National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Divesh Soneji
Student ID: X21172749

School of Computing
National College of Ireland

Supervisor: Prof. Yasantha Samarawickrama

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Divesh Soneji
Student ID:	X21172749
Programme:	Cloud Computing
Year:	2022-2023
Module:	MSc Research Project
Supervisor:	Prof. Yasantha Samarawickrama
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual
Word Count:	400
Page Count:	6

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	13th August 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Divesh Soneji
X21172749

1 Introduction

Mentioned below are the steps to Configure the Notebook file on SageMaker

1.1 Requirement

- Data in the form of CSV
- AWS account
- SageMaker Access with Compute optimize instances.
- Boto 3 and python knowledge
- VScode (For future work)

2 Implementation

1. Login into AWS
2. Get access to SageMaker studo with appropriate permission

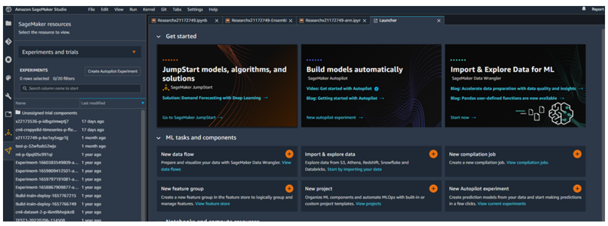


Figure 1: SageMaker Jumpstart Page

3. Create a new Notebook experiment.

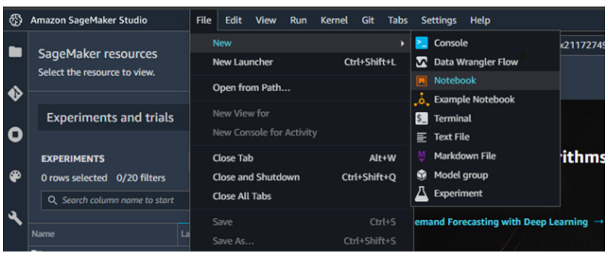


Figure 2: Creating New Notebook

4. Set up the notebook environment and choose suitable kernel image for our experiment select PyTorch 2.0.0 CPU optimized image kernel

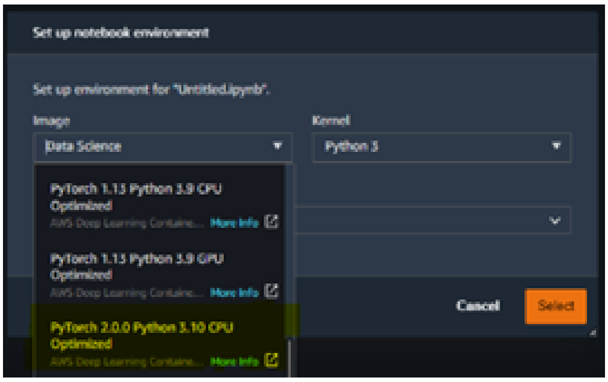


Figure 3: Choosing Kernel Instance

5. Select instance type ml.C5.24xLarge as the model we are experimenting on is complex we need compute optimized instance with large memory and vCPU

