2020/2021

Mini-Project

Presented in order to obtain

Final Grade of Mini-Project at Lebanese University – Faculty of Engineering III

Specialty: Telecommunication Engineering

Prepared By:

Julien El Dib (5556)

Rida Diab (5515)

Classifying Iris using Machine Learning

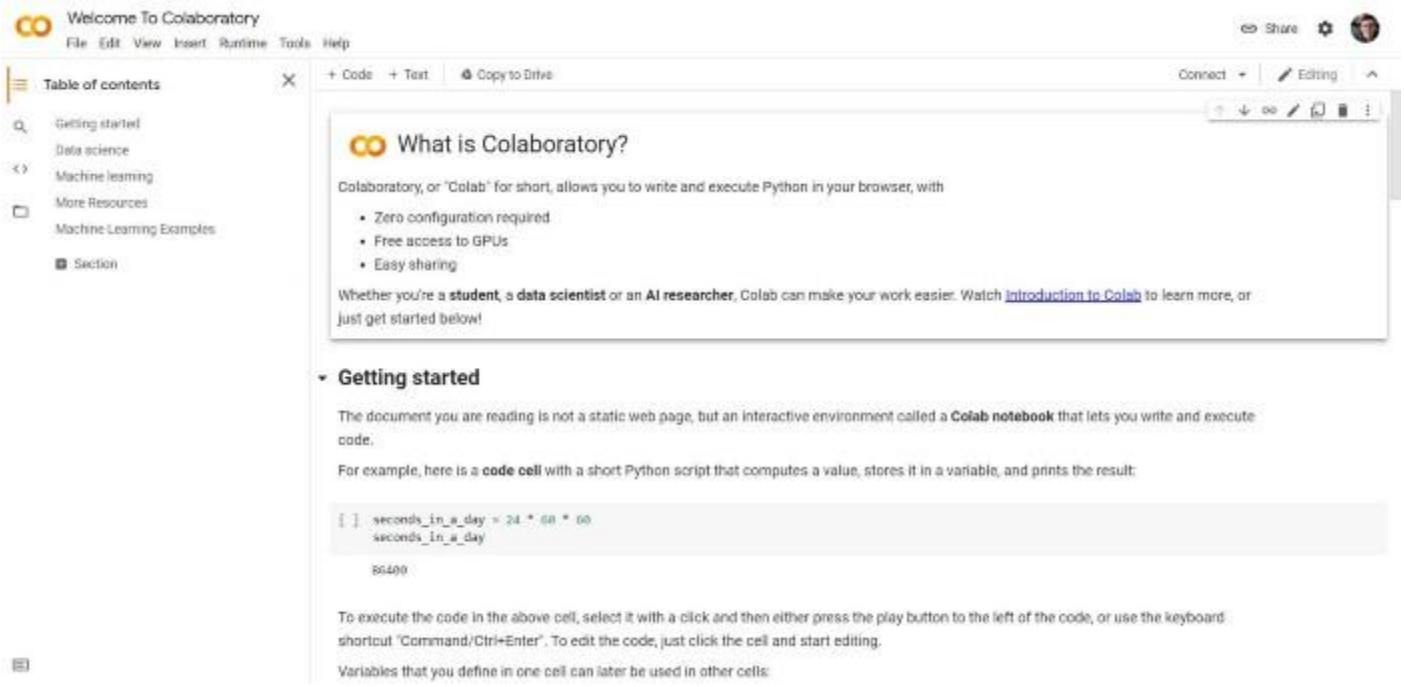
Under the direction of:

Dr. Mohamad Aoude

Submitted on **22** July 2021

Contents

Google Colab



Google Colab — Welcome page

Google Colab is a platform that allows you to run code directly on the cloud, this means that you can use very powerful hardware to run your code and the only requirement to do it is to have a Google account.
In Google Colab you can only use **Python** as a programming language but this is fairly enough for the features it offers. Every line of code you write is automatically saved on your **Google Drive** storage and you can easily access your project notebook whenever you want, wherever you are.

