



Iris flower classification

By Yasser Mamdouh
Machine learning
Intern
Intern2Grow

About me

Fresh graduate student enthusiasm about Machine Learning and passionate about Python programming seeking a challenge position to leverage my Skills.

Outlines

- What Iris flower
- Why we need an AI model to classify the Iris flower type
- About the Data used
- Outlier detection
- Data Exploration
- Model selection
- Best model evaluation
- Model deployment

What Iris Flower is?

If you're seeking a flower with a unique story behind it, the iris is the one to look for.

The iris was originally cultivated in 1749 BC, after King Thutmose III conquered Syria. This land was covered in irises.

Why do we need an AI model to classify the Iris flower type?

Iris, (genus Iris), genus of about 300 species of flowering plants, including some of the world's most popular and varied garden flowers.

Speed

AI models can classify Iris flowers quickly and efficiently, processing large amounts of data in a fraction of the time it would take a human.

Scalability: AI models can handle classification tasks at scale, making them suitable for applications where thousands or millions of samples need to be processed.

Accuracy and Consistency

Consistency: AI models provide consistent results, eliminating human errors and biases that can occur with manual classification.

Accuracy: With proper training and data, AI models can achieve high levels of accuracy in classification tasks, often surpassing human performance.



About the Data used!

Before going further into data overview, there is a handy of Python libraries are most important for ML that need to be installed, then imported. like:

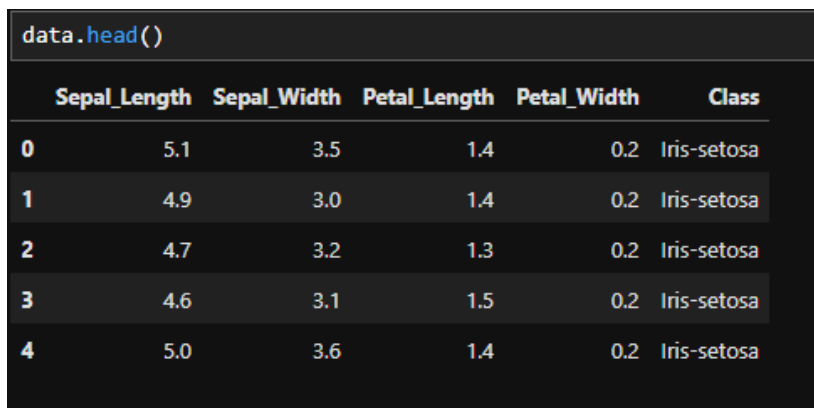
```
import pandas as pd, numpy as np, seaborn as sns, matplotlib.pyplot  
as plt  
%matplotlib inline
```

Loading the Iris Data

```
data = pd.read_csv('dataset.csv')
```

Data Overview

- data samples



	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

- Rows & Columns

```
data.shape  
(150, 5)
```

As the shape (pandas DataFrame attribute) show us, Our data has 150 rows and 5 columns.

Data cleaning and processing

- Adjust column names to lowercase for ease writing

```
data_col = data.columns  
data = data.set_axis(data_col.str.lower(), axis=1)
```

- Check for any missing values

```
data.isna().sum()
```

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
class           0
dtype: int64
```

- Check for any duplicate

```
duplicated_indices = data[data.duplicated()]
duplicated_indices
```

	sepal_length	sepal_width	petal_length	petal_width	class
34	4.9	3.1	1.5	0.1	Iris-setosa
37	4.9	3.1	1.5	0.1	Iris-setosa
142	5.8	2.7	5.1	1.9	Iris-virginica

- Remove duplicated_indices

```
data.drop_duplicates(inplace=True)
```

- Check duplicates removed successfully

```
data.duplicated().sum()
0
```

Data explorations

Is a process in which we get more insights about the data through drawing graphs and charts using Matplotlib and Seaborn (Python datascience libraries)



Features and target

Note: It's best to not doing any transformation for the target variable, so we separate the target variable ("class" in our iris data) from the rest features.