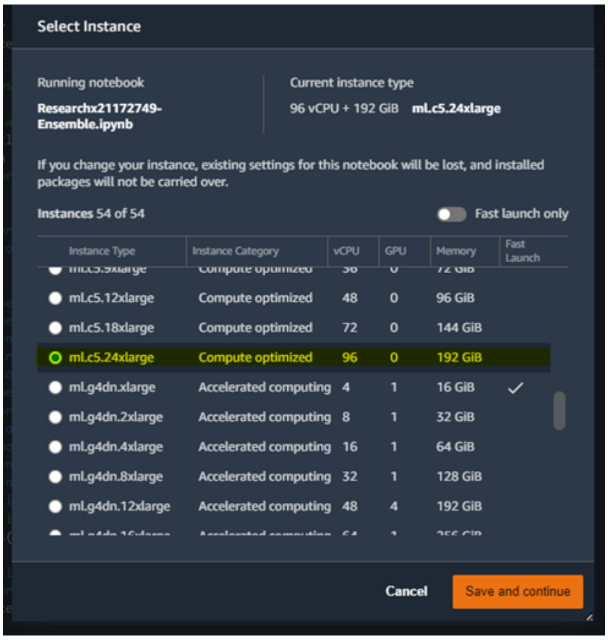


Figure 4: Choosing Cluster Instance

6. Upload the mentioned data into the notebook by clicking on the highlighted button

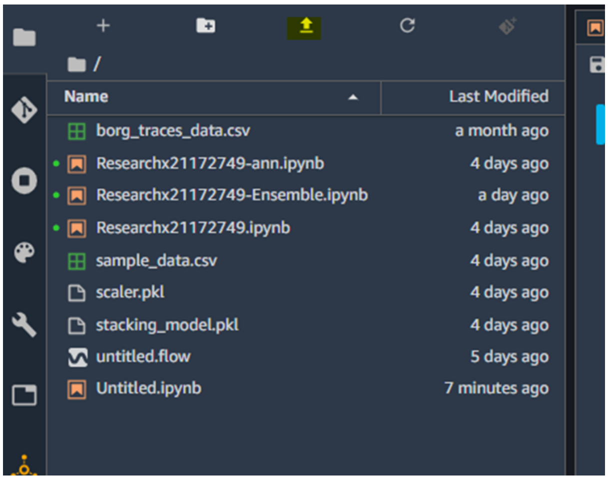
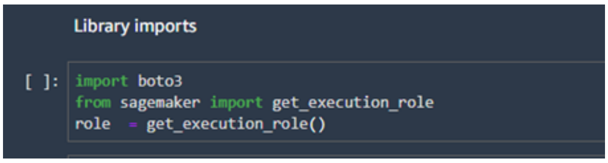


Figure 5: Uploading dataset

7. To work on AWS notebook, we need boto3. This package gives us access to python SDK

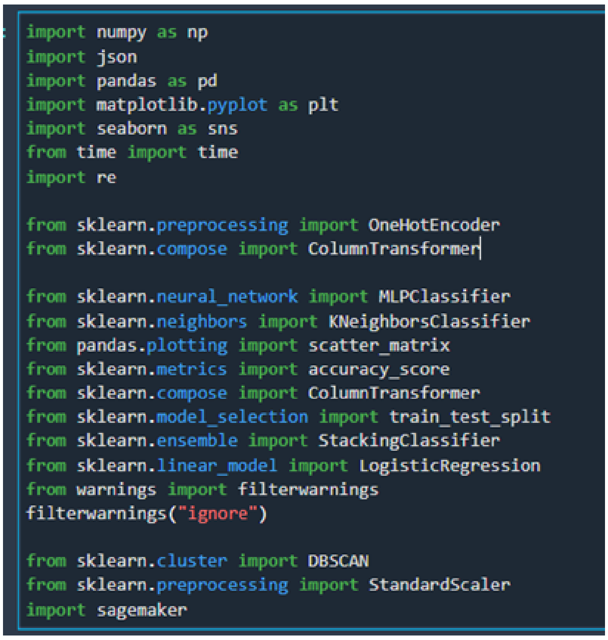


Library imports

```
[ ]: import boto3
     from sagemaker import get_execution_role
     role = get_execution_role()
```

Figure 6: Importing Boto3

8. Now we can import the necessary libraries for implementation of our experiment



```
import numpy as np
import json
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from time import time
import re

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from pandas.plotting import scatter_matrix
from sklearn.metrics import accuracy_score
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from warnings import filterwarnings
filterwarnings("ignore")

from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
import sagemaker
```

Figure 7: Importing libraries

9. Once the experiment prediction is complete, we can export the model as a pkl file which can then be imported into an API framework. Here we are importing joblib library that will dump the two objects and create .plk file with name as mentioned in the fig 8.

```
[750]: import joblib

       model_filename = "stacking_model.pkl"
       scaler_filename = "scaler.pkl"

       joblib.dump(stacking_clf, model_filename)
       joblib.dump(scaler, scaler_filename)

       model_filename, scaler_filename

[750]: ('stacking_model.pkl', 'scaler.pkl')
```

