Week 4: Deployment on Flask

Name: Farha Jabin Oyshee

Batch Code: LISUM13

Date: 27 October 2022

Submitted to: Data Glacier

1. Introduction

In this project, we are going to deploying machine learning model using the Flask Framework.
As a demonstration, our model help to predict Iris Species.

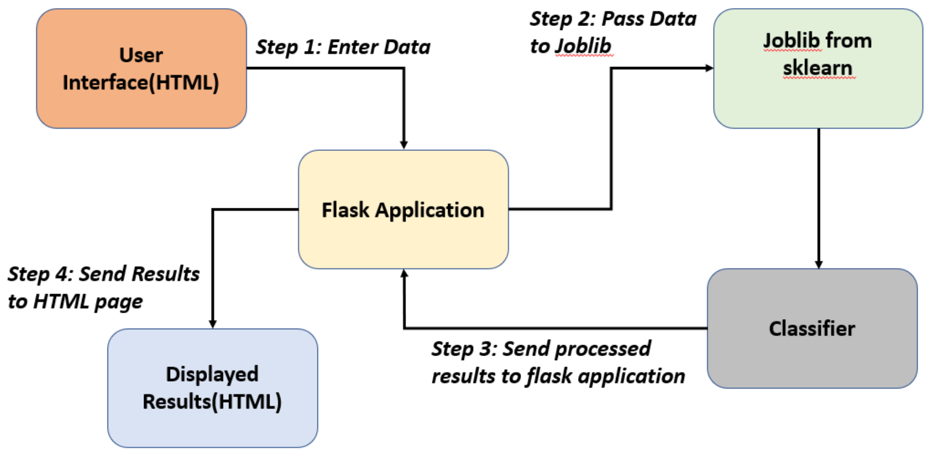


Figure 1.1: Application Workflow

We will focus on both: building a machine learning model to predict Iris Species, then create an API for the model, using Flask, the Python micro-framework for building web applications. This API allows us to utilize predictive capabilities through HTTP requests.

2. Data Information

The samples were extracted from the number of samples based on sepal and petal's length and width in each species and the total number of samples per dataset.

Table 2.1: Dataset Information

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa

3.1.1 Data Import & Pre-processing

- Supervised ML with Iris Dataset

In [1]:
```
# EDA packages
import pandas as pd
import numpy as np
```

In [3]:
```
# Plotting Packages
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# ML Packages

from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
```

3.1.2 EDA Descriptive Analysis

EDA Descriptive

In [4]:
```
# Load our dataset
df = pd.read_csv("iris.csv")
```

In [5]:
```
df.head()
```

Out[5]:

	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

In [6]:
```
df.describe()
```

Out[6]:

