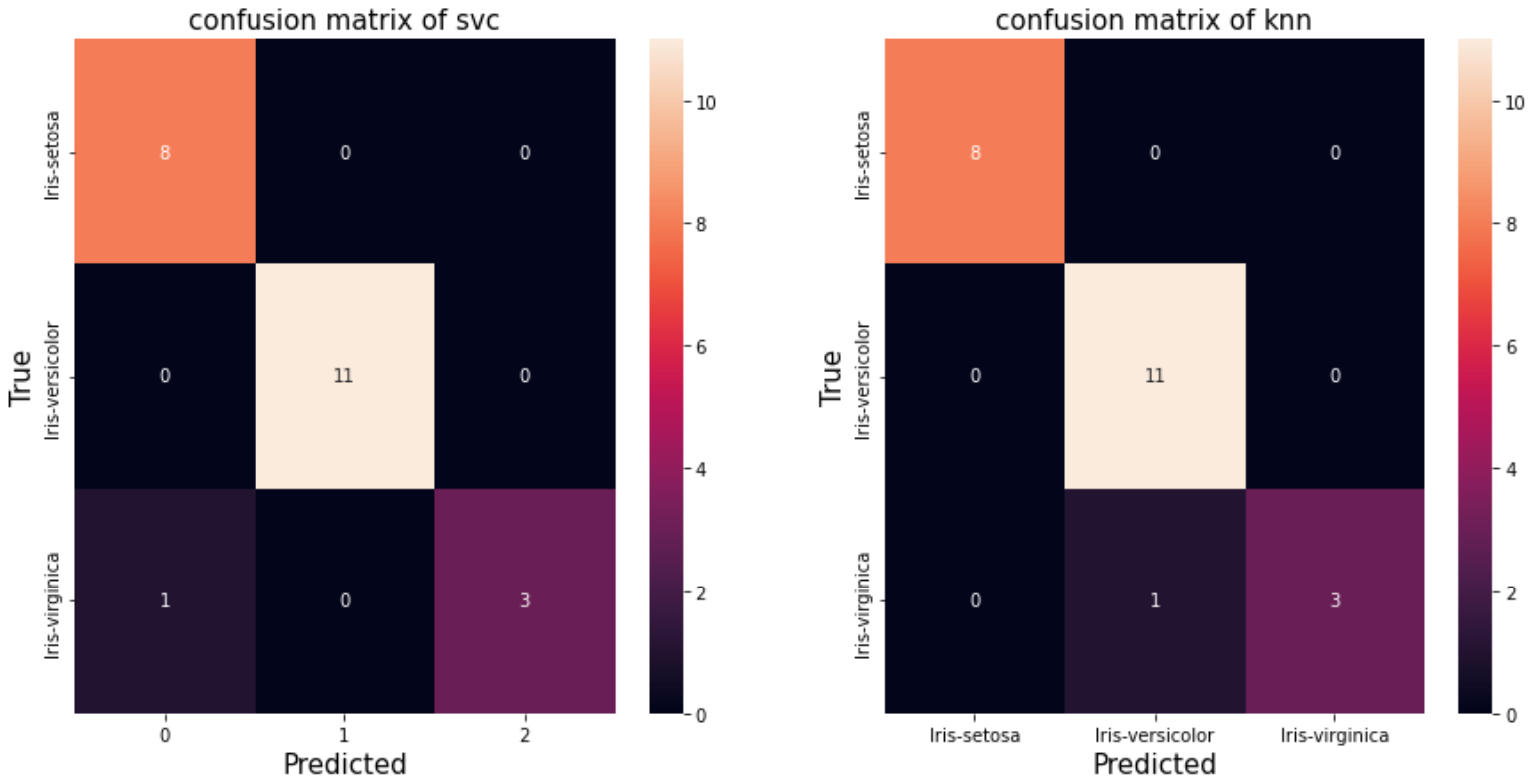

Fig. 21. Code of confusion matrix comparison of K-NN and SVC and the result

As shown in (Fig. 21),

SVM model predicted the iris-setosa 8 and they are true positive.