```
y = data['class']
x = data.drop('class', axis=1)
```

- Summary statistics for Iris Flower Specifics

The describe() function is a DataFrame method used to know about mean, max, min, and standard deviation for each column.

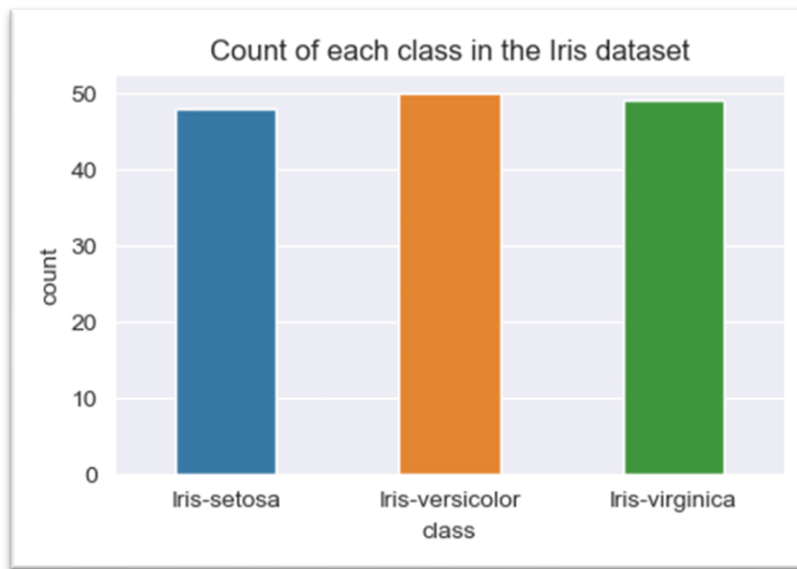
```
x.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	147.000000	147.000000	147.000000	147.000000
mean	5.856463	3.055782	3.780272	1.208844
std	0.829100	0.437009	1.759111	0.757874
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.400000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

- CountPlot for y axis (class target)

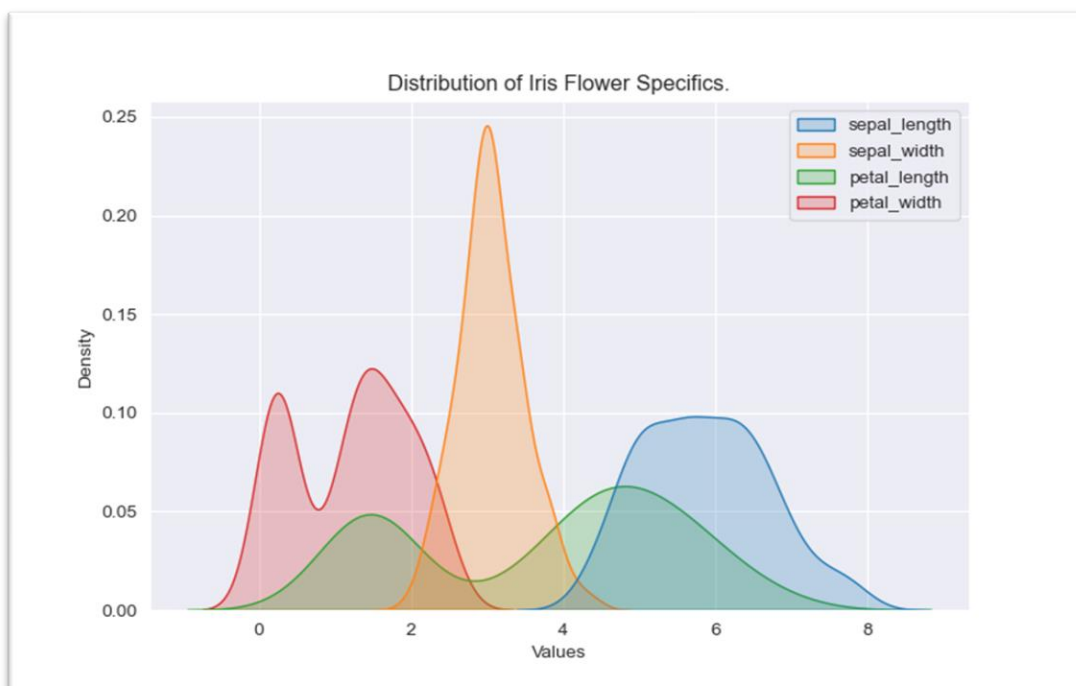
It gives us a bar for each value represent how many samples are found for the same category

```
plt.figure(figsize=(5, 4))
sns.set_style('darkgrid')
sns.countplot(x='class', data=data, width=0.45)
plt.xlabel("class")
plt.ylabel("count")
plt.title('Count of each class in the Iris dataset')
plt.show()
```



