INTELLIGENT SYSTEMS

(Part Three – Classification Model Building, Classification Web Services)

Version: 0.1 (20181104T1436)

Author:

Ms Juan Antonio Castro Silva (juan.castro@usco.edu.co)

PhD Diego Hernán Peluffo Ordoñez

1. Introduction:

In this third part of the course (tutorial), we are going to create two web services, one to build the classification model, and other to make classifications using the saved model (consume). To create the model, we will use the iris dataset, the Flask Microframework to build the Web Services and Joblib to save the model. To create the enterprise application, we will use the Java programming language and the WildFly web application server.

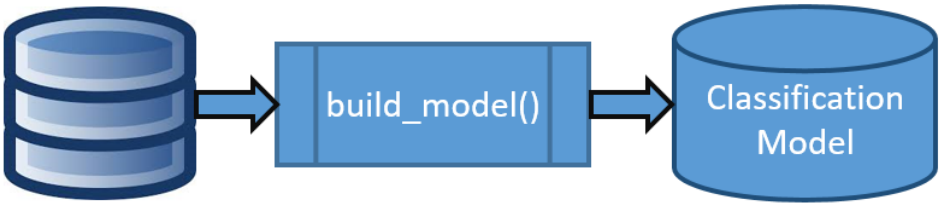


Diagram 1: Classification model building.

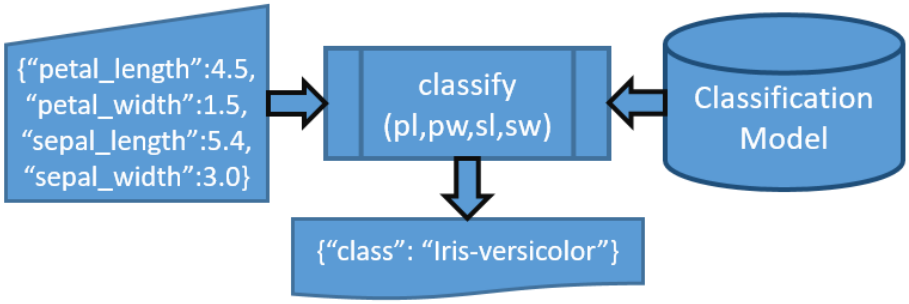


Diagram 2: Flower classification.

2. Machine Learning

In this age of modern technology, there is one resource that we have in abundance: a large amount of structured and unstructured data. In the second half of the twentieth century, machine learning evolved as a subfield of *artificial intelligence* that involved the development of self-learning algorithms to gain knowledge from that data in order to make predictions. Instead of requiring humans to manually derive rules and build models from analyzing large amounts

of data, machine learning offers a more efficient alternative for capturing the knowledge in data to gradually improve the performance of predictive models, and make data-driven decisions [1].

There are two main types of machine learning: *supervised learning* and *unsupervised learning*.

2.1 Supervised learning

The main goal in supervised learning is to learn a model from labeled *training* data that allows us to make predictions about unseen or future data. Here, the term *supervised* refers to a set of samples where the desired output signals (labels) are already known.

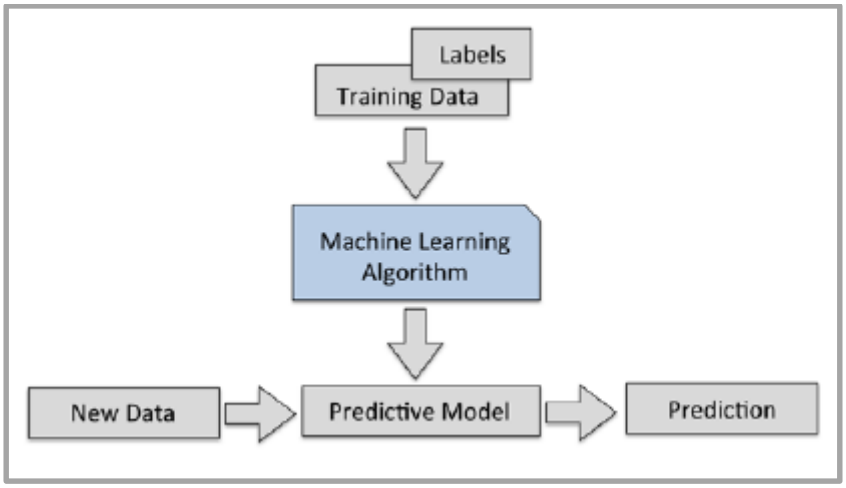
2.2 Unsupervised learning

In unsupervised learning, however, we are dealing with unlabeled data or data of *unknown structure*. Using unsupervised learning techniques, we are able to explore the structure of our data to extract meaningful information without the guidance of a known outcome variable or reward function.

Iris flowers classification problem

In this tutorial, you will be introduced to classification problems and learn how to solve them using supervised learning techniques. Our task is to identify to which category an object (flower-specie) belongs to.

A typical example of a *multi-class classification* (multi-nominal) task is flower classification. Here, we could collect a training dataset (Iris) that consists of multiple examples of each category. Now, if a user provides a new unlabeled instance, via an input device, our predictive model will be able to predict the correct category in the set of class labels with certain accuracy.