	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

In [8]:
```
# Check for missing values
df.isna().sum()
```

Out[8]:
```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

In [10]:
```
df.shape
```

Out[10]:
```
(150, 5)
```

In [11]:
```
# Species distribution
print(df.groupby('species').size())
```

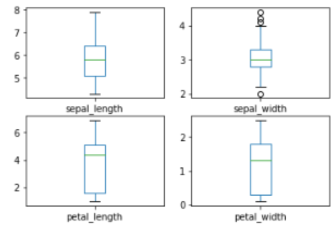
```
species
setosa        50
versicolor    50
virginica     50
dtype: int64
```

3.1.3 Data Visualization

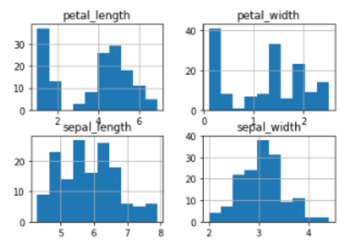
Data Visualization

- Understand each attribute
- Understand relationship between each

In [13]:
```
df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()
```

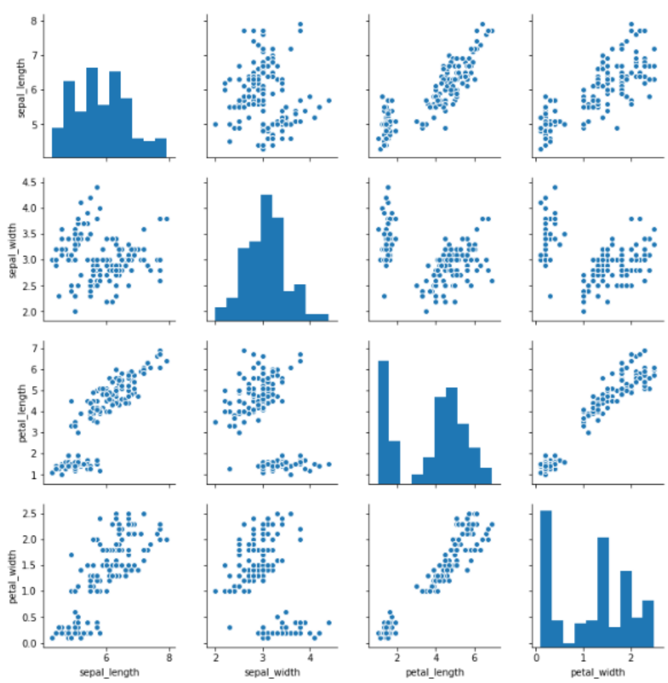


In [14]:
```
# histograms using pandas plot
df.hist()
plt.show()
```

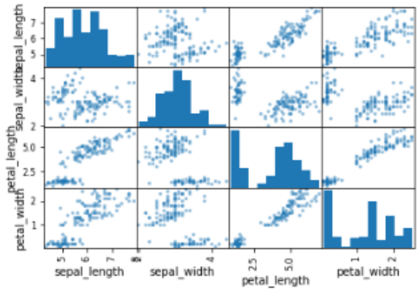


In [15]:
```
# Multivariate Plots
# Relationships between each attribute
sns.pairplot(df)
```

Out[15]: `<seaborn.axisgrid.PairGrid at 0x7f53f30d6278>`



In [18]:
```
# scatter plot matrix
from pandas.plotting import scatter_matrix
scatter_matrix(df)
plt.show()
```



In [19]:
```
### ML
```

In [20]:
```
# Split-out validation dataset
array = df.values
X = array[:,0:4]
Y = array[:,4]
```

3.1.4 Persisting the Model

Saving or Persisting Our Model

- Pickle
- Joblib

In [36]:
```
from sklearn.externals import joblib
joblib.dump(logit, 'logit_model_iris.pkl')
```

Out[36]:
```
['logit_model_iris.pkl']
```

In [38]:
```
# Reloading the Model
logit_model = joblib.load('logit_model_iris.pkl')
```

In [39]:
```
df.tail()
```

Out[39]:

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

In [40]:
```
ex2 = np.array([6.2,3.4,5.4,2.3]).reshape(1,-1)
```

In [41]:
```
logit_model.predict(ex2)
```

Out[41]:
```
array(['virginica'], dtype=object)
```

```
In [ ]: ### Get the Models for the other ML Algorithms
```

```
In [42]: from sklearn.tree import DecisionTreeClassifier
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
         from sklearn.naive_bayes import GaussianNB
         from sklearn.svm import SVC
```

```
In [43]: knn = KNeighborsClassifier()
         dtree = DecisionTreeClassifier()
         svm = SVC()
```

```
In [50]: # Fit the model
         knn.fit(X_train, Y_train)
         print("accuracy :" , knn.score(X_validation,Y_validation))

         accuracy : 0.9
```

```
In [51]: # save the model to disk
         joblib.dump(knn, 'knn_model_iris.pkl')

Out[51]: ['knn_model_iris.pkl']
```

```
In [48]: dtree.fit(X_train, Y_train)
         print("accuracy :" , dtree.score(X_validation,Y_validation))

         accuracy : 0.9
```

```
In [49]: # save the model to disk
         joblib.dump(dtree, 'dtree_model_iris.pkl')

Out[49]: ['dtree_model_iris.pkl']
```

```
In [52]: svm.fit(X_train, Y_train)
         print("accuracy :" , svm.score(X_validation,Y_validation))

         accuracy : 0.9333333333333333
```

```
In [53]: # save the model to disk
         joblib.dump(svm, 'svm_model_iris.pkl')

Out[53]: ['svm_model_iris.pkl']
```

4. Display the Model into Web Application

We develop a web application that consists of a simple web page with a form field that lets us toggle the length and width of sepal-petal. After submitting the necessary input in the web application, it will render it which gives us the results.

First, we create a folder for this project, this is the directory tree inside the folder. We will explain each file.

Table 4.1: Application Folder File Directory