- Check feature distributions

The KDE-Plot is a most common one to see how your data distribution's shape like Normal (Bell-shaped) or Non-normal, also to know if your data is unimodal or multimodal, and to know about if there is a Skewness or not.



As the we saw in the previous figure, the data contains a multimodal features and may have a slight right Skewness.

Outlier detection

Outliers are data points that significantly different from other observations in the dataset, it may distract the analysis, especially in skewed distributions.

ZScore

Is a statistical measure to detect outliers using threshold (standardized one = 3). It describes a value's position relative to the mean of a group of values.

- Positive z-score: Indicates the value is above the mean.
- Negative z-score: Indicates the value is below the mean.
- Z-score of 0: Indicates the value is exactly at the mean.
- Magnitude of the z-score: Indicates how many standard deviations the value is away from the mean.

So Values with a z-score less than -3 or greater than +3 are considered outliers.

```
Outliers:
sepal_length  sepal_width  petal_length  petal_width  class
15           5.7         4.4         1.5         0.4  Iris-setosa
```


Model selection

Splitting the data into train and test

It's a good to check the model performance using unseen data, also it's a way to check for Overfitting and Underfitting through comparing the train with test accuracy.

The following are rows&columns for the train and test data after doing split.

```
Shape of x_train = (116, 4), and x_test = (30, 4)
Shape of y_train = (116,), and y_test = (30,)
```

Model training

To prepare a diverse set of machine learning models for evaluation and comparison in the project.

I implemented a function, **get_based_models**, which initializes and returns a list of different machine learning models. This function ensures that a variety of algorithms and configurations are considered for model selection.

The purpose of the previous function is to choose the best model for further Hupertuning process.

5-Fold Cross Validation

To evaluate the performance of various machine learning models using cross-validation.

- The `get_based_models` function is called to get a list of predefined models.
- A 5-fold cross-validation is used, where the dataset is split into 5 subsets. Each subset is used as a test set once while the remaining 4 subsets form the training set. This process is repeated 5 times (folds).

- The mean accuracy and standard deviation of the accuracy across the folds are calculated using np.mean and np.std, respectively.
- Results are appended to the results list and model names to the names list.

```
LDA: Mean Accuracy: 0.9743 (Std Dev: 0.0210)
KNN_5: Mean Accuracy: 0.9656 (Std Dev: 0.0506)
KNN_7: Mean Accuracy: 0.9656 (Std Dev: 0.0506)
KNN_9: Mean Accuracy: 0.9659 (Std Dev: 0.0417)
KNN_11: Mean Accuracy: 0.9656 (Std Dev: 0.0324)
DT_gini: Mean Accuracy: 0.9572 (Std Dev: 0.0264)
DT_entropy: Mean Accuracy: 0.9659 (Std Dev: 0.0314)
NB: Mean Accuracy: 0.9572 (Std Dev: 0.0381)
SVM_Linear: Mean Accuracy: 0.9830 (Std Dev: 0.0209)
SVM_RBF: Mean Accuracy: 0.9743 (Std Dev: 0.0346)
SVM_Sigmoid: Mean Accuracy: 0.2500 (Std Dev: 0.0638)
SVM_Poly: Mean Accuracy: 0.9486 (Std Dev: 0.0318)
AdaBoost: Mean Accuracy: 0.9486 (Std Dev: 0.0318)
GradientBoosting: Mean Accuracy: 0.9572 (Std Dev: 0.0264)
GradientBoosting: Mean Accuracy: 0.9572 (Std Dev: 0.0264)
RF_Entropy_100: Mean Accuracy: 0.9659 (Std Dev: 0.0417)
RF_Entropy_500: Mean Accuracy: 0.9572 (Std Dev: 0.0381)
RF_Gini_100: Mean Accuracy: 0.9572 (Std Dev: 0.0381)
RF_Gini_500: Mean Accuracy: 0.9572 (Std Dev: 0.0381)
ExtraTrees_100: Mean Accuracy: 0.9489 (Std Dev: 0.0490)
ExtraTrees_500: Mean Accuracy: 0.9489 (Std Dev: 0.0490)
ExtraTrees_1000: Mean Accuracy: 0.9489 (Std Dev: 0.0490)
```