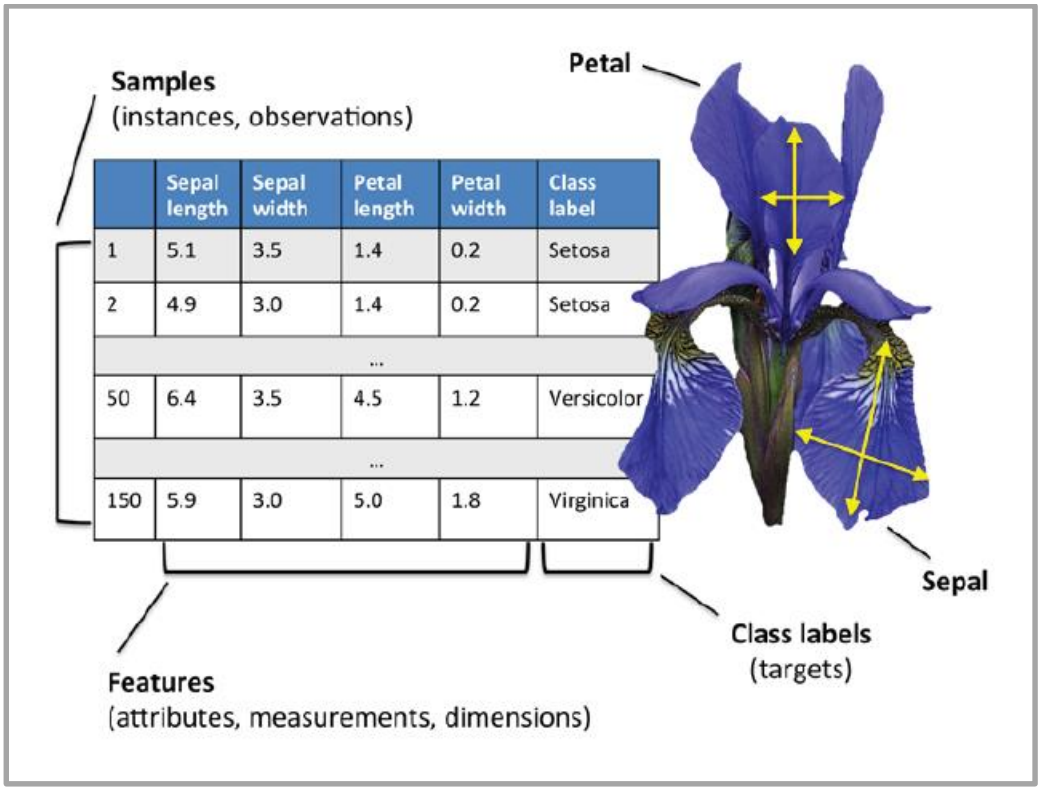


Source: Python Machine Learning [1].

Classification is a subcategory of supervised learning where the goal is to predict the categorical class labels of new instances based on past observations. Those class labels are discrete, unordered values that can be understood as the *group memberships* of the instances.

Considering the example of flower classification, we can train a model using a supervised machine learning algorithm on a dataset of labeled flowers, specie that are correctly marked as Virginica, Setosa or Versicolor to predict whether a new specie belongs to either of the three categories. A supervised learning task with discrete *class labels* is also called a *classification* task.

The following table depicts an excerpt of the *Iris* dataset, which is a classic example in the field of machine learning. The Iris dataset contains the measurements of 150 iris flowers from three different species: *Setosa*, *Versicolor*, and *Virginica*. Here, each flower sample represents one row in our data set, and the flower measurements in centimeters are stored as columns, which we also call the features of the dataset:



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

Source: Python Machine Learning [1].

Iris dataset attribute information [2]:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
    - Iris Setosa
    - Iris Versicolor
    - Iris Virginica.

To complete your first machine learning project using Python visit this URL:

https://machinelearningmastery.com/machine-learning-in-python-step-by-step/

3. Software requirements:

Bellow, there is the list of the required software to build and run the Intelligent Web Application.

- Python 3.5 +
- Spyder IDE (Python)
- Flask Microframework 1.0.2
- Joblib
- Flask CORS
- Scikit-Learn
- MongoDB (NoSQL)
- PostgreSQL (Object-Relational Database)
- Java Development Kit (JDK)
- WildFly
- HttpComponents (HttpClient)

3.1 Software Installation

Apache HttpComponents

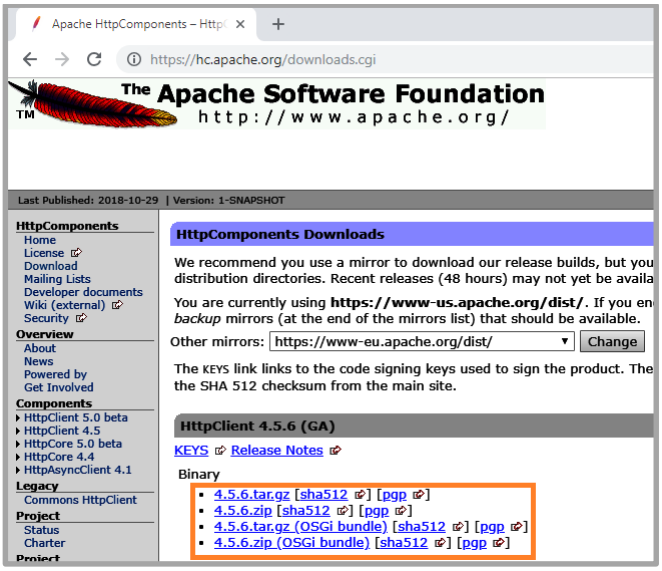
This project functions under the Apache Software Foundation (http://www.apache.org), and is part of a larger community of developers and users. The Apache HttpComponents™ project is responsible for creating and maintaining a toolset of low level Java components focused on HTTP and associated protocols.

The Hyper-Text Transfer Protocol (HTTP) is perhaps the most significant protocol used on the Internet today. Web services, network-enabled appliances and the growth of network computing continue to expand the role of the HTTP protocol beyond user-driven web browsers, while increasing the number of applications that require HTTP support.

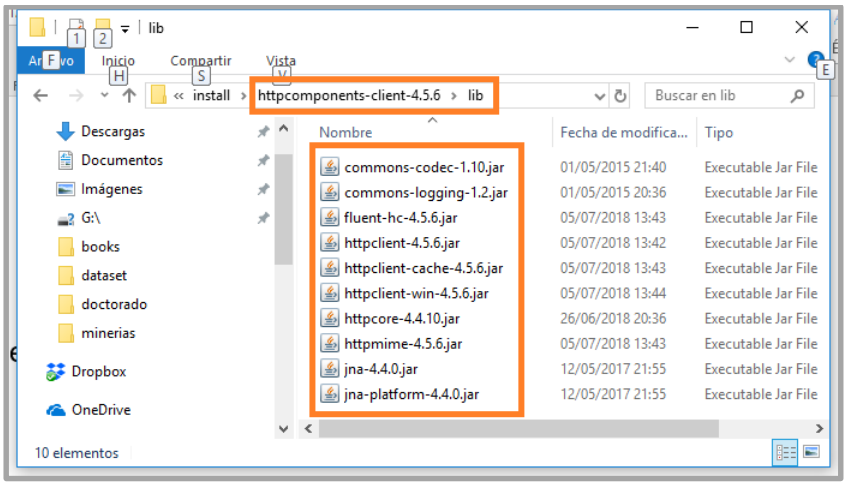
HttpComponents Client (HttpClient) is a HTTP/1.1 compliant HTTP agent implementation based on HttpCore. It also provides reusable components for client-side authentication, HTTP state management, and HTTP connection management. HttpComponents Client is a successor of and replacement for Commons HttpClient 3.x. Users of Commons HttpClient are strongly encouraged to upgrade. [3], [4].

