Foundations Of Machine Learning : Assignment 3

Name: Ahmik Virani
Roll Number: ES22BTECH11001

October 31, 2024

Question 1 : Non-Uniform Weights in Linear Regression

Answer:

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

We know that to find the value of w that minimizes the the above error function, we need to differential the above function with respect to w and equate it to 0.

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

Thus, simplifying further,

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

Now since, $w^T\Phi(x_n)$ is a scalar and $\phi(x_n)^T w = w^T\phi(x_n)$, rearranging:

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

Therefore the value for w is:

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

Question 2 : Neural Networks

Answer: Given softmax function:

$$y_k(x, w) = \frac{exp(a_k(x, w))}{\sum_j exp(a_k(x, w))}$$

First we need to find the derivative of this softmax function with respect to a_i:
Take log before starting and then differentiate:

$$\log(y_k(x, w)) = a_k(x, w) - \log\left(\sum_j \exp(a_j(x, w))\right)$$

Now we differentiate:

$$\frac{\partial \log(y_k(x, w))}{\partial a_i} = \frac{\partial a_k}{\partial a_i} - \frac{\partial \log\left(\sum_j \exp(a_j(x, w))\right)}{\partial a_i}$$

Now, we look at the second term of the derivative:

$$\frac{\partial \log\left(\sum_j \exp(a_j(x, w))\right)}{\partial a_i} = \frac{1}{\sum_j \exp(a_j(x, w))} \sum_j \left(\exp(a_j(x, w)) \frac{\partial a_k}{\partial a_i}\right)$$

Now, the above is non zero only when i does not equal k Thus the above just equals $y_i(x, w)$ again.
Thus

$$\frac{\partial \log(y_k(x, w))}{\partial a_i} = \delta_{ki} - y_i(x, w)$$

Going back to the loss function:

$$\begin{aligned}
\frac{\partial E}{\partial a_i} &= -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{kn} \frac{\partial \left(\log y_k(x_n, w)\right)}{\partial a_i} \\
&= -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{kn} \left(\delta_{ki} - y_i(x, w)\right) \\
&= -\sum_{n=1}^{N} t_{in} + \sum_{n=1}^{N} \sum_{k=1}^{K} t_{kn} y_i(x, w)
\end{aligned}$$

Now since t is a one-hot vector $\sum_{k=1}^{K} t_{kn} = 1$

$$\frac{\partial E}{\partial a_i} = \sum_{n=1}^{N} \left(y_i(x, w) - t_{in}\right)$$

Therefore the above equations, for a given input, just boils down to:

$$\frac{\partial E}{\partial a_i} = y_i - t_i$$

without loss of generality replace i with k we get

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

Hence proven.

Question 3 : Ensemble Methods

Answer:
Clearly, $(y_m(x) - f(x))^2$ is convex, thus I am going to use Jensen's inequality.

$$E_{AV} = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}\left[(y_m(x) - f(x))^2\right]$$

$$E_{ENS} = \mathbb{E}\left[\left(\frac{1}{M}\sum_{m=1}^{M} y_m(x) - f(x)\right)^2\right]$$

As per the Jensen's inequality:
From Figure 1, we can see that $\frac{1}{y-x+1}\sum_{m=x}^{y} g(m) >= g(\frac{1}{y-x+1}\sum_{m=x}^{y} m)$
Going back to the original question, if we remove the expectations and compare,