KNeighborsClassifier

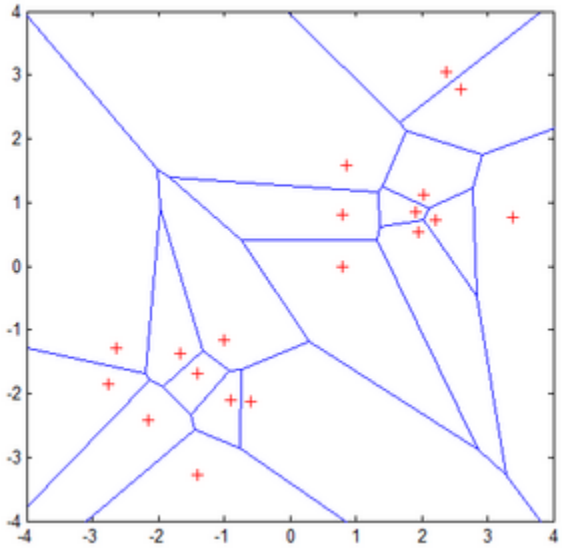
K-nearest-neighbor (kNN) classification is one of the most fundamental and simple classification methods and should be one of the first choices for a classification study when there is little or no prior knowledge about the distribution of the data. K-nearest-neighbor classification was developed from the need to perform discriminant analysis when reliable parametric estimates of probability densities are unknown or difficult to determine.

Between-sample geometric distance

The k-nearest-neighbor classifier is commonly based on the Euclidean distance between a test sample and the specified training samples. Let xi be an input sample with p features (xi1,xi2,…,xip) , n be the total number of input samples (i=1,2,…,n) and p the total number of features (j=1,2,…,p) . The Euclidean distance between sample xi and xl (l=1,2,…,n) is defined as

$$d(\mathbf{x}_i,\mathbf{x}_l)= \sqrt{(x_{i1}-x_{l1})^2+(x_{i2}-x_{l2})^2+\cdots+(x_{ip}-x_{lp})^2}$$

A graphic depiction of the nearest neighbor concept is illustrated in the Voronoi tessellation shown in the Figure. The tessellation shows 19 samples marked with a "+", and the Voronoi cell, *R* , surrounding each sample. A Voronoi cell encapsulates all neighboring points that are nearest to each sample and is defined as



$$R_i=\{\mathbf{x}\in \mathrm{R}_p : d(\mathbf{x},\mathbf{x}_i)\le d(\mathbf{x},\mathbf{x}_m), \quad \forall i\neq m\},$$

where *Ri* is the Voronoi cell for sample $\mathbf{x}i$, and **x** represents all possible points within Voronoi cell *Ri* . Voronoi tessellations primarily reflect two characteristics of a coordinate system: i) all possible points within a sample's Voronoi cell are the nearest neighboring points for that sample, and ii) for any sample, the nearest sample is determined by the closest Voronoi cell edge. Using the latter characteristic, the k-nearest-neighbor classification rule is to assign to a test sample the majority category label of its k nearest training samples. In practice, k is usually chosen to be odd, so as to avoid ties.

Classification decision rule and confusion matrix

Classification typically involves partitioning samples into training and testing categories. Let **x***i* be a training sample and **x** be a test sample, and let ω be the true class of a training sample and $\omega\hat{}$ be the predicted class for a test sample ($\omega,\omega\hat{}=1,2,\ldots,\Omega$) . Here, Ω is the total number of classes.

During the training process, we use only the true class ω of each training sample to train the classifier, while during testing we predict the class $\omega\hat{}$ of each test sample. It warrants noting that kNN is a "supervised" classification method in that it uses the class labels of the training data.

With 1-nearest neighbor rule, the predicted class of test sample **x** is set equal to the true class ω of its nearest neighbor, where **m***i* is a nearest neighbor to **x** if the distance $d(\mathbf{m}i,\mathbf{x})=\min j\{d(\mathbf{m}j,\mathbf{x})\}$.
For k-nearest neighbors, the predicted class of test sample **x** is set equal to the most frequent true class among *k* nearest training samples. This forms the decision rule $D:\mathbf{x}\rightarrow\omega\hat{}$.

The confusion matrix used for tabulating test sample class predictions during testing is denoted as **C** and has dimensions Ω×Ω . During testing, if the predicted class of test sample **x** is correct (i.e., $\omega\hat{}=\omega$), then the diagonal element $C_{\omega\omega}$ of the confusion matrix is incremented by 1. However, if the predicted class is incorrect (i.e., $\omega\hat{}\neq\omega$), then the off-diagonal element $C_{\omega\omega\hat{}}$ is incremented by 1. Once all the test samples have been classified, the classification accuracy is based on the ratio of the number of correctly classified samples to the total number of samples classified, given in the form

$$Acc=\left|\frac{\sum_{\omega}^{\Omega} C_{\omega\omega}}{n_{total}}\right|$$

where $C_{\omega\omega}$ is a diagonal element of **C** and n_{total} is the total number of samples classified.

