FOML ASSIGNMENT 3

Anurag Sarva

cs24mtech14003

1 Non-Uniform Weights in Linear Regression:-

In this question the error function is Given in the question:

$$E_D(w) = \frac{1}{2}\sum_{n=1}^{N} \sigma_n \left(y_n - w^T \Phi(x_n)\right)^2$$

name of the symbols that are used in the error function -
$\rightarrow \Phi(x_n)$ this is the transformation or representation of a data,
$\rightarrow (x_n\ , y_n)$ this are data point of a dataset,
$\rightarrow w$ this is a weighted vector.
$\rightarrow \sigma_n$ this is a weighted factor that associated with the each data points,

Too finding the optimal value of w, for the we have to take the value of the gradient error function with respect to w will be:

$$\nabla_w E_D(w) = \nabla_w \frac{1}{N}\sum_{n=1}^{N} \sigma_n \left(y_n - w^T \Phi(x_n)\right)^2$$

Now using the chain rule form differentiation, then we find this:

$$\nabla_w \left(y_n - w^T \Phi(x_n)\right)^2 = -2\left(y_n - w^T \Phi(x_n)\right)\Phi(x_n)$$

Then, the gradient of the n-th term is looks like:

$$\sigma_n \left(y_n - w^T \Phi(x_n)\right)\Phi(x_n)$$

After all this summing over all N points by considering the weights σ_n, then we obtain the value of the gradient to the entire error function:

$$\nabla_w E_D(w) = -\sum_{n=1}^{N} \sigma_n \left(y_n - w^T \Phi(x_n)\right)\Phi(x_n)$$

To minimize its value $E_D(w)$, what i am doing just setting the gradient equal to zero:

$$\sum_{n=1}^{N} \sigma_n \left(y_n - w^T \Phi(x_n)\right)\Phi(x_n) = 0$$

Now rearranging all the terms:

$$\sum_{n=1}^{N} \sigma_n y_n \Phi(x_n) = \sum_{n=1}^{N} \sigma_n \Phi(x_n) \Phi(x_n)^T w$$

Now solving for w,what i do is, just multiplying inverse of the matrix on the both sides of the above equation :

$$w = \left(\sum_{n=1}^{N} \sigma_n \Phi(x_n) \Phi(x_n)^T \right)^{-1} \sum_{n=1}^{N} \sigma_n y_n \Phi(x_n)$$

At the end the optimal weight vector w^* is looks like:

$$w^* = \left(\sum_{n=1}^{N} \sigma_n \Phi(x_n) \Phi(x_n)^T \right)^{-1} \sum_{n=1}^{N} \sigma_n y_n \Phi(x_n)$$

This is the required expression we have to find in this question.

2 Neural Network:-

Here i am just trying to show that the derivative from differentiation of the error function with the help of the activation function $a_k for an output, which will have the logistic sigmoid activation fu$

$$\frac{\partial E}{\partial a_k} = y_k - t_k$$

This is the cross-entropy error function $E(w)$ which is given in the question, for a network with a soft max output layer is:

$$E(w) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_k(x_n, w)$$

where:
K, this is the total number of classes.
N, this is used to represent number of data samples.
t_{nk},it will used to represent a one-hot encoded vector, which will indicating the true label for the data point n for class k.
$y_k(x_n, w)$, this is the predicted output for the class k, computed with the help of soft max function:

$$y_k(x_n, w) = \frac{\exp(a_k(x_n, w))}{\sum_{j=1}^{K} \exp(a_j(x_n, w))}$$

Here i am just trying to show that the difference(subtraction) of the predicted probability y_k and the true label t_k is a gradient.
Lets see the derivativee of $E(w)$ with respecct to $a_k(x_n, w)$:

$$\frac{\partial E}{\partial a_k} = -\sum_{j=1}^{K} t_j \frac{\partial \ln y_j}{\partial a_k}$$

Here, i am using the chain rule from differentiation, which is like:

$$\frac{\partial \ln y_j}{\partial a_k} = \{ y_k (1 - y_k) if j = k - y_j y_k if j \neq k$$