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

Taking expectation on both sides (linearity of expectations hold), we prove: $E_{AV} >= E_{ENS}$
Now for a general case $E(y)$, as per Jensen's inequality for a convex function y:

$$\frac{1}{M}\sum_{m=1}^{M} E(y_m) >= E(\frac{1}{M}\sum_{m=1}^{M} y_m)$$

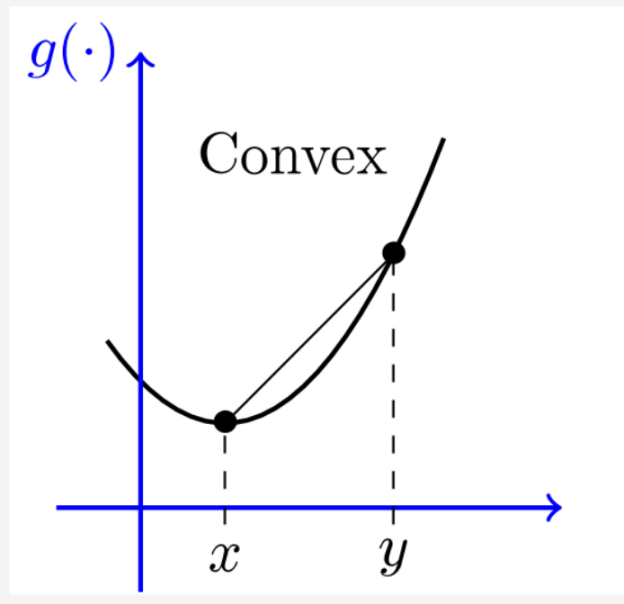

Figure 1: Jensen's Ineqality for convex functions

thus taking expectation on both sides, we get:

$$\frac{1}{M}\sum_{m=1}^{M}\mathbb{E}\left[E(y_m)\right] >= \mathbb{E}\left[E(\frac{1}{M}\sum_{m=1}^{M}y_m)\right]$$

Question 4 : Regularizer

Answer: Let us begin with omiting the w_0 term.

$$y = \sum_{i=1}^{N} w_i x_i$$

Now, due to some noise, say the $x_i' = x_i + \epsilon_i$ Thus the error causes the new predicted output:

$$y' = \sum_{i=1}^{N} w_i x_i + \sum_{i=1}^{N} w_i \epsilon_i \tag{1}$$

$$= y + \sum_{i=1}^{N} w_i \epsilon_i \tag{2}$$

We are interested in $\mathbb{E}\left[(y' - t)^2\right]$

$$\mathbb{E}\left[(y' - t)^2\right] = \mathbb{E}\left[\left(y + \sum_{i=1}^{N} w_i \epsilon_i - t\right)^2\right] \tag{3}$$

$$= \mathbb{E}\left[\left((y - t) + \sum_{i=1}^{N} w_i \epsilon_i\right)^2\right] \tag{4}$$

$$= \mathbb{E}\left[(y - t)^2 + 2(y - t)\left(\sum_{i=1}^{N} w_i \epsilon_i\right) + \left(\sum_{i=1}^{N} w_i \epsilon_i\right)^2\right]. \tag{5}$$

$$= \mathbb{E}\left[(y - t)^2\right] + 0 + \mathbb{E}\left[\sum_{i=1}^{N} w_i^2 \epsilon_i^2\right] \tag{6}$$

Note the above reduces because ϵ_i is independent of ϵ_j as given in the question when i does not equal j and clearly y is the term which is not influenced by error, thus $(y - t)$ is independent of ϵ_i.

Also, $\mathbb{E}\left[\epsilon_i^2\right] = \sigma^2$

Multiplying $\frac{N}{2}$ on both sides and taking $\sum_{i=1}^{N}$, the earlier expression further reduces to:

$$\mathbb{E}\left[\frac{1}{2}\sum_{i=1}^{N}(y' - t)^2\right] = E\left[\frac{1}{2}\sum_{i=1}^{N}(y - t)^2\right] + \frac{N}{2}\sigma^2 \sum_{i=1}^{N} w_i^2$$

Comparing with L2 regularization, $\lambda = \sigma^2 \frac{N}{2}$
Minimizing sum of squares error in noise is same as L2 regularization term.
In summary, the presence of Gaussian noise in the input variables leads to an increased expected error in the predictions. By minimizing the sum-of-squares error averaged over noisy data, we get a penalty on the weights similar to L2 regularization. This relationship indicates the importance of regularization in robust model training, ensuring that the model remains stable and generalizes well to unseen data.