Download the HttpClient 4.5.6.zip file.

https://hc.apache.org/downloads.cgi



Unzip the 4.5.6.zip file.



Flask-CORS

A Flask extension for handling Cross Origin Resource Sharing (CORS), making cross-origin AJAX possible. This extension led you to access a python web service from other server domain [5].

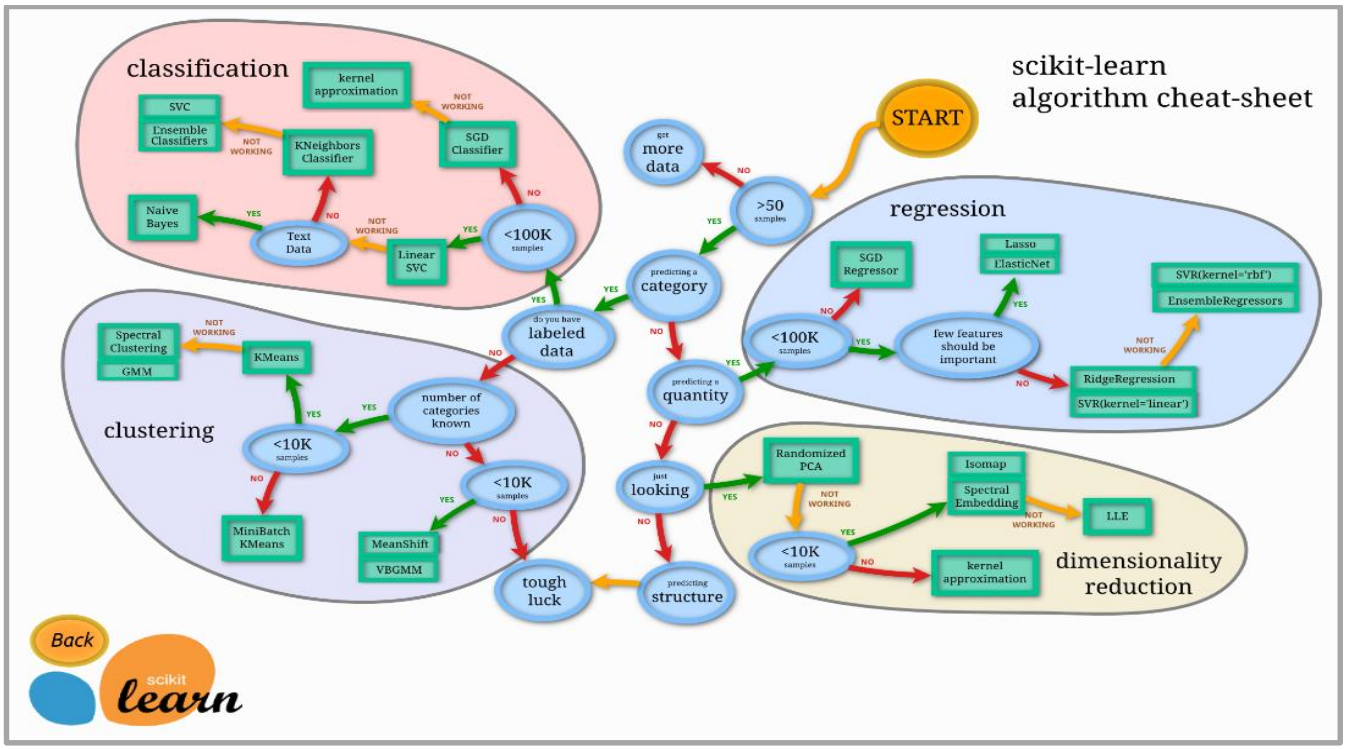
Install the extension with using pip, or easy_install.

```
pip install -U flask-cors
```

Scikit-Learn (Machine Learning in Python)

scikit-learn is a Python module for machine learning built on top of SciPy and distributed under the 3-Clause BSD license.

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license [6].



Source: http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

If you already have a working installation of numpy and SciPy, the easiest way to install scikit-learn is using pip.

```
pip install -U scikit-learn
```

Joblib (Model Persistence)

After training a scikit-learn model, it is desirable to have a way to persist the model for future use without having to retrain [7].

Joblib is a set of tools to provide lightweight pipelining in Python. In particular:

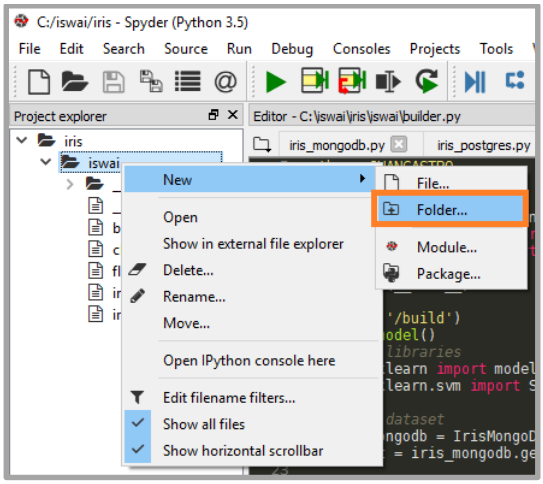
1. transparent disk-caching of functions and lazy re-evaluation (memorize pattern)
2. easy simple parallel computing

Joblib is optimized to be fast and robust in particular on large data and has specific optimizations for numpy arrays. It is BSD-licensed [8].

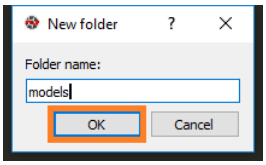
4. Machine Learning Web Services

Storage folder of models

Create a folder called models to store the classification models, right click the iswai package, select the New menu, and click the Folder… option.