```
app.py
templates/
        index.html
        preview.html
static/
    imgs/
        iris_setosa.jpg
        iris_versicolor.jpg
        iris_virginica.jpg
      styles.css

data/
    dtree_model_iris.pkl
    finalized_model.sav
    iris.xlxs
    knn_model_iris.pkl
    logit_model_iris.pkl
    svm_model_iris.pkl
    ML -Supervised Learning with Iris Dataset.ipyb
```

The sub-directory templates are the directory in which Flask will look for static HTML files for rendering in the web browser, in our case, we have two HTML files: *index.html* and *preview.html*.

4.1 App.py

The *app.py* file contains the main code that will be executed by the Python interpreter to run the Flask web application, it included the ML code for classifying SD.

```python
from flask import Flask,render_template,url_for,request
from flask_material import Material

# EDA PKg
import pandas as pd
import numpy as np

# ML Pkg
from sklearn.externals import joblib


app = Flask(__name__)
Material(app)

@app.route('/')
def index():
    return render_template("index.html")

@app.route('/preview')
def preview():
    df = pd.read_csv("data/iris.csv")
    return render_template("preview.html",df_view = df)

@app.route('/',methods=["POST"])
def analyze():
    if request.method == 'POST':
        petal_length = request.form['petal_length']
        sepal_length = request.form['sepal_length']
        petal_width = request.form['petal_width']
        sepal_width = request.form['sepal_width']
        model_choice = request.form['model_choice']

        # Clean the data by convert from unicode to float
        sample_data = [sepal_length,sepal_width,petal_length,petal_width]
        clean_data = [float(i) for i in sample_data]
```

```python
        # Clean the data by convert from unicode to float
        sample_data = [sepal_length,sepal_width,petal_length,petal_width]
        clean_data = [float(i) for i in sample_data]

        # Reshape the Data as a Sample not Individual Features
        ex1 = np.array(clean_data).reshape(1,-1)

        # ex1 = np.array([6.2,3.4,5.4,2.3]).reshape(1,-1)

        # Reloading the Model
        if model_choice == 'logitmodel':
            logit_model = joblib.load('data/logit_model_iris.pkl')
            result_prediction = logit_model.predict(ex1)
        elif model_choice == 'knnmodel':
            knn_model = joblib.load('data/knn_model_iris.pkl')
            result_prediction = knn_model.predict(ex1)
        elif model_choice == 'svmmodel':
            knn_model = joblib.load('data/svm_model_iris.pkl')
            result_prediction = knn_model.predict(ex1)

    return render_template('index.html', petal_width=petal_width,
        sepal_width=sepal_width,
        sepal_length=sepal_length,
        petal_length=petal_length,
        clean_data=clean_data,
        result_prediction=result_prediction,
        model_selected=model_choice)


if __name__ == '__main__':
    app.run(debug=True)
```

Figure 3.1: App.py

- We ran our application as a single module; thus, we initialized a new Flask instance with the argument __name__ to let Flask know that it can find the HTML template folder (*templates*) in the same directory where it is located.
- Next, we used the route decorator (*@app.route('/'))* to specify the URL that should trigger the execution of the home function.
- Our *home* function simply rendered the *index.html* HTML file, which is located in the *templates* folder.
- Inside the *predict* function, we access the spam data set, pre-process the text, and make predictions, then store the model. We access the new message entered by the user and use our model to make a prediction for its label.
- we used the *POST* method to transport the form data to the server in the message body. Finally, by setting the *debug=True* argument inside the app.run method, we further activated Flask's debugger.
- Lastly, we used the *run* function to only run the application on the server when this script is directly executed by the Python interpreter, which we ensured using the *if* statement with __name__ == '__main__'.

4.2 index.html

The following are the contents of the *home.html* file that will render a text form where a user can enter a message.

