FoML assignment IV : Linear regression, Optimal Bayes classifier, VC dimension, Regularizers

Tamal Mondal

November 17, 2021

1 Non-Uniform Weights in Linear Regression

1.1 Minimize weighted least square error

If the data points of a dataset is denoted by (x_n, t_n) where $n = 1, 2, .., N$ and each data point is associated with a non-negative weighting factor $g_n > 0$ then weighted squared error function is as follows:

$$E_D(w) = \frac{1}{2}\sum_{n=1}^{N} g_n(t_n - w^T\phi(x_n))^2 \qquad \text{where } \phi(.) \text{ is any representation of the data}$$

Now to get an expression for the solution w^* that minimizes the above error function, we can simply take the gradient with respect to w and set it to 0(zero) as following:

$$\frac{\partial(E_D(w))}{\partial w} = -\sum_{n=1}^{N} g_n(t_n - w^T\phi(x_n))\phi(x_n) = 0$$

The above expression can be simplified as follows:

$$\left(\sum_{n=1}^{N} g_n\phi(x_n)\phi(x_n)^T\right) w = \sum_{n=1}^{N} g_n t_n\phi(x_n)$$

$$w^* = \left(\sum_{n=1}^{N} g_n\phi(x_n)\phi(x_n)^T\right)^{-1} \sum_{n=1}^{N} g_n t_n\phi(x_n)$$

If we want to remove the summation from the above expression and use matrix instead where G is a diagonal matrix having the weights $g_1, g_2, ..., g_N$ along the diagonal and all other elements are 0, ϕ is the dataset and t is ground truth results in matrix form, then the above expression can be written as follows:

$$w^* = (\phi^T G\phi)^{-1}(\phi G t)$$

1.2 Two interpretations

1.2.1 Data-dependent noise variance

When the noise is constant among all data points, the scenario is called **homoskedasticity** whereas when the magnitude of the noise is not constant, it's called **heteroskedasticity**. The **ordinary least square error** function works well in case of homoskedasticity as it gives equal importance to each data point or it measures the regression function with the same degree of precision for all the samples.

But in case of heteroskedasticity, ordinary least square error will not give optimal results as we can't give equal importance to every point because noise variance is different for each. Intuitively we want to measure more accurately where the noise is less and fine if it fits poorly where the noise is high. **Weighted least square error** function works well in this scenario when associated weights are inversely proportional to the noise variance($g_i \propto \frac{1}{\sigma_i^2}$). So where the noise variance is small, weight will be large so the model will try to measure it more accurately than the points having smaller weights(larger noise variance).

1.2.2 Replicated data points

When we closely look into the expression for weighted least square error function, we can see that ordinary least square error is just a special case of weighted least square error where all the weights(g_i's) are same(1) for all the points. So extending this idea, we can say that weighted least square error is the same(or applicable) when ith data point is repeated g_i times.

2 Bayes Optimal Classifier

From the different probabilities given in the question, followings are the different **Bayes optimal estimates** for the robot to move either Forward(F) or Left(L) or Right(R):

$$P\left(\frac{F}{D}\right) = \sum_{h_i} P\left(\frac{F}{h_i}\right) * P\left(\frac{h_i}{D}\right) = (0.4 \times 1) + 0 + 0 + 0 + 0 = 0.4$$

$$P\left(\frac{L}{D}\right) = \sum_{h_i} P\left(\frac{L}{h_i}\right) * P\left(\frac{h_i}{D}\right) = 0 + (0.2 \times 1) + 0 + (0.1 \times 1) + (0.2 \times 1) = 0.5$$

$$P\left(\frac{R}{D}\right) = \sum_{h_i} P\left(\frac{R}{h_i}\right) * P\left(\frac{h_i}{D}\right) = 0 + 0 + (0.1 \times 1) + 0 + 0 = 0.1$$

So comparing the above conditional probabilities we can say that **moving left(L)** is the most probable for a new instance or sample according to Bayes optimal classifier.

Now followings are the **Maximum a Posteriori(MAP)** estimates for the given hypothesises: $P\left(\frac{h_1}{D}\right) = 0.4, P\left(\frac{h_2}{D}\right) = 0.2, P\left(\frac{h_3}{D}\right) = 0.1, P\left(\frac{h_4}{D}\right) = 0.1$ *and* $P\left(\frac{h_5}{D}\right) = 0.2$. So, as per MAP the most likely hypothesis for given training sample is $\boldsymbol{h_1}$ and that means the most probable result is **moving Forward(F)**.

So the MAP and Bayes optimal estimates are **not same**.

3 VC-Dimension

The decision boundary for the given hypothesis can be imagined with a pair of parallel lines(at x=p and x=q) and any sample between them will have a label 1. So clearly the hypothesis can shatter any arrangement of 2 points for any assignment but with 3 points it's not possible as shown in the figure 1.

So by the definition, VC-Dimension for the given hypothesis is **2**.