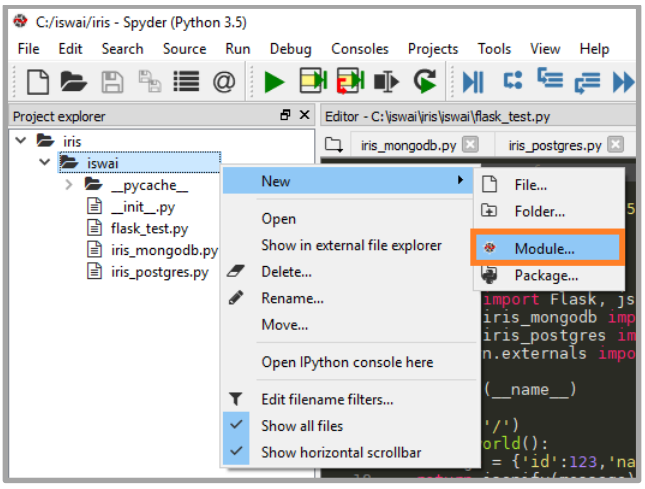


Write the folder name (models) and click the OK button.

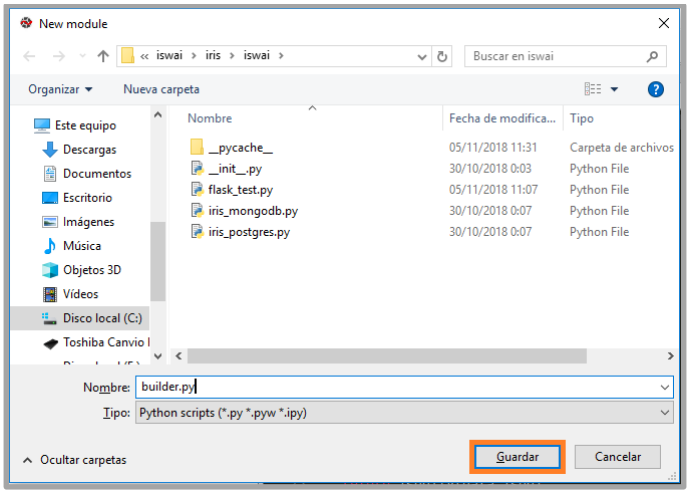


Web Services

Create two modules called builder.py and classifier.py to implement the machine learning web services. To create a new module right click the iswai package, select the New menu, and click the Module… option.



Write the module name (builder.py) and click the Save button.



File: builder.py

```
001 from flask import Flask, jsonify
002 from iswai.iris_mongodb import IrisMongoDB
003 from sklearn import model_selection
004 from sklearn.svm import SVC
005 from sklearn.externals import joblib
006
007 app = Flask(__name__)
008
009 @app.route('/build')
010 def build_model()
011     # Load dataset
012     iris_mongodb = IrisMongoDB()
013     dataset = iris_mongodb.getDataframe()
014
015     # Split-out validation dataset
016     array = dataset.values
017     X = array[:,1:5]
018     Y = array[:,0]
019     validation_size = 0.20
020     seed = 7
021     X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(
022             X, Y, test_size=validation_size, random_state=seed)
023
024     # Make predictions on validation dataset
025     model = SVC()
026     model.fit(X_train, Y_train)
027
028     # save the iris classification model
029     joblib.dump(model, 'models/iris_svc.model')
030
031     return jsonify({'response':'Iris SVC model saved'})
032
033 if __name__ == '__main__':
034    app.run()
035
```

Create a New Module, save it as classifier.py

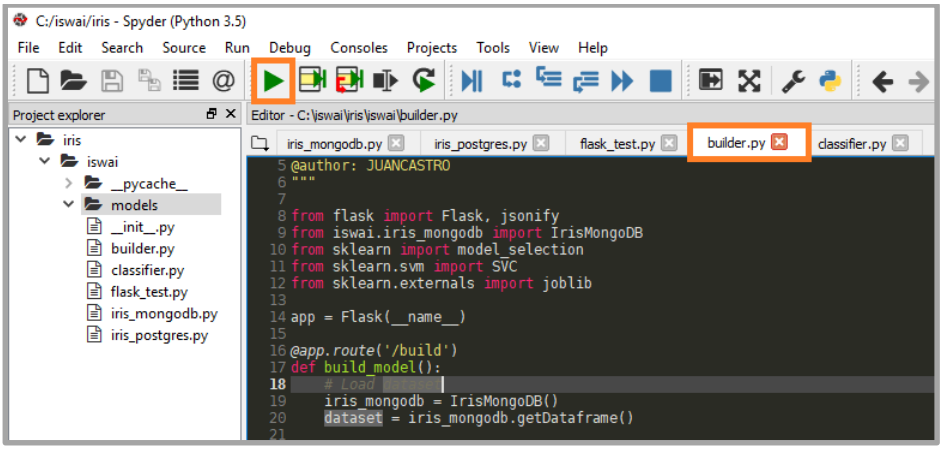
File: classifier.py

```
001 from flask import Flask, jsonify
002 from sklearn.externals import joblib
003
004 app = Flask(__name__)
005
006 @app.route('/classify/<float:pl>/<float:pw>/<float:sl>/<float:sw>')
007 def classify(pl, pw, sl, sw):
008
009     # load the saved iris classification model
010     model = joblib.load('models/iris_svc.model')
011
012     # Make predictions on request data
013     data = [pl, pw, sl, sw]
014     predictions = model.predict([data])
015
016     #return the classification in JSON format
017     return jsonify({'class':predictions[0]})
018
019 @app.route('/classify', methods=['POST'])
020 def classify_json():
021     # load the saved iris classification model
022     model = joblib.load('models/iris_svc.model')
023
024     content = request.get_json()
025
026     data = []
027     for row in content:
028         pl = row['pl']
029         pw = row['pw']
030         sl = row['sl']
031         sw = row['sw']
032         item = [pl ,pw, sl, sw]
033         data.append(item)
034
035     # make predictions
036     predictions = model.predict(data)
037
038     #return the classification in JSON format
039     return jsonify(especies=predictions[0])
040
041 @app.route('/list', methods=['GET'])
042 def list():
043     # load the saved iris classification model
044     model = joblib.load('models/iris_svc.model')
045
046     iris_mongodb = IrisMongoDB()
047     dataframe = iris_mongodb.getDataframe()
048     print(dataframe)
049
050     json_data = []
051
052     for index, row in dataframe.iterrows():
053         pl = row['petal_length']
054         pw = row['petal_width']
055         sl = row['sepal_length']
056         sw = row['sepal_width']
057
```

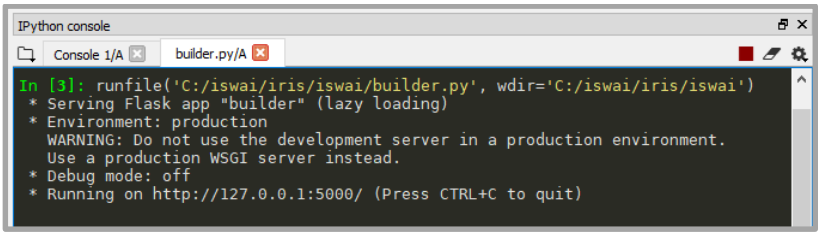
```
058         item = [pl ,pw, sl, sw]
059 
060         category = model.predict([item])[0]
061         json_item = {'pl':pl, 'pw':pw, 'sl':sl, 'sw':sw, 'class':category}
062         json_data.append(json_item)
063 
064     return jsonify(flowers=json_data)
065 
066 if __name__ == '__main__':
067    app.run()
```

Test the Web Services

To test the build model web service, in Spyder open the builder.py file and click the run button.