After combining what i got is this:

$$\frac{\partial E}{\partial a_k} = y_k - t_k$$

3 Ensemble Methods:-

Here, i am using an Jensen's Inequality statement, for the convex function f, it states that, the value of a function at the expected point will be less than or equals to the expected values of the function at random points.
This is look like this-

$$f(E[X]) \leq E[f(X)]$$

here,
f will be a convex function and X will be a random variable.
Let us assume-

$$X = \frac{1}{M} \sum_{m=1}^{M} (y_m(x) - f(x))$$

Define $f(x) = x^2$, it is a exponential function. Now using an Jensen's Inequality by taking the help of $f(x) = x^2$, then we obtain:

$$E_x \left(\frac{1}{M} \sum_{m=1}^{M} (y_m(x) - f(x)) \right)^2 \leq E_x \left(\frac{1}{M} \sum_{m=1}^{M} (y_m(x) - f(x)) \right)^2$$

Now in above equation lets replace, left part with the $E_{(ENS)}$ and right part with the $E_{(AV)}$
Then it is look likes-

$$E_{ENS} \leq E_{AV}$$

This is what we have to proof.
Hence proof.

4 Regularizer:-

This are the few thing which are given in the question:

This is the linear model function 'y' with respect to x and w, which look like-

$y(x, w) = w_0 + \sum_{k=1}^{D} w_k x_k$.

Below is the Sum of squares (sos) error (or mean squared error(mse)) function with respect to the w, which looks like:

$$E(w) = \frac{1}{2}\sum_{i=1}^{N}(y(x_i, w) - t_i)^2$$

The Gaussian noise $\epsilon_k \sim \mathcal{N}(0, \sigma^2)$ is going to add on each of the input variable x_k.

When, the noise ϵ_k is added to each of the input variable x_k, then this will reflect changes on the linear model, which is like:

$$y(x + \epsilon, w) = w_0 + \sum_{k=1}^{D} w_k(x_k + \epsilon_k) = w_0 + \sum_{k=1}^{D} w_k x_k + \sum_{k=1}^{D} w_k \epsilon_k$$

Then the expectation of the error function $E_\epsilon[E_{noisy}(w)]$ over the noise ϵ is becomes:

$$E_\epsilon[E_{noisy}(w)] = E(w) + \frac{\sigma^2}{2}\sum_{k=1}^{D} w_k^2$$

Then this will behaves like a regularizing term.
Now comparing the coefficients, then i identify that the regularization parameter λ is:

$$\lambda = \sigma^2 N$$

This is the relation, that we have to find in this question.

5 Random Forest:-

Part (a): Random Forest Classifier Implementation

For this Random Forest classifier assignment question, i am trying to implementing a Random Forest by using, my own decision tree code, and make ensuring that, it will be adaptable so that i can handle different feature of subsets. Here i am loading the Spam dataset which is provided in the question and then spliting the dataset into training (70%) and testing (30%) with the help of Scikit-learn's `train_test_split`. My Random Forest implementation will creates create a multiple decision trees (defined by `n_trees`), and then utilizing bootstrap sampling for the training data and then randomly selecting a subset of a feature (denoted as `m`) for each of the split in the tree. After the training process, i will be comparing the accuracy and time taken for my this implementation of the random forest with respect to the Scikit learn's, which is a built-in Random Forest classifier.

```
Question 5
part A:
*** Inmplementing Random Forest with help of De.cision Tree ***
#Accuracy# of Custom Random Forest : 0.91, Time Taken: 22.47s
#Accuracy# of Scikit-Learn Random Forest : 0.93, Time Taken: 0.03s
```