Question 5 : Random Forests

Answer: I have used my code from Assignment 1 and built on that.

Part (a)

Comparison of models:
The model which I have built:
- Accuracy: 0.9341057204923968
- Training Time: 72.49823307991028
- Testing Time: 0.03123784065246582

sklearn classifier:
- Accuracy: 0.9514844315713251
- Training Time: 0.31179022789001465
- Testing Time: 0.010241031646728516

Analysis: The model which I am testing have a trade-off between time and accuracy, when we increase the number of trees in the random forest, the time is more but accuracy is less. However in sklearn, it is giving a higher accuracy even at a smaller time. Now the reason for sklearn being so efficient is because, it uses several optimizations such as parallel processing, adjusting parameters like max_samples and max_features which may reduce run time. It also uses optimal parameters for various hyper-parameters which take a long time testing, which lead to a higher accuracy.
Now, if I need to match or come close to sklearn, I need to do much more testing...
Thanks to the extension in deadline, I did some hyper-parameter tuning and got the model to have a better accuracy of : 0.9485879797248371. (This is shown at the very end the python noteboox)

Part (b)

Sensitivity of Random Forests to the parameter m:

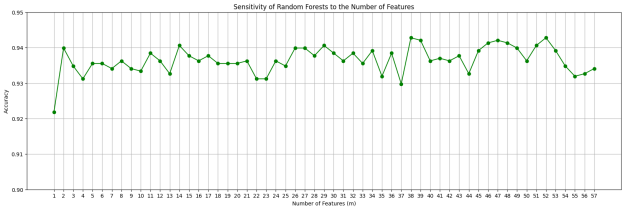

Figure 2: sensitivity of Random Forests to the parameter m

Analysis: When m = 1, the accuracy is the lowest, and at m = 38, it is the highest. However, the gap betweem the lowest and highest accuracy is less than 2%. Thus this model would

be 'called' better when m=38 only when the false negative is very expensive such as in the case of earthquake where lives are at risk. Our scenario is spam detection and it is ok to misclassify some, as long as there are no false positives. However, if we try to analyze the trend, the value of m is not affecting accuracy in a predictable manner, as it increases and decreases arbitrarlily. The reason for such a trend is a low value of m may be very simple whereas a high value would be very complex and may lead to overfitting. Also, it depends on the interactions between features in the dataset. Certain values of m may effectively capture these interactions better than others.

Part (c)

Sensitivity of OOB and test errors to the parameter m:
The range of m which I have chosen for this is from $m = 29$ to $m = 52$. This is because this range covers almost all cases of accuracy, be it increase, decrease, reach a local minimum/maximum and also constant.

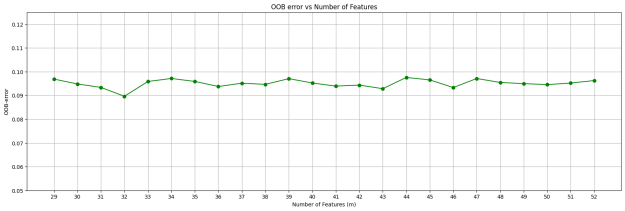

Figure 3: sensitivity of OOB error to the parameter m

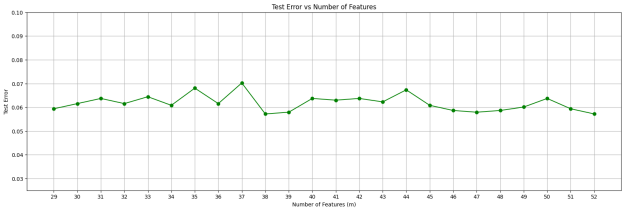

Figure 4: sensitivity of test error to the parameter m

OOB error analysis: OOB error remains almost constant throughout. Some significant observations include, the error taking value of local minimum at m = 32 and m = 46. These are the regions where accuracy was increasing, however at other points no clear trend is shown and this error remains almost constant.