Figure 8: Dumping plk files

10. The files will be created in the root folder where the data is imported

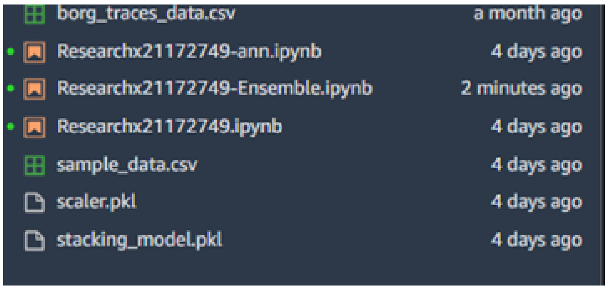


Figure 9: .plk files in the root folder

3 Future work Implementation

1. Create flask API using python by adding the package through python package manger also known as pip using Vs Code terminal

```
pip install Flask
```

Figure 10: Installing flask using PIP

2. Then using the terminal create a virtual environment and activate it

```
>python -m venv .venv
```

Figure 11: Creating Virtual Environment

3. Inside the environment create a class file

```python
import joblib
import numpy as np

class StackingModelAPI:
    def __init__(self, model_path, scaler_path):

        self.model = joblib.load(model_path)
        self.scaler = joblib.load(scaler_path)

    def preprocess(self, input_data):

        data_array = np.array(input_data)
        reshaped_data = data_array.reshape(1, -1)

        scaled_data = self.scaler.transform(reshaped_data)

        return scaled_data

    def predict(self, input_data):

        processed_data = self.preprocess(input_data)

        predictions = self.model.predict(processed_data)

        return predictions.tolist()
```

Figure 12: Creating class file with name StackingModelAPI

4. Now create main.py and create the endpoint in the following file

```python
.venv > main.py > ...
from flask import Flask, request, jsonify
import numpy as np

app = Flask(__name__)

import joblib

from stacking_model_api import StackingModelAPI


@app.route('/predict', methods=['POST'])
def predict():

    data = request.json
    features = np.array(data['features'])
    model = StackingModelAPI('stacking_model.pkl', 'scaler.pkl')

    features_scaled = model.preprocess(features)

    prediction = model.predict(features_scaled)


    return jsonify({'prediction': int(prediction[0])})

if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

Figure 13: Creating an endpoint

5. To debug VS code will auto-generate a launch.json file which contain few debugging instructions

```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Python: Flask",
            "type": "python",
            "request": "launch",
            "module": "flask",
            "env": {
                "FLASK_APP": "main.py",
                "FLASK_DEBUG": "1"
            },
            "args": [
                "run",
                "--no-debugger",
                "--no-reload"
            ],
            "jinja": true,
            "justMyCode": true
        }
    ]
}
```

Figure 14: Auto-generated launch.json

6. We can use postman to test the API that will return a prediction

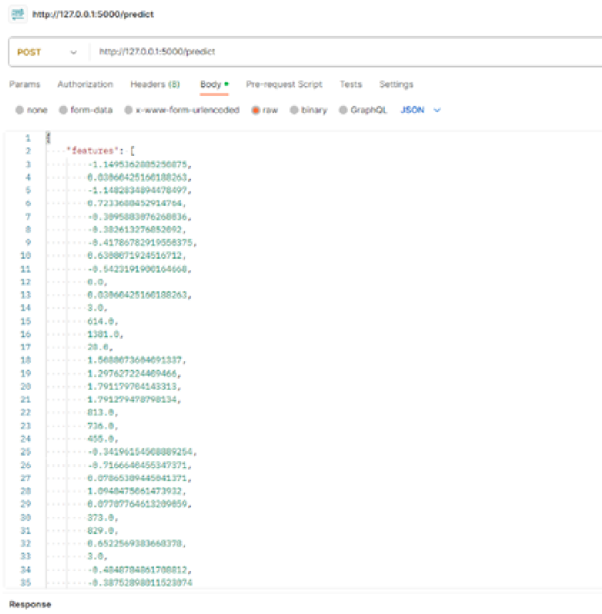


Figure 15: Postman for calling API