Part (b): Sensitivity to Parameter m

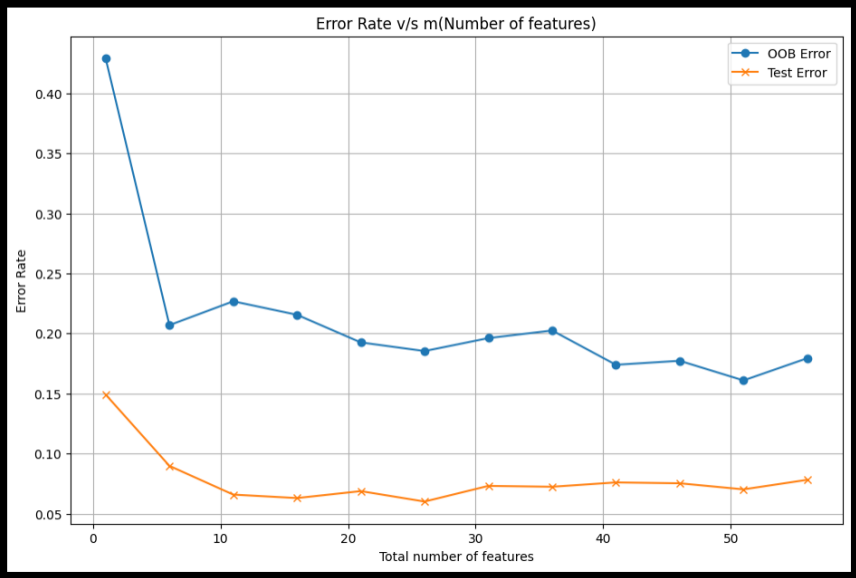
In the second part of this question, i am exploring the sensitivity of the Random Forest by the parameter `m`, with the help of training and evaluation the model across a different range of the `m` values. This will involves the adjusting `m` (i.e. the total number of feature used for a best split) and then observing how it will be changing the affect of the model's performance.

```
part B:
*** Exploring the sensitivity of a Random Forest ***
here is the m's Sensitivity:
for the value of m is 1 having Accuracy is 0.89
for the value of m is 7 having Accuracy is 0.93
for the value of m is 28 having Accuracy is 0.92
for the value of m is 57 having Accuracy is 0.93
```

Part (c): OOB Error and Test Error Analysis

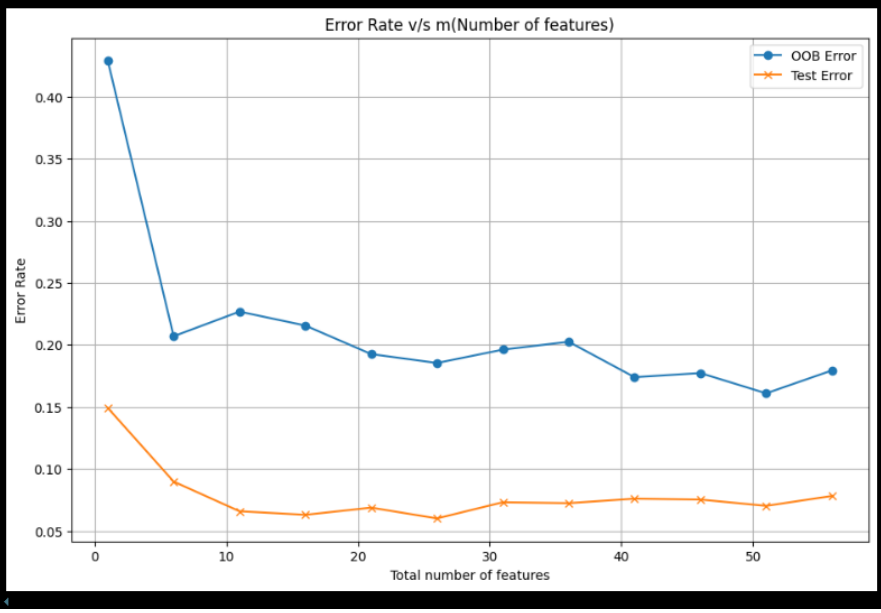
In the third part of this question, i am computing the Out of Bag (OOB) error, which tells the performance of the entire model by using that data which are not included in the bootstrap samples for all the trees, along with the side of the test error. For each of the value of the `m`, gathering the OOB error and then testing the errors and creating a plot so that we can visualize both of the errors as a function of `m`. So, this approach will provide me a comprehensive analysis for the Random Forest performance and the sensitivity to the hyper-parameter.

```
part C:
*** Ploting the OBB error graph ***
```



The complete output.

```
        Assignment 3 FOML
Question 5
part A:
*** Inmplementing Random Forest with help of De.cision Tree ***
#Accuracy# of Custom Random Forest : 0.91, Time Taken: 22.47s
#Accuracy# of Scikit-Learn Random Forest : 0.93, Time Taken: 0.03s
part B:
*** Exploring the sensitivity of a Random Forest ***
here is the m's Sensitivity:
for the value of m is 1 having Accuracy is 0.89
for the value of m is 7 having Accuracy is 0.93
for the value of m is 28 having Accuracy is 0.92
for the value of m is 57 having Accuracy is 0.93
part C:
*** Ploting the OBB error graph ***
```