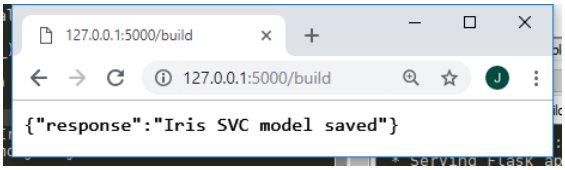
In the IPython console you will see the Flask server running.



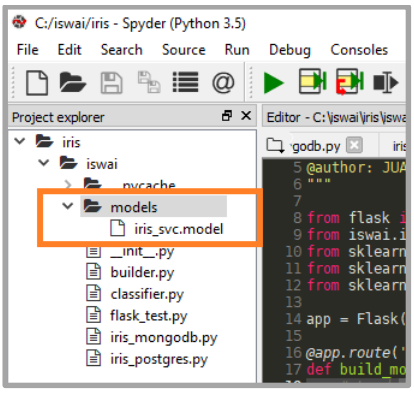
In a browser, open the web service URL:

http://127.0.0.1:5000/build

The web service JSON response, will be shown in the browser.

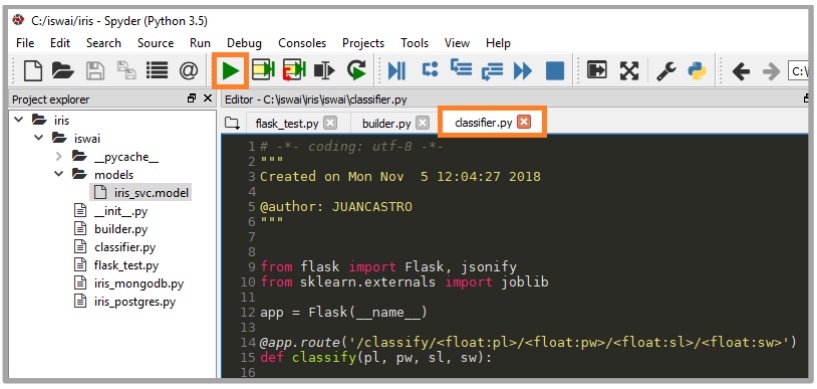


The web service will create a classification model file called iris_svc.model inside the models folder.



Stop the server, click the stop button that is located up and right, or press the CTRL+C keys.

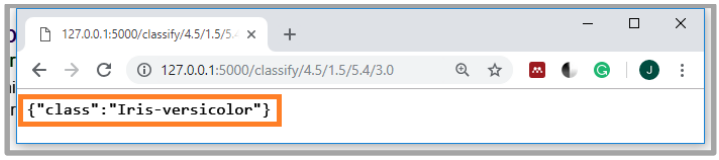
To test the classifier web service, open the classifier.py file and click the run button.



Open the web service URL in a browser and pass the attribute values in the URL:

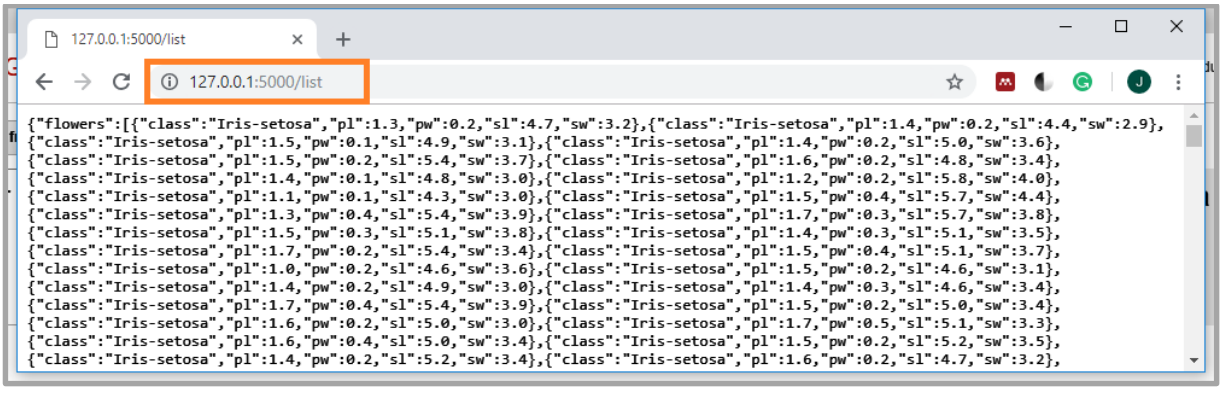
http://127.0.0.1:5000/classify/4.5/1.5/5.4/3.0

The server will send a JSON response with the class label.



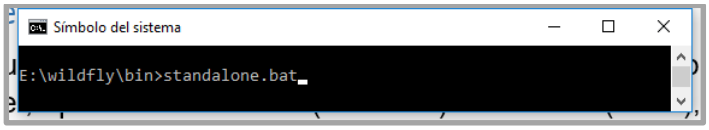
To test the list web service open in a browser the URL:

http://127.0.0.1:5000/list

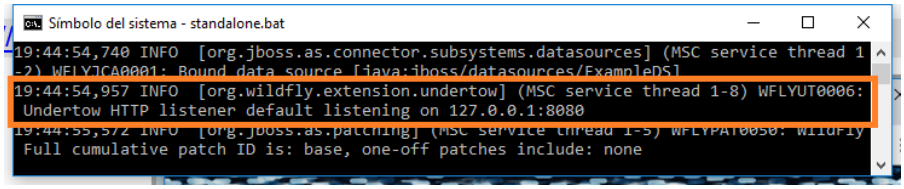


5. Enterprise Application

To build the information system, we will use the WildFly Application Server. To run the WildFly server, open a cmd window (Windows) or terminal (Linux), enter to the bin folder and run the command standalone.bat in Windows or ./standalone.sh in Linux.



