**Creating a Flask Web App and Deploying it on AWS**

Technologies used: Python, Flask, Html, CSS Bootstrap, AWS EC2

The Problem:

To create a predictive model to classify the poker hand based on a set of attributes and build a web app that lets the user interact with the model by selecting the input attributes and getting the predictions. Further, deploying the web app on AWS EC2 to provide accessibility. The steps for creating the learning model are not described in this document to reduce the scope to deployment of the app.

The entire solution can be found at <https://github.com/mihird94/Machine-Learning-webApp>

The Data:

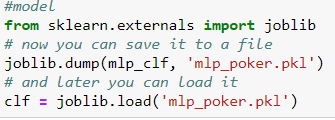
[Poker Hand Dataset](http://archive.ics.uci.edu/ml/datasets/Poker+Hand) is a publicly available dataset on UCI machine learning repository. It contains 11 attributes, 10 being about the selection of 5 cards and their suits respectively and 11 is the target attribute CLASS or the poker hand.

Attribute Information:

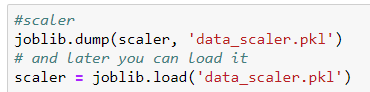
1. S1 "Suit of card #1"   
   Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}   
     
   2) C1 "Rank of card #1"   
   Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)   
     
   3) S2 "Suit of card #2"   
   Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}   
     
   4) C2 "Rank of card #2"   
   Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)   
     
   5) S3 "Suit of card #3"   
   Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}   
     
   6) C3 "Rank of card #3"   
   Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)   
     
   7) S4 "Suit of card #4"   
   Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}   
     
   8) C4 "Rank of card #4"   
   Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)   
     
   9) S5 "Suit of card #5"   
   Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}   
     
   10) C5 "Rank of card 5"   
   Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)   
     
   11) CLASS "Poker Hand"   
   Ordinal (0-9)   
     
   0: Nothing in hand; not a recognized poker hand   
   1: One pair; one pair of equal ranks within five cards   
   2: Two pairs; two pairs of equal ranks within five cards   
   3: Three of a kind; three equal ranks within five cards   
   4: Straight; five cards, sequentially ranked with no gaps   
   5: Flush; five cards with the same suit   
   6: Full house; pair + different rank three of a kind   
   7: Four of a kind; four equal ranks within five cards   
   8: Straight flush; straight + flush   
   9: Royal flush; {Ace, King, Queen, Jack, Ten} + flush

The Learning Model:

A Multi-Layer perceptron model was trained using the scikit-learn library in python and the model was saved on disk using the joblib library to be used in the web app. This saved model can be used to make prediction while deploying the web app without the need to train the model in the web-app back-end script.



Similarly, for the data-processing step in the web app, the python object used for scaling was pickled.



These objects were saved in the application folder and a script ‘helper\_fns’ was written that modularized the preprocessing and predictive steps that the app that to take in the backend using these objects. This script was then called in the flask application script. Once the model was trained and saved, the flask application was written.

Front-end:

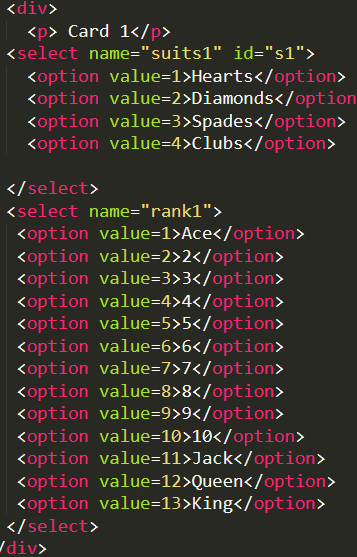
The front-end section consists of two parts, the main landing page used for taking inputs ‘main.html’ and the page that displays the results ‘results.html’. Bootstrap framework was used for CSS-styling and some additional inline CSS was also added. The Bootstrap framework can be imported by various ways. A CDN can be used to directly import the library over a https connection. The files can also be downloaded from their website and added in the project directory. In this app, a python package called flask-bootstrap was installed using pip and imported into the flask app.py script.