Best model evaluation

Evaluating the best ones

It's a process to measure the model performance with different metrics, also to check Overfitting and understanding through comparing the training set and testing set accuracy.

- LDA: Mean Accuracy: 0.9743 (Std Dev: 0.0210)
- KNN_9: Mean Accuracy: 0.9659 (Std Dev: 0.0417)

- SVM_Linear: Mean Accuracy: 0.9830 (Std Dev: 0.0209)
- SVM_RBF: Mean Accuracy: 0.9743 (Std Dev: 0.0346)

Linear Discriminant Analysis

- accuracy (train set): 0.97 %
- accuracy (test set): 1.00 %



KNearest Neighbors

- accuracy (train set): 0.97 %
- accuracy (test set): 1.00 %

Support Vector Machine (linear kernel)

- accuracy (train set): 0.98 %
- accuracy (test set): 1.00 %

Support Vector Machine (RBF kernel)

- accuracy (train set): 0.98 %
- accuracy (test set): 0.97 %

As a result of evaluation for the best models, the best one with a well performance without a significant difference between a training set and testing set is SVC-rbf with a train accuracy 98% and test accuracy 97%.

Hyperparameter tuning for SVC-rbf

```
Best parameters found: {'C': 0.7, 'kernel': 'linear'}  
accuracy (train set): 0.98 %  
accuracy (test set): 1.00 %
```

But as it show us, there may be a underfitting as long as the test set accuracy is (slightly) greater than the train set.





SVC (poly kernel)

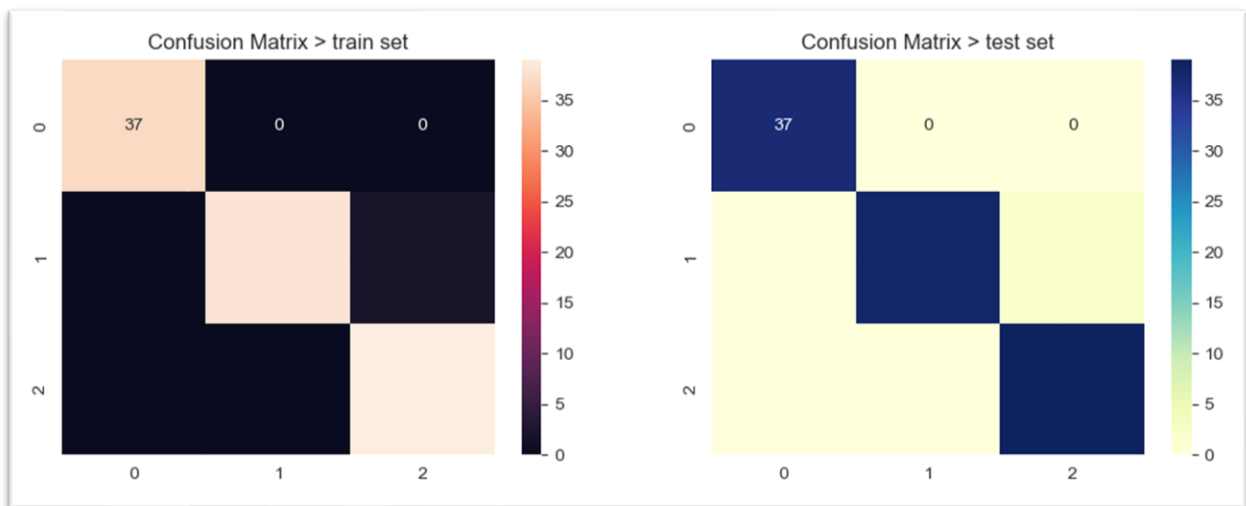
It's my last effort to get a model with high performance, and guess What!!.