Our Project

We are going to build a machine learning model to determine to which species an Iris flower belongs to. The Iris dataset contains tabular data about characteristics of the flower exemplars like petal width and length, which are used as input to the model. The output will be an integer indicating representing one of the 3 possibilities of species: Iris Setosa, Iris Versicolour or Iris Virginica. The next sections will describe the construction of the model, step by step.

1-Setup

In this section we import the necessary libraries so you can build your model.

```
import numpy as np
import pandas as pd

from matplotlib import pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix

from joblib import dump, load
```

2-Load the data

The first step is to load the necessary data. Use the command read_csv() from pandas library to load the Iris dataset. After loading the data into a dataframe, show the top of the dataset. The dataset file URL is https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data.

```
cols = ['sepal_length', ' sepal_width', 'petal_length', 'petal_width', 'class']
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data', names=cols)
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	class
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

3-Explore and visualize the data

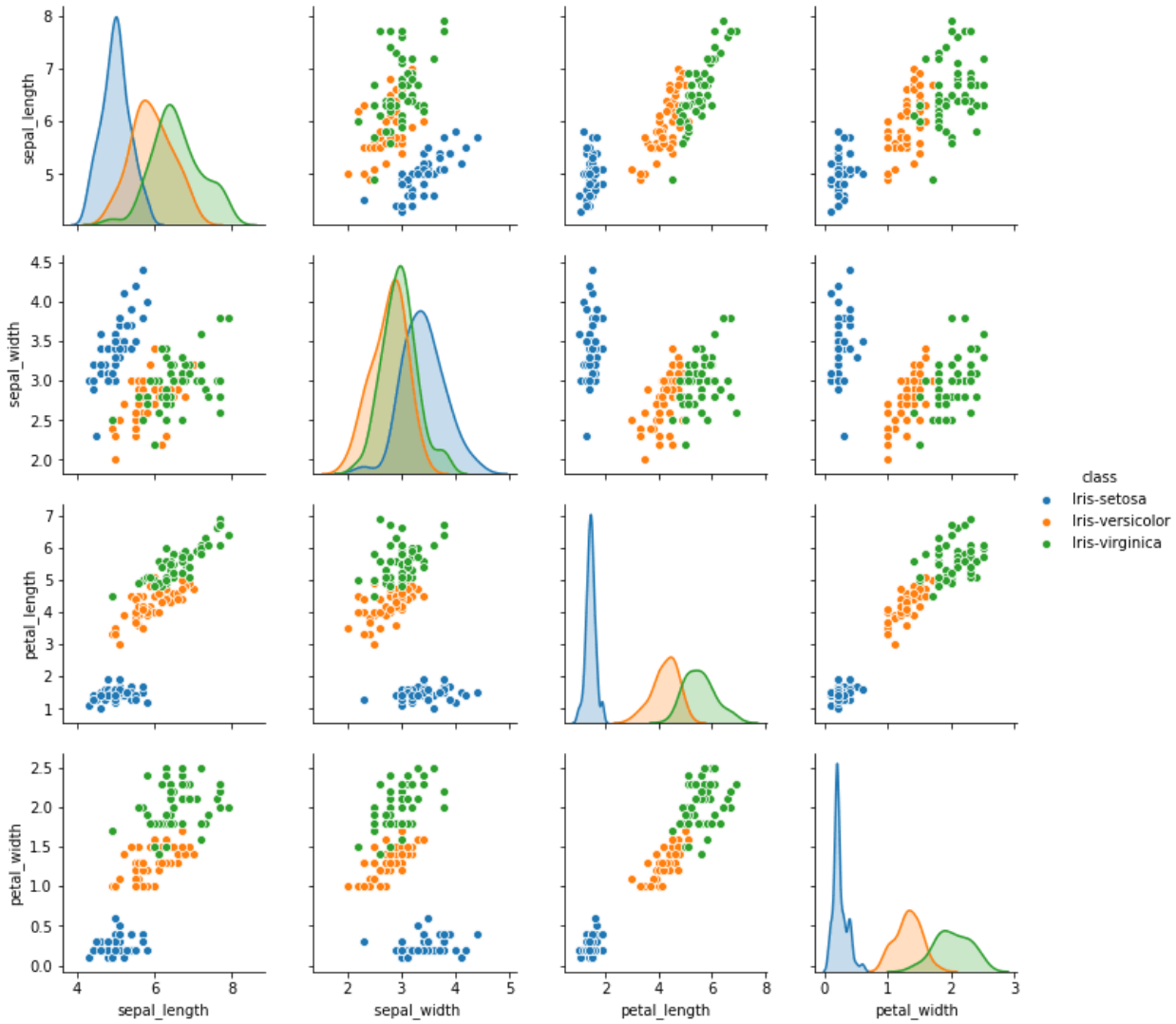
After loading the dataset into a dataframe in memory, the next step is to perform an exploratory data analysis. The objective of the EDA is to discover as much information as possible about the dataset. The describe() method is a good starting point. The describe() method prints statistics of the dataset, like mean, standard deviation, etc.