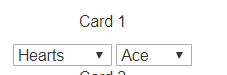
*main.html*

This page allows to take user inputs and send them to the back-end. <form> tag is used to submit the user input to the backend using flask. This done by declaring the method POST in the <form> tag. url\_for(‘predict’) is a url building method that points to the predict route of the application. It is used with {{url\_for(‘predict’)} curly braces to use it as a python method within html.

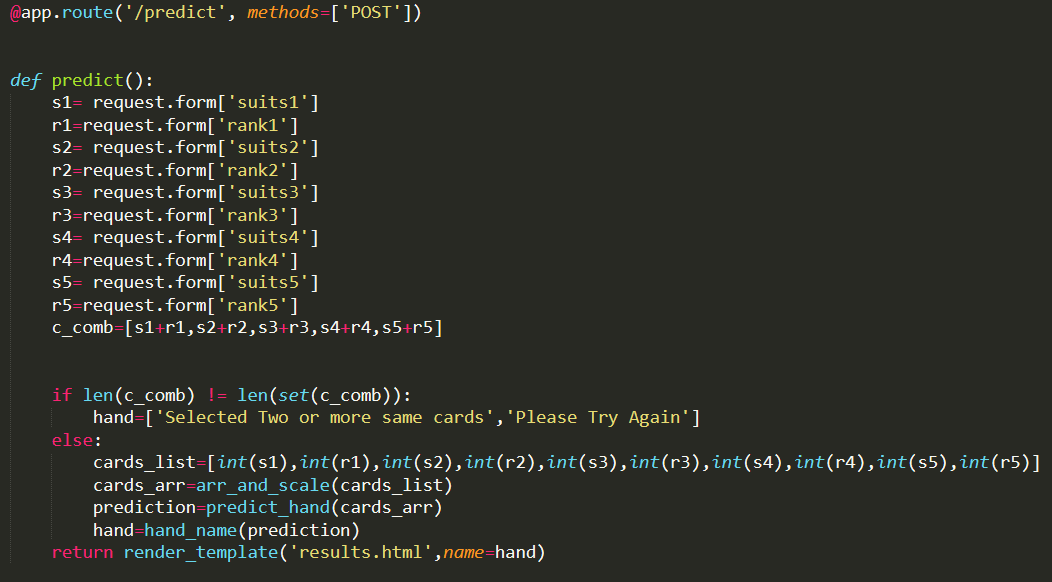


Dropdown menus are created using <select> tag. A total of 10 <select> tags were used to create dropdowns for the 10 input variables for 5 cards. The name attribute in the select tag gives a unique identifier to the inputs to be used in the back-end.

This snippet demonstrates creating the dropdown menu for one card.

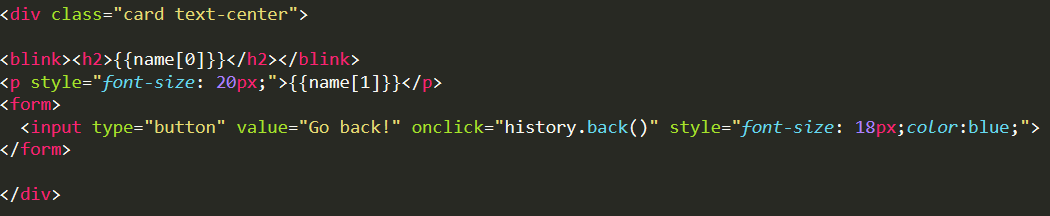


Similarly, dropdowns for all 5 cards were created. Finally, an input tag with attribute type submit was added after all the select tags and before closing the form tag. This creates a button named predict hand. This button posts the inputs in the form and re-directs to the predict route of the flask application. This triggers the function predict in the back-end.



When the user inputs all the values and presses the predict button, this function runs in the back-end, defines python variables with the html inputs using requests module, does input error handling for same values, manipulates the data , applies three user-defined functions imported from helper\_fns script which pre-process the input data, performs the predictions and does some post-processing and stores the result in a variable ‘hand’. It finally returns the render\_template() with the results.html page and the results stored in another variable called name to used in the results page.