In the console window, the server shows the URL to access the server (127.0.0.1:8080).

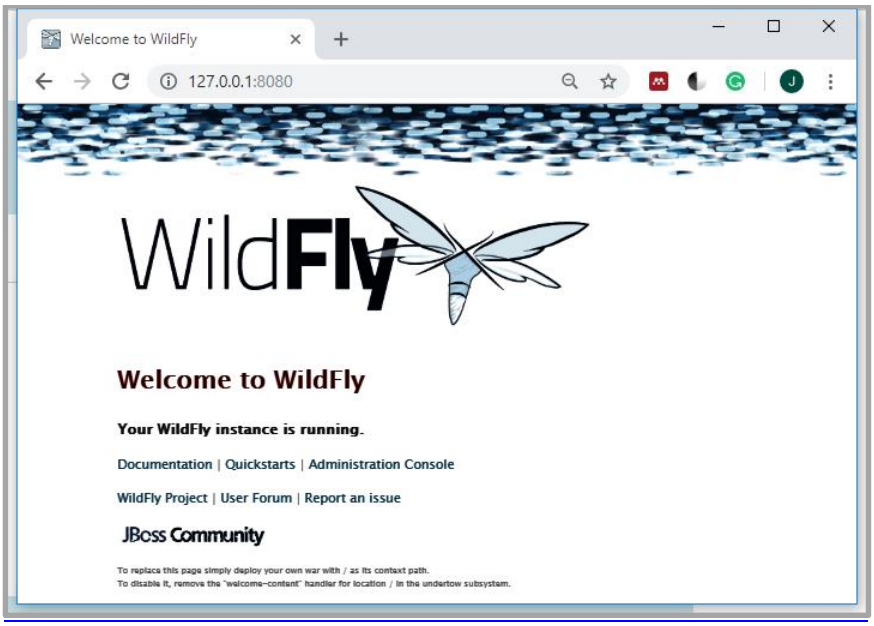


Open a browser and test the server status, you will see the message “Your WildFly instance is running”.

http://127.0.0.1:8080/

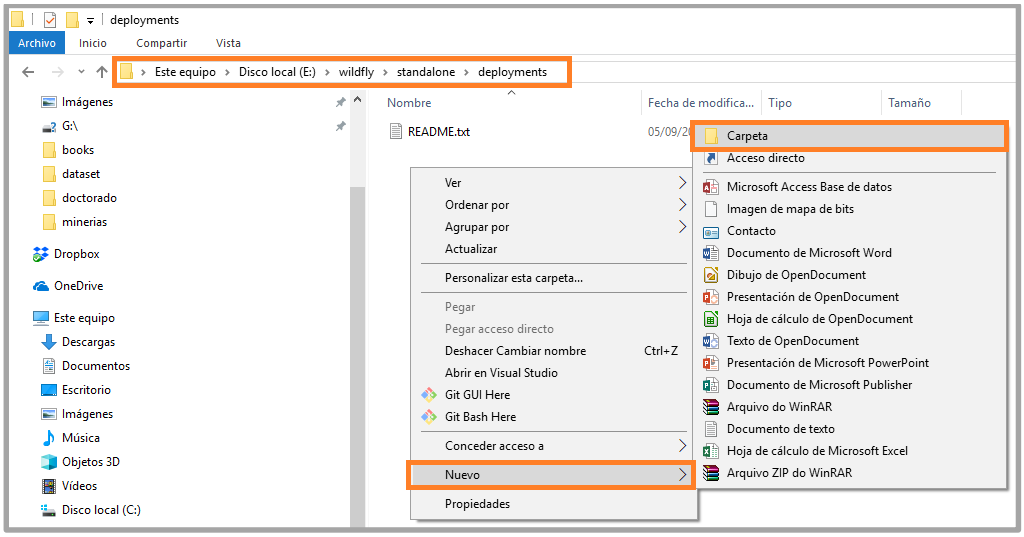
or

http://localhost:8080/

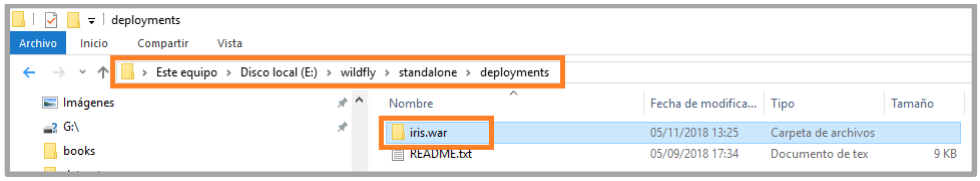


Build the web application

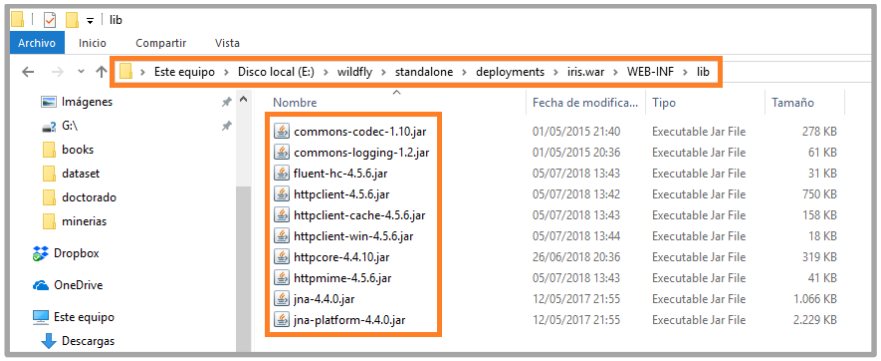
To build a java web application that consumes the machine learning web services (Python) and classify the iris flowers, create a folder called (iris.war) inside the WildFly server publication folder, (wildfly/standalone/deployments).