```
{% extends "material/base.html" %}
{% block content %}
<div class="showcase container purple lighten-3">
    <div class="row">
        <div class="col 12 m10 offset-ml center">
            <h2>Iris Species Predictor </h2>
            <p>ML Web App</p>
            <a href="{{url_for('index')}}" class="btn btn-small purple white-text waves-effect waves-dark">Reset</a>
            <a href="{{url_for('preview')}}" class="btn btn-small white purple-text waves-effect waves-dark">View Dataset</a>
        </div>

    </div>
</div>
<section class="section section-signup">
        <div class="container">
          <div class="row">

            <div class="col s12 m4">
              <div class="card-panel grey lighten-4 grey-text text-darken-4 z-depth-0">
                <form action="{{ url_for('analyze')}}" method="POST">
                  <div class="input-field">
                   <p class="range-field">
                        <input type="range" id="sepal_lengthInput" name="sepal_length" min="4" max="8" value="0" step="0.1" >
                    <label for="Sepal Length">Sepal Length</label>

                  </div>
                   <div class="input-field">
                   <p class="range-field">
                        <input type="range" id="sepal_widthInput" name="sepal_width" min="2" max="5" value="0" step="0.1">
                    <label for="">Sepal Width</label>
                  </div>
                   <div class="input-field">
                   <p class="range-field">
                        <input type="range" id="petal_lengthInput" name="petal_length" min="0" max="7" value="0"
                   step="0.1" >
                    <label for="">Petal Length</label>
```

```html
</div>
 <div class="input-field">
 <p class="range-field">
     <input type="range" id="petal_lengthInput" name="petal_length" min="0" max="7" value="0"
 step="0.1" >
  <label for="">Petal Length</label>
</div>
<div class="input-field">
 <p class="range-field">
     <input type="range" id="petal_widthInput" name="petal_width" min="0" max="3" value="0"
step="0.1">
  <label for="">Petal Width</label>
</div>

<div class="input-field">
  <select id="role" name="model_choice">
    <option value="" disabled selected>Select Model</option>
    <option value="logitmodel">Logistic Regression</option>
      <option value="knnmodel">K-Nearest Neighbour</option>
      <option value="svmmodel">SVM</option>
  </select>
  <label for="role">Select ML Algorithm</label>
</div>
<input type="submit" value="Predict" class="btn btn-small purple waves-effect waves-light btn-extend">
<input type="reset" value="Clear" class="btn btn-small white waves-effect waves-light btn-extend">
</form>
</div>
</div>
<div class="col s12 m4 offers">
    <div class="card-panel purple lighten-4 grey-text text-darken-4 z-depth-0">

      <p>Sepal Length: {{ sepal_length }}</p>
<p>Sepal Width: {{ sepal_width }}</p>
<p>Petal Length: {{ petal_length}} </p>
<p>Petal Width: {{ petal_width}}</p>
Using {{ model_selected }} on {{ clean_data }}
```

```html
</div>
    </div>



<div class="col s12 m4 offers">
  <h5>Prediction</h5>
    <div class="collection" role="alert">
      <p  class="collection-item active purple">Predicted result {{ result_prediction }} </p>
  </div>
  <div class="card-image waves-effect waves-block waves-light">

      {% if result_prediction == ['versicolor'] %}
        <img src="static/imgs/iris_versicolor.jpg" width="200px" height="200px">

        {% elif result_prediction == ['setosa']  %}
          <img src="static/imgs/iris_setosa.jpg" width="200px" height="200px">

        {% elif result_prediction == ['virginica']  %}
          <img src="static/imgs/iris_virginica.jpg"width="200px" height="200px" >

        {% else %}
            <p></p>


      {% endif%}
  </div>

</div>

</div>
</div>
</div>
</section>
```

Figure 4.2: index.html

4.1.1 preview.html

we create a preview.html file that will be rendered via the *render_template('preview.html', prediction=my_prediction)* line return inside the *predict* function, which we defined in the *app.py* script to display the text that a user-submitted via the text field.

From *preview.html* we can see that some code using syntax not normally found in HTML files: *{% if prediction ==1%},{% elif prediction == 0%},{% endif %}*This is Jinja syntax, and it is used to access the prediction returned from our HTTP request within the HTML file.

```
{% extends "material/base.html" %}
{% block content %}
<div class="showcase container purple lighten-3">
    <div class="row">
        <div class="col 12 m10 offset-ml center">
            <h2>Iris Species Predictor </h2>
            <p>ML Web App</p>
            <a href="{{url_for('index')}}" class="btn btn-small purple white-text waves-effect waves-dark">Back</a>

        </div>

    </div>
</div>

<div class="container">
        {{ df_view.to_html(classes="table striped",na_rep="-") | safe}}

</div>






{% endblock %}


{% block scripts %}
{{ super() }}




{% endblock %}
```

Figure 3.3: Result.html