SVM model predicted the iris-versicolor 11 and they are true positives.

SVM model predicted the iris-virginica 3 and they are true positives, and 1 false negative type (Iris- setosa).

The KNN model predicted the iris-setosa 8 and they are true.

The KNN model predicted the iris-versicolor 11 and they are true.

The KNN model predicted the iris-virginica 3 and they are true, and 1 false negative type (Iris-versicolor).

5.4 Classification Report

Classification Report of SVC

	precision	recall	f1-score	support
0	0.89	1.00	0.94	8
1	1.00	1.00	1.00	11
2	1.00	0.75	0.86	4

Fig. 22. SVM precision and recall

Classification Report of knn

	precision	recall	f1-score	support
0	1.00	1.00	1.00	8
1	0.92	1.00	0.96	11
2	1.00	0.75	0.86	4

Fig. 23.KNN precision and recall

As shown from (Fig.19, Fig.20, Fig.21, Fig.22, Fig.23), it appears from the results that SVM is a bit better than K-NN, as the two models have same training accuracy equal to 0.97, but SVC is more accurate in validation test and classification report than K-NN.

6. Conclusion

This report presented the solution of classification problem using Iris dataset that consists of continuous features and categorical labels. Two models were selected, K-NN and SVM models to solve this classification problem, because both are more suitable for continuous features than the other models. We made an appropriate preprocessing and preparation that helped us to get a high level of accuracy. We get that the two models fitting on the data in the best appropriate manner, that reflected on the accuracy of the two model, the accuracy of two models slightly different from each other but SVM is better than K-NN, we made confusion matrix, recall and precision to ensure the accuracy of the two models. As shown above the result obtained from applied SVM and K-NN on Iris dataset is the training accuracy for each of them = 0.97, but cross validation accuracy for SVC= 0.96 and cross validation accuracy for K-NN = 0.91, so SVC is more accurate than K-NN.

Also, we learned and practiced a lot of libraries like panda, numpy, matplotlib, seaborn and more other libraries as we mentioned above for each step that we used on the code.
Also, we learned how to deal with the data in the cleaning step and preprocessing step. These phases are very important because these steps increase the accuracy level when we train the model, and we may drop some variables that depend on the correlation between variables as we mentioned above. And more about these skills we get and more problems we face that lead us to search to learn it.

References:

[1] A. Kadhim, "Survey on supervised machine learning techniques for automatic text classification", *Artificial Intelligence Review*, vol. 52, no. 1, pp. 273-292, 2019. Available: 10.1007/s10462-018-09677-1.

[2] A.Mohamed, "Comparative Study of Four Supervised Machine Learning Techniques for Classification" , International Journal of Applied Science and Technology ,Vol. 7, No. 2, June 2017

[3] W. Raseman, B. Rajagopalan, J. Kasprzyk and W. Kleiber, "Nearest neighbor time series bootstrap for generating influent water quality scenarios", *Stochastic Environmental Research and Risk Assessment*, vol. 34, no. 1, pp. 23-31, 2020. Available: 10.1007/s00477-019-01762-3.

[4] J. Salvador–Meneses, Z. Ruiz–Chavez and J. Garcia–Rodriguez, "Compressed kNN: K-Nearest Neighbors with Data Compression", *Entropy*, vol. 21, no. 3, p. 234, 2019.

[5] W. Chen, H. Pourghasemi, A. Kornejady and N. Zhang, "Landslide spatial modeling: Introducing new ensembles of ANN, MaxEnt, and SVM machine learning techniques", *Geoderma*, vol. 305, pp. 314-327, 2017.

[6] M. Ghorbani, H. Zadeh, M. Isazadeh and O. Terzi, "A comparative study of artificial neural network (MLP, RBF) and support vector machine models for river flow prediction", *Environmental Earth Sciences*, vol. 75, no. 6, 2016. Available: 10.1007/s12665-015-5096-x.