Mission is complete. 🎉👉

The following are the choosed paramnters:

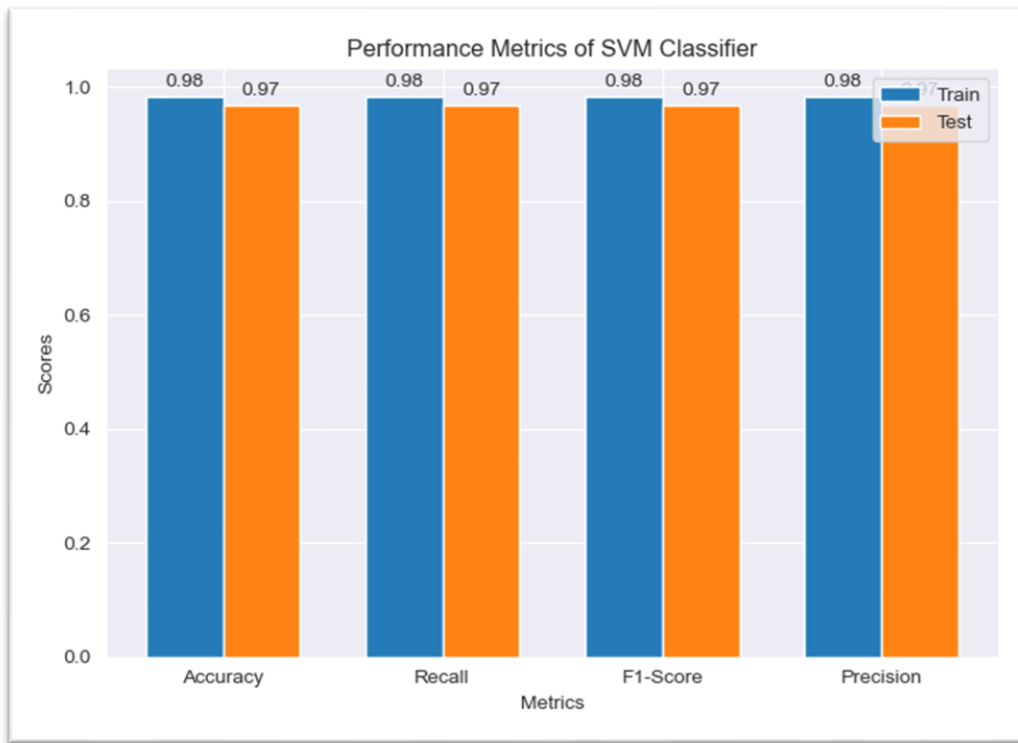
```
(kernel='poly', C=0.3, random_state=42, class_weight='balanced')
```

And here is the confusion matrix for both training set and test set:





And here is the most common preformance measures:



Model deployment

Is a final step for any project is deploying your work to be user-friendly, like edge devices or what ever your desired manner.

Model saving

It's a first sub-process for model deployment in our field is to save the model, and here I used a joblib is a Python library.

```
['files\\iris_classifier_svc_poly.pkl']
```



Deployment

I created a function to take user input for the features of an iris flower, create a DataFrame from those features, and then use a pre-trained model (the model I saved in the previous step) to predict the class of the iris flower.

```
The Iris Sepal length: 4.3
The Iris Sepal width: 2
The Iris Petal length: 1.5
The Iris Petal width: 6
It's Iris-virginica.
```

Resources



- <https://www.britannica.com/plant/Iris-plant-genus>
- <https://www.floraly.com.au/blogs/news/the-iris-flower-meanings-images-insights>
- <https://scikit-learn.org/stable/index.html>