6 Gradient Boosting:-

Part (a): Data Preprocessing by using Gradiant Boosting

In this part of the question, i have focusing on the preparing of the lending dataset so that i can applying on the Gradient Boosting to classifying whether a customer should be getting a loan or not. So, iin appraoch the first step is loading the datasets from `loan_train.csv` and `loan_test.csv` basically i have two different data sets one is for training and one is for testing, then i am exploring the training data so that i can identify is there any missing value. If i got some of the missing value, then i am handling the missing values by filling the numerical attributes with their median and categorical attributes with their mode, and making ensure that the integrity of the data will be remain to interacting, by avoiding data loss through deletion.

After all this i am performing the feature selection so that it retains only

the relevant attributes, which are necessary for my classification task. and all of this is achieved by dropping an unnecessary features, which is based on the domain knowledge. For all the categorical data, i am just try to transform there variables into binary a features by using an one of the most famous process which is one hot encoding, which will be facilitating the application to the machine learning algorithms. Then the target variable, `loan_status`, is encoded such that the "Fully Paid" is mapped with the +1 and "Charged Off" is mapped with the -1, and so that it aligning with my classification objective.

```
*** Part A, Preprocessing the data ***

Accuracy of part a, after preprocessing: 1.0
Precision of part a, after preprocessing: 1.0
Recall of part a, after preprocessing: 1.0
```

Part (b): Applying Gradient Boosting

In the second part of this question, i am employing the `GradientBoostingClassifier` from the built in scikit learn library. I am initializing the classifier by a specific number of trees, it may be any number of (`n_estimators`), learning rate, and the maximum depth or height of the tree. After that the training of the model on the training dataset, i am evaluating its performance on the validation set, now calculating metrices such as accuracy of a model, precision of a model, and recall. To optimize the model performance, i am conducting the hyperparameter tuning, for specifically analyzing the effect on the increasing of the number of the tree on model accuracy. The best performance among all are recorded, along with the corresponding hyperparameter.

Then finally, i am comparing the performance of the Gradient Boosting model against with the simple Decision Tree classifier builting using an information gain calculation. So, that this comparison will highlight the effectiveness of the Gradient Boosting by handling the loan classification task, as it is generally performing, outperform on the Decision Tree in context of the accuracy of the model, precision of the model, and recall. In this question, i am provided with valuable insights into the data preprocessing step which is necessary necessary to the machine learning applications and for demonstrating the benefits of the advanced ansemble method like Gradient Boosting which is improving classification outcome.

```
*** Part B ***

Part B, Gradient Boosting Classifier and its result :
Best Validation Accuracy for part b: 0.9954741822670232
Best Validation Precision for part b: 0.9947267497603068
Best Validation Recall for part b: 1.0
Best Parameters for part b: {'n_estimators': 200}

Here is the Test-Set Result for the Gradient Boosting Model:
Accuracy in test for part b: 0.994326141776408
Precision in test for part b: 0.9933628318584071
Recall in test for part b: 1.0

Results for the Decision Tree Classifier in Part B:
Accuracy in test of the model: 0.9932
Precision in test of the model: 0.9946
Recall in test of the model: 0.9974
```

The complete output.

```
*** Part A, Preprocessing the data ***

Accuracy of part a, after preprocessing: 1.0
Precision of part a, after preprocessing: 1.0
Recall of part a, after preprocessing: 1.0

*** Part B ***

Part B, Gradient Boosting Classifier and its result :
Best Validation Accuracy for part b: 0.9954741822670232
Best Validation Precision for part b: 0.9947267497603068
Best Validation Recall for part b: 1.0
Best Parameters for part b: {'n_estimators': 200}

Here is the Test-Set Result for the Gradient Boosting Model:
Accuracy in test for part b: 0.994326141776408
Precision in test for part b: 0.9933628318584071
Recall in test for part b: 1.0

Results for the Decision Tree Classifier in Part B:
Accuracy in test of the model: 0.9932
Precision in test of the model: 0.9946
Recall in test of the model: 0.9974
```