```
df.describe()
```

	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

A very important tool in exploratory data analysis is data visualization, which helps us to gain insights about the dataset. The plot below shows the relationship between the attributes of the dataset.

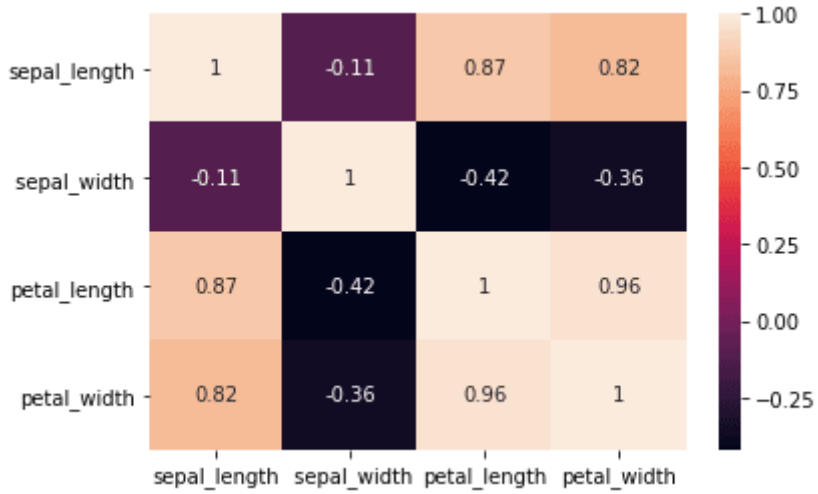
```
sns.pairplot(df, hue='class');
```



Joint distributions of Iris dataset attributes

Another interesting case of data visualization is use a heatmap to visualize the correlation matrix of the dataset.

```
sns.heatmap(df.corr(), annot=True)
```



Correlation matrix represented as an heatmap

4-Preprocess the data

Frequently, the dataset collected from databases, files or scraping the internet is not ready to be consumed by a machine learning algorithm. In most cases, the dataset needs some kind of preparation or preprocessing before being used as input to a machine learning algorithm. In this case, we convert the string values of the class column to integer numbers because the algorithm we are going to use does not process string values.

```
df['class_encod'] = df['class'].apply(lambda x: 0 if x == 'Iris-setosa' else 1 if x == 'Iris-versicolor' else 2)
df['class_encod'].unique()
```

5-Select an algorithm and train the model

After exploring and preprocessing our data we can build our machine learning model to classify Iris specimens. So, the first step is to split our dataframe in input attributes and target attributes.

```
y = df[['class_encod']] # target attributes
X = df.iloc[:, 0:4] # input attributes
X.head()
```

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

If in the previous step we splitted the dataframe by separating columns, in this step we split the data by rows. The method train_test_split() will split the X and y dataframes in training data and test data.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=0, stratify=y)

np.shape(y_train)
```

Then we use the datasets X_train and y_train to build a KNN classifier, using the KNeighborsClassifier class provided by scikit-learn. Because the machine learning algorithm is already implemented by the library, all we have to do is call the method fit() passing the X_train and y_train datasets as arguments.

```
m = KNeighborsClassifier()
m.fit(X_train, np.ravel(y_train))
```

Once the model is built, we can use the predict() method to calculate the predicted category of an instance. In this case, we want to predict the class of the first 10 lines of the X_test dataset. The return is an array containing the estimated categories.

```
m.predict(X_test.iloc[0:10])
```

```
array([2, 2, 0, 0, 1, 0, 1, 2, 0, 1])
```

We can use methods like score() and confusion_matrix() to measure the performance of our model. We see that the accuracy of our model is 1.0 (100%), which means that the model predicted correctly all cases of the test dataset.

```
m.score(X_test, y_test)
```

```
1.0
```

```
confusion_matrix(y_test, m.predict(X_test))
```

```
array([[15,  0,  0],
       [ 0, 15,  0],
       [ 0,  0, 15]])
```