*results.html*

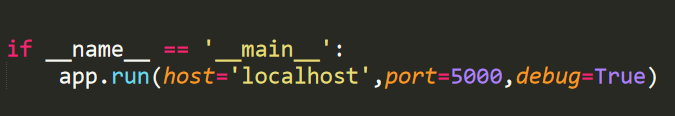


The styling of the page is like the main.html page except this page doesn’t have the <form> tag for the user inputs. Except it uses the <h2> tag and <p> tag to display the results and their description. The name variable is a list of results returned from the prediction step. Another <from> tag is used to embed a button that allows the user to back to the main input page.

Other Flask components



This section initializes the flask app, instantiates the bootstrap framework and renders the main landing page on the app’s root.



This section runs the app on localhost port 5000 and debug=True allows any changes made to reflect live on the app.

**Deploying on AWS EC2:**

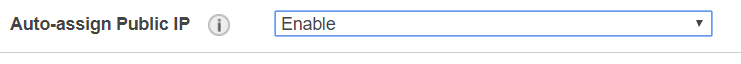
When is the flask app is up and running on the local machine, it is ready to be deployed on an AWS cloud server. This allows the app to be accessed by anyone using the public IP address of the amazon EC2 instance.

To deploy the flask app on EC2:

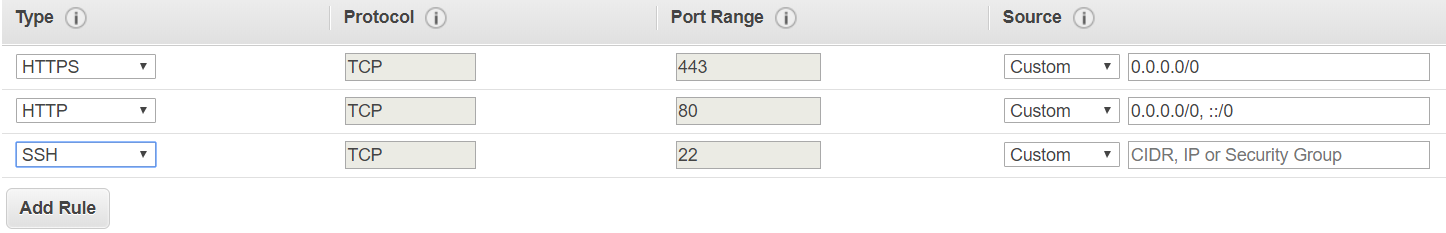
1. Log in to AWS management console
2. Select a region (N. Virginia is our case)
3. Go to EC2 under services
4. Choose Launch Instance
5. Select Ubuntu 16.04 as the Amazon Machine Image or the operating system on our instance.



1. Choose the instance type. For the purpose of this application the free-tier eligible t2 micro with 1GB memory was deployed. But, for commercial or heavy loads, appropriate instance type can be selected.
2. In the configure instance tab, choose enable for auto-assign public-IP. This is needed to make our server accessible to the public internet traffic.



1. Select the size of EBS storage volume. This is attached to the instance. For the purpose of this project, 10 GB SSD volume was selected.
2. Create a security group which controls the network access at the instance level



For a public web app, grant http and https access for internet traffic and ssh to login into our server. 0.0.0.0/0 denotes that all traffic is allowed.

1. Create a new key-pair. This will generate a key-pair of public key (AWS holds) and private key(developer holds)
2. Use the private key to ssh into the AWS instance. Linux or mac users can use the terminal using SSH client and windows users can use a SSH application like putty. <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstances.html>
3. Once we are into the instance, install a version of python using

sudo apt-get install python3.6

1. Use scp( secure copy) to move the application directory from local machine to a folder on our EC2 server. Example:

$ scp -r /local/directory/ username@to\_host:/remote/directory/

Make sure that you are a root user using sudo su command. Give all permissions using chmod 777.

1. Once the application is copied to our AWS server, navigate to the application. It will contain a file requirements.txt. It basically is a text file with names and version of all the dependencies.
2. Run pip install -r requirements.txt, this will install all the dependencies.
3. Open the app.py file using nano command.
4. In the app.py file change the host to “0.0.0.0” and port to 80 for http traffic.

**if** \_\_name\_\_ == "\_\_main\_\_":

app.run(host="0.0.0.0", port=80)

1. Run the command python app.py from the shell
2. Go to the EC2 instance’s public IP address to check if the app is deployed
3. If all went well, the IP should be running our flask app’s main landing page

Here’s what the app looks like:-