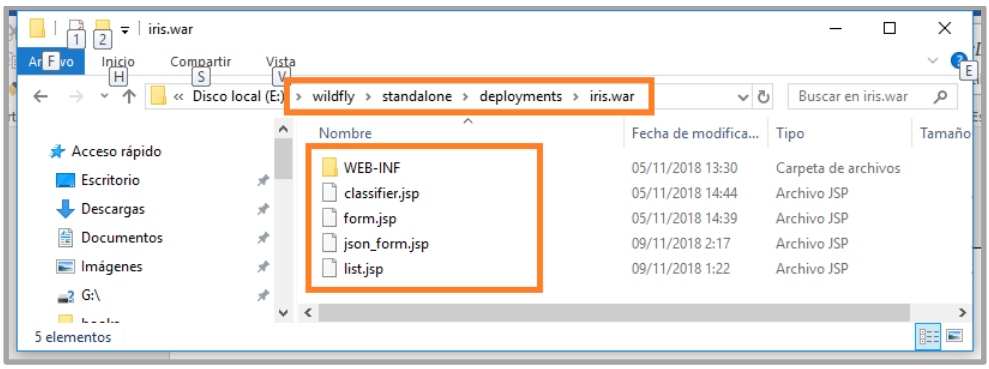
Write the folder application name (iris.war), the war extension means web archive.



Create a folder called (WEB-INF/lib) inside the iris.war folder. Copy the Apache HttpComponents files (*.jar) inside the lib folder.



Create three JSP (Java Server Pages) inside the iris.war folder (form.jsp, classifier.jsp, list.jsp and json_form.jsp).



File: form.jsp

```
<form action="classifier.jsp" method="post">
<p>Petal Length (cm):
<br/>
<input type="text" name="pl">
</p>
<p>Petal Width (cm):
<br/>
<input type="text" name="pw">
</p>
<p>Sepal Length (cm):
<br/>
<input type="text" name="sl">
</p>
<p>Sepal Width (cm):
<br/>
<input type="text" name="sw">
</p>
<p><input type="submit" value="Classify"></p>
</form>
```

File: json_form.jsp

```
<script>
  function classify(){
    // Sending and receiving data in JSON format using POST method
    var pl = parseFloat(document.getElementById("pl").value);
    var pw = parseFloat(document.getElementById("pw").value);
    var sl = parseFloat(document.getElementById("sl").value);
    var sw = parseFloat(document.getElementById("sw").value);

    var xhr = new XMLHttpRequest();
    var url = "http://127.0.0.1:5000/classify";
    xhr.open("POST", url, true);
    xhr.setRequestHeader("Content-Type", "application/json");
    xhr.onreadystatechange = function () {
      if (xhr.readyState === 4 && xhr.status === 200) {
        var json = JSON.parse(xhr.responseText);
        document.getElementById("category").innerHTML = "Category: " + json.especies;
      }
    };
    var data = JSON.stringify([{"pl":pl, "pw":pw, "sl":sl, "sw":sw}]);
    xhr.send(data);
  }
</script>

<form method="post">
<p>Petal Length (cm):
<br/>
<input type="text" name="pl" id="pl">
</p>
<p>Petal Width (cm):
<br/>
<input type="text" name="pw" id="pw">
</p>
<p>Sepal Length (cm):
<br/>
<input type="text" name="sl" id="sl">
</p>
<p>Sepal Width (cm):
<br/>
```

```
<input type="text" name="sw" id="sw">
</p>
<p><input type="button" value="Classify JSON" onclick="classify()">
</form>
<p id="category"></p>
```

File: classifier.jsp

```
<%@ page import="java.io.*" %>
<%@ page import="org.apache.http.HttpResponse" %>
<%@ page import="org.apache.http.client.HttpClient" %>
<%@ page import="org.apache.http.client.methods.CloseableHttpResponse" %>
<%@ page import="org.apache.http.client.methods.HttpGet" %>
<%@ page import="org.apache.http.impl.client.CloseableHttpClient" %>
<%@ page import="org.apache.http.impl.client.DefaultHttpClient" %>
<%@ page import="org.apache.http.impl.client.HttpClients" %>

<%
  String pl = request.getParameter("pl");
  String pw = request.getParameter("pw");
  String sl = request.getParameter("sl");
  String sw = request.getParameter("sw");

  CloseableHttpClient client = HttpClients.createDefault();
  String url = "http://127.0.0.1:5000/classify";
  url = url + "/" + pl + "/" + pw + "/" + sl + "/" + sw;

  HttpGet httpRequest = new HttpGet(url);
  CloseableHttpResponse httpResponse = null;
  try {
    httpResponse = client.execute(httpRequest);
    int status = httpResponse.getStatusLine().getStatusCode();
    if (status >= 200 && status < 300) {
      BufferedReader br;
      br = new BufferedReader(new InputStreamReader(
      httpResponse.getEntity().getContent()));
      String data = "";
      String line = "";
      while ((line = br.readLine()) != null) {
        data = data + line;
      }
      out.print(data);
    }
    else {
      System.out.println("Unexpected response status: " + status);
    }
  }
  catch (IOException | UnsupportedOperationException e) {
    e.printStackTrace();
  }
  finally {
    if(null != httpResponse){
      try {
        httpResponse.close();
        client.close();
      }
      catch (IOException e) {
        e.printStackTrace();
      }
    }
  }
%>
```

File: list.jsp

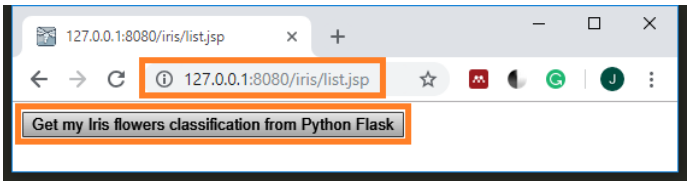
```
<!DOCTYPE html>
<html>
<style>
table,th,td {
  border : 1px solid black;
  border-collapse: collapse;
}
th,td {
  padding: 5px;
}
</style>
<body>

<button type="button" onclick="loadXMLDoc()">Get my Iris flowers classification from Python Flask</button>
<br><br>
<table id="demo"></table>

<script>
function loadXMLDoc() {
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      myFunction(this);
    }
  };
  xmlhttp.open("GET", "http://127.0.0.1:5000/list", true);
  xmlhttp.send();
}
function myFunction(data) {
  var str = data.responseText;

  var x = JSON.parse(str);
  x = x.flowers;

  var table = "<tr><th>#</th><th>Petal Length</th><th>Petal Width</th>"
  table = table + "<th>Sepal Length</th><th>Setal Width</th><th>Class</th></tr>";

  for(var i=0; i<x.length; i=i+1){
    table += "<tr>";
    table += "<td>" + (i+1) + "</td>";
    table += "<td>" + x[i].pl + "</td>";
    table += "<td>" + x[i].pw + "</td>";
    table += "<td>" + x[i].sl + "</td>";
    table += "<td>" + x[i].sw + "</td>";
    table += "<td>" + x[i].class + "</td>"
    table += "</tr>";
  }
  document.getElementById("demo").innerHTML = table;
}
</script>

</body>
</html>
```

