CS771A Assignment 3

The Boys

Rajarshi Dutta 200762

Udvas Basak 201056

Shivam Pandey 200938

1 Question 1

The first part of the solution essentially uses a linear model named **Elastic Regression**, which takes into account both the effects of **Ridge Regression Penalty - L2 loss** and **Lasso Regression Penalty - L1 loss**. The objective/loss function of the model is represented below:

$$C(w) = \arg\min_{w} \sum_{i} (y_i - w' x_i)^2 + \alpha\lambda_1 \sum_{i} w_i^2 + \frac{\alpha\lambda_2}{2} \sum |w_i| \tag{1}$$

where,

- w' = Weights multiplied with each of the features present in the dataset taken into account for the construction of the regression model.
- α = constant that multiplies both the lasso and ridge penalties.
- λ_1 = Regularization parameter for the Ridge regression penalty.
- λ_2 = Regularization parameter for the Lasso regression penalty.

During the regularization procedure, the λ_1 parameter leads to the formation of a sparse model, whereas the quadratic section of the penalty makes the part of λ_1 portion more stable to the path of regularization, decreases the number of variables to be selected thereby promoting the grouping effect. The grouping effect enables the variables to be easily identified by correlation leading to the enhancement of the sampling procedure. This method first finds the **Ridge Regression** coefficients and then conducts the second step by using a **Lasso** sort of shrinkage of the coefficients.

Inorder to determine the best performing linear model, **Randomized Search Cross Validation** technique has been applied for efficient hyperparameter selection out of the following sets of hyperparameters of the model. Here **grid** is a dictionary that contains the different hyperparameters of the **Elastic Net Regression** model as keys and its values.

```
grid = dict()
grid['alpha'] = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 0.0, 0.2, 1.0]
grid['l1_ratio'] = np.arange(0,1, 0.01)
grid['selection'] = ['cyclic', 'random']
grid['max_iter'] = [2, 10, 100, 1000]
grid['tol'] = [1e-4, 1e-3, 1e-5, 1e-2, 1e-1, 0.0]
```

The **Randomized Search Cross Validation** returns the best estimator with the following hyperparameters

```
ElasticNet(alpha=0.1, l1_ratio=0.62, selection='random', tol=0.001,
warm_start=True)
```

Preprint. Under review.

The corresponding **MAE** score on the training data are **5.6248** (for the O_3 levels) and **6.5136** (for the NO_2 levels) respectively.

2 Question 2

A suitable non-linear model which can effectively fit on the given dataset is **K Nearest Neighbour** algorithm. The **KNN** can be both used as a classification and regression model, which essentially classifies unknown data points based on the distance with respect to k nearest data points. The model essentially works on two different kinds of distance parameters, namely **Euclidean distance**, Minkowski distance and the **Manhattan distance**.

One strength of the KNN regression algorithm that we would like to draw attention to at this point is its ability to work well with non-linear relationships. In the case of **regression** tasks, the algorithm chooses the k nearest points based on the given **distance** parameter. It calculates the new point to be approximately equal to the average of these nearest points.

Also here **Distance-weighted k-nearest-neighbor** rule should be such where that it varies with the distance between the sample and the considered neighbor in such a manner that the value decreases with increasing sample-to-neighbor distance.

$$w_{ij} = \frac{1}{(d_{ij}(x,y))^p} \tag{2}$$

Here, the optimal value of **K** has been calculated by plotting the **Train MAE loss** and **Test MAE loss** for the case of both O_3 and NO_2 sensor values. The MAE loss variation for different k values has been shown below.

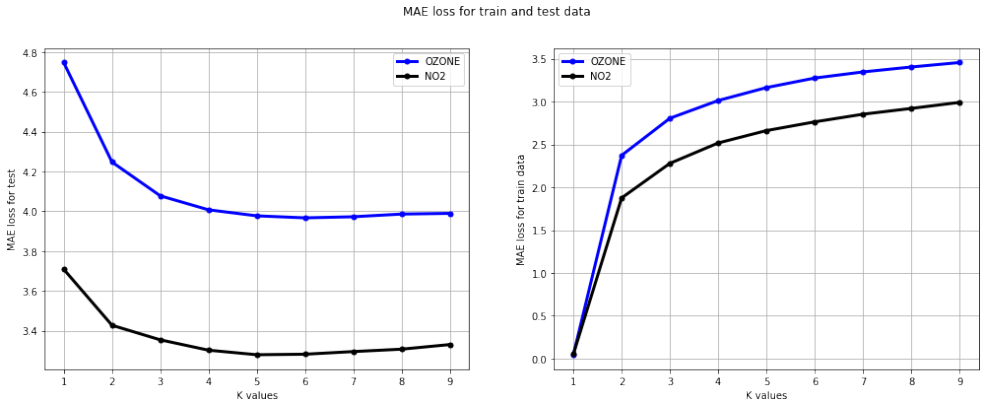

Figure 1: MAE loss variation for K values

We find that the **KNN Regressor** provides the minimum **Mean absolute error** for a k value of 5.

```
    model = KNeighborsRegressor(n_neighbors=5, P = 1, algorithm="auto"
    , n_jobs = -1)
```

Some other models tested for the given regression problem includes **Xgboost**, **Multi-layer Perceptron**, **Random Forest Regressor** models.

- **XGBoost Regressor** : XGBoost uses a combination of two techniques, gradient boosting and tree boosting, to improve the accuracy of the model. Gradient boosting involves minimizing the loss function **mean absolute error** by iteratively adding weak learners (e.g., decision trees) to the model. Tree boosting, on the other hand, involves optimizing the split points of the decision tree by minimizing the loss function.
- **Random Forest Regressor** : The algorithm works by creating a large number of decision trees, each of which is trained on a subset of the data and a subset of the features. The

final prediction is then made by averaging the predictions of all the trees in the forest. Random forest regression also includes several regularization techniques to prevent overfitting and improve the generalization of the model. These techniques include maximum depth, minimum samples per leaf, and maximum features.

- **Arttifical Neural Network** : The given **neural network** consists of 64 followed by 32 and 8 layers. The optimizer used is **Adam**, and the kernel initialisation is **He_uniform** and the loss function used is **Mean absolute Error** loss. The **Perceptron** is backpropagated for around **100 epochs** on the training data with a **batch size of 64**.

Non-linear Models	O_3 **MAE loss**	NO_2 **MAE loss**
K Neighbours Regressor	3.3284	2.3228
XGboost Regressor	5.1433	4.7230
Random Forest Regressor	6.0784	5.3145
MultiLayer Perceptron	4.8807	4.9178

Table 1: Stats of different non-linear models

3 Question 3

The code is available in the **submit.py** file.

Code snippet is mentioned below:

```
import numpy as np
import pickle as pkl

# Define your prediction method here
# df is a dataframe containing timestamps, weather data and
potentials
def my_predict( df ):

  X = np.array(df.iloc[:, 1:])
  # Load your model file

  model = pkl.load(open("knn_model.pkl", "rb"))
  test_preds = np.transpose(model.predict(X))

  # Make two sets of predictions, one for O3 and another for NO2
  pred_o3, pred_no2 = test_preds[0], test_preds[1]

  # Return both sets of predictions
  return ( pred_o3, pred_no2 )
```