Test error analysis: Note that this is just the opposite of the accuracy, this is as per the definition.

Now once we have a good look and try to compare, both the trends look to be somwhat similar. And this is actually is the good sign because OOB error is on that training data

which we have not trained due to bootstrapping and the below is on the testing model. If these appear to be similar means we have do a pretty good job generalizing the model.
(Note the similarity: beyound m = 39, the two graphs are very similar)

Question 6 : Gradient Boosting

Part (a)

Data pre-processing: The first step which I did was to check for null values in the columns. Also, I checked for garbage values and any duplicate rows present. I found that there were no garbage values nor there were any duplicate rows, however there were many null values.

To handle null values, I used a function where I stated that any columns which have more than 50% of values which are null/Nan, I would delete and not consider such columns. Next, any remaining null values, based on the data type I assigned the value. For example, if the value was numerical, I assigned the mean value, if it was categorical, I assigned it the mode value.

The next step which I did was to ensure to only consider those rows which have loan_status value "Fully Paid" or "Charged off". Also, for the remaining categorical data which I felt were useful, I followed a function that would get the one-hot encoding vector for them.

Finally, for the preprocessing step, as stated in the question, we were allowed to eyeball the attributes which may not be used. I did so and even plotted some histograms to get some idea of how the trend is. Also, some of the columns had percentage, so I took care of them by stripping the sign and assigning a numerical value.

Part (b)

Gradient Boosting Classifier: The first thing I did was to test the sklearn's standard Decision Tree Classifier and record its accuracies on the testing sample based on the 10 fold cross validation. The results I got are reported here:
- Test Accuracy: 0.9976183804987392
- Test Precision: 0.9973669052908747

- Test Recall: 0.9998350243339107

Next I focused on the hyperparameters that affect the gradient boosting classifier.
First I took the parameters individually and then kept the other default. I took the following observations:
- Best Learning Rate: 0.5, best accuracy: 0.9966377136452788
- Best Max Depth: 7, Accuracy: 0.9978985710282993
- best_n_estimators: 400, Accuracy: 0.9969179041748389
- best_min_samples_split: 5, Accuracy: 0.9948865228355281

Then I decided to take the best 2-3 values of each and iterate a for loop to test what best accuracy I can get, and the best test accuracy I got was:
- best_learning_rate: 0.3, best_n_estimators: 300 best_min_samples_split: 4, best_max_depth: 5, best Accuracy: 0.9981087139254693

Note that in the question, we were asked to maximize the test accuracy, so I assumed that basically we need to maximize that, so we need to consider it as the validation set and do that, because otherwise it is not possible to maximize test data without actually testing hyperparameters on that.

Finally lets compare the parameters of the one I reported to be best, and the one we got using a decision tree:
My reported best: learning_rate: 0.3, _estimators: 300 min_samples_split: 4, max_depth: 5, Accuracy: 0.9981087139254693, Precision: 0.9977777777777778, Recall: 1.0

Decision Tree: Test Accuracy: 0.9976183804987392, Test Precision: 0.9973669052908747, Test Recall: 0.9998350243339107

Analysis:
1. Performance: The performance of both the models is high, however the gradient boosting model performed slightly better and this reflects the ensemble model's ability to generalize slightly better by combining multiple learners.

2. Precision: The precision of both the models is high, showing that false positives are less, however the gradient boosting model precision is slightly better due to its robust nature in dealing with misclassifications.

3. Recall: The ensemble classifier has the best possible recall, which means that it would not miss any positives. Such a classifier is very useful when you would never want to get a false positive perhaps due to its high cost.

Efficiency Trade-off: The decision tree is more efficient in terms of training and testing time. If computational efficiency is more important and a not fully perfect outcome is satisfiable, a decision tree might be a more practical choice. However, if perfection is required, then

ensemble learners are better.

References:
- Open Source
- Youtube tutorials

I have taken reference from online sources however I have made all the necessary changes as per my knowledge that the questions have asked and done all the work on my own.