Test the Web Application

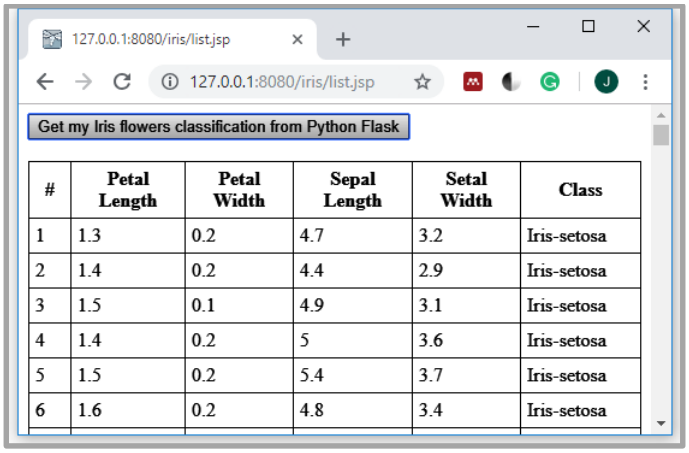
To test the list web service, open in a browser the URL:

http://127.0.0.1:8080/iris/list.jsp

Click the [Get my Iris flowers classification from Python Flask] button.



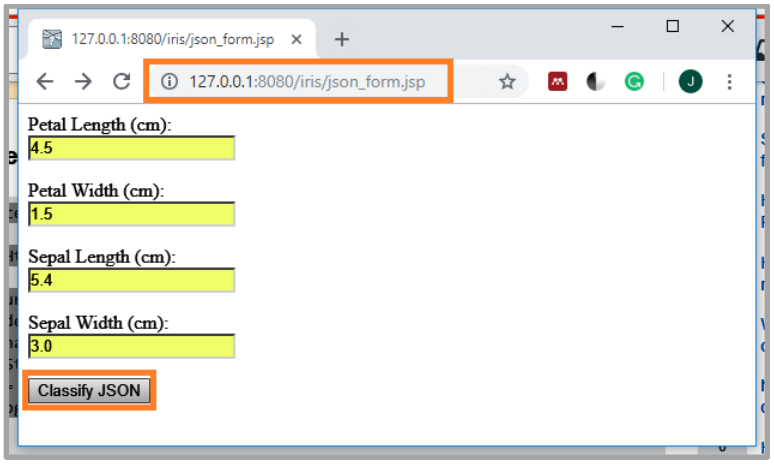
The server JSON response is shown in a HTML table using JavaScript (AJAX).



#	Petal Length	Petal Width	Sepal Length	Setal Width	Class
1	1.3	0.2	4.7	3.2	Iris-setosa
2	1.4	0.2	4.4	2.9	Iris-setosa
3	1.5	0.1	4.9	3.1	Iris-setosa
4	1.4	0.2	5	3.6	Iris-setosa
5	1.5	0.2	5.4	3.7	Iris-setosa
6	1.6	0.2	4.8	3.4	Iris-setosa

To test the AJAX form, open the URL:

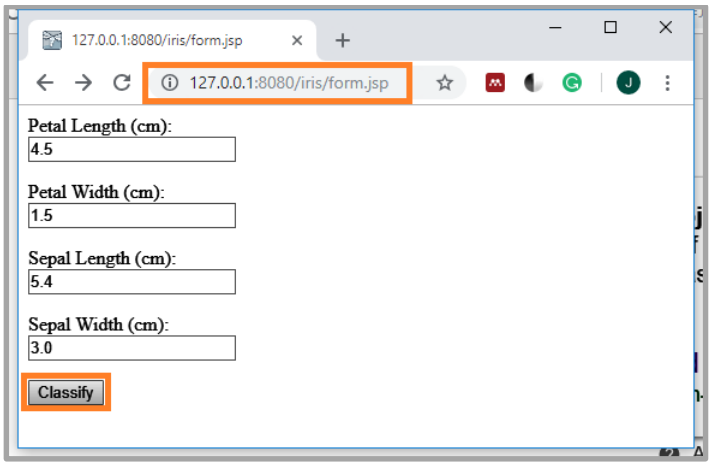
http://127.0.0.1:8080/iris/json_form.jsp



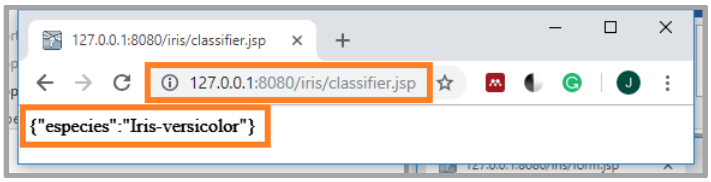
Write the attribute values in the form and click the [Classify JSON] button.

To test the normal (common) POST form, open the URL:

http://127.0.0.1:8080/form.jsp



White the attribute values in the form and click the [Classify] button.



The server JSON response is shown in the browser.

6. REFERENCES

[1] S. Raschka, *Python Machine Learning*. Packt Publising Ltd., 2015.

[2] M. Lichman, "Iris Data Set," *UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]*, 2013. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Iris. [Accessed: 05-Nov-2018].

[3] "Apache HttpComponents," 2015. [Online]. Available: http://hc.apache.org/. [Accessed: 08-Nov-2018].

[4] R. Chakraborty, "Apache HTTP Client Example." [Online]. Available: https://examples.javacodegeeks.com/enterprise-java/apache-http-client/apache-http-client-example/. [Accessed: 08-Nov-2018].

[5] "Flask-CORS." [Online]. Available: https://flask-cors.readthedocs.io/en/latest/. [Accessed: 08-Nov-2018].

[6] "scikit-learn." [Online]. Available: https://scikit-learn.org/stable/. [Accessed: 08-Nov-2018].

[7] "Model persistence." [Online]. Available: http://scikit-learn.org/stable/modules/model_persistence.html. [Accessed: 08-Jan-2018].

[8] "Joblib: running Python functions as pipeline jobs." [Online]. Available: https://joblib.readthedocs.io/en/latest/. [Accessed: 04-Nov-2018].