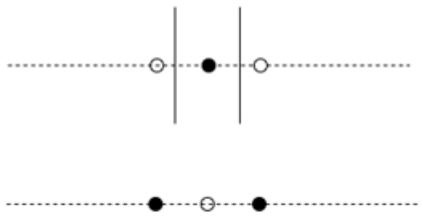

Figure 1: Given hypothesis can't shatter all arrangement of 3 points

4 Regularizer

Given that we have D-dimensional data $X = [x_1, x_2, .., x_D]$ and so the linear model is as follows:

$$Y(X, w) = w_0 + \sum_{k=1}^{D} w_k x_k$$

Now, a Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$(i.e. zero mean and variance σ^2) is added independently to each of the input variables x_k. So we can represent modified linear model $y_{noise}(X_i, w)$ for ith data point as follows:

$$Y_{noise}(X_i, w) = \left(w_0 + \sum_{k=1}^{D} w_k x_k \right) + \sum_{k=1}^{D} w_k \epsilon_{ik} = Y(X_i, w) + \sum_{k=1}^{D} w_k \epsilon_{ik} \tag{1}$$

Now we know that minimizing the standard sum-of-squares error (averaged over noise-free input data) with a L_2 weight-decay regularization term can be expressed as follows where λ is the regularization parameter:

$$W = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{N} \left(Y(X_i, w) - t_i \right)^2 + \frac{\lambda}{2} \sum_{k=1}^{D} w_k^2 \tag{2}$$

We need to find relation of the above minimization with minimizing the sum-of-squares(or mean-squared-error) error averaged over the noisy data which is defined as:

$$E(w) = \frac{1}{2} \sum_{i=1}^{N} \left(Y_{noise}(X_i, w) - t_i \right)^2$$

We will try to find the relation by finding expectation of above mentioned error function over the noisy data in the following way:

$$\begin{aligned}
E_{noise} &= \frac{1}{2} \mathbb{E} \left[\sum_{i=1}^{N} \left(Y_{noise}(X_i, w) - t_i \right)^2 \right] \\
&= \frac{1}{2} \mathbb{E} \left[\sum_{i=1}^{N} \left(\left(Y(X_i, w) - t_i \right) + \sum_{k=1}^{D} w_k \epsilon_{ik} \right)^2 \right] \quad \text{(using equation 1)} \\
&= \frac{1}{2} \mathbb{E} \left[\sum_{i=1}^{N} \left(\left(Y(X_i, w) - t_i \right)^2 + 2.\left(Y(X_i, w) - t_i \right). \sum_{k=1}^{D} w_k \epsilon_{ik} + \left(\sum_{k=1}^{D} w_k \epsilon_{ik} \right)^2 \right) \right] \\
&= \frac{1}{2} \left[\sum_{i=1}^{N} \left(Y(X_i, w) - t_i \right)^2 + 2\mathbb{E} \left[\sum_{i=1}^{N} \left(\left(Y(X_i, w) - t_i \right). \sum_{k=1}^{D} w_k \epsilon_{ik} \right) \right] + \mathbb{E} \left[\sum_{i=1}^{N} \left(\sum_{k=1}^{D} w_k \epsilon_{ik} \right)^2 \right] \right] \\
&= \frac{1}{2} \left[\sum_{i=1}^{N} \left(Y(X_i, w) - t_i \right)^2 + \mathbb{E} \left[\sum_{i=1}^{N} \left(\sum_{k=1}^{D} w_k \epsilon_{ik} \right)^2 \right] \right] \text{(As in the middle term, } \epsilon_{ik} \text{ is independent of } w_k \text{ and } (Y - t_i)) \\
&= \frac{1}{2} \sum_{i=1}^{N} \left(Y(X_i, w) - t_i \right)^2 + \frac{1}{2} \sum_{i=1}^{N} \mathbb{E} \left[\sum_{k=1}^{D} w_k^2 \epsilon_{ik}^2 \right] \quad \text{(As, } \epsilon_{ik} \text{ is independent of } \epsilon_{jl} \text{ where } i \neq j \text{ or } k \neq l) \\
&= \frac{1}{2} \sum_{i=1}^{N} \left(Y(X_i, w) - t_i \right)^2 + \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{D} w_k^2 \sigma_{ik}^2 \quad \text{(As, } \mathbb{E}[\epsilon^2] = variance(\sigma^2)) \\
&= \frac{1}{2} \sum_{i=1}^{N} \left(Y(X_i, w) - t_i \right)^2 + \frac{N\sigma^2}{2} \sum_{k=1}^{D} w_k^2
\end{aligned}$$

If we compare the above expression with the sum-of-squares error (averaged over noise-free input data) with L_2 weight-decay regularization in equation 2, we can write:

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

5 Logistic regression programming question(part b)

5.1 Logistic model and cross-entropy error

Given that the linear model used is $f_\theta(x_1, x_2) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$ and the logistic regression function is $\sigma(f_\theta(x_1, x_2)) = \frac{1}{1+\exp^{-f_\theta(x_1,x_2)}}$.

So using initial values of $\theta_0 = -1, \theta_1 = 1.5$ *and* $\theta_2 = 0.5$ we get linear model as $f_\theta(x_1, x_2) = -1 + 1.5x_1 + 0.5x_2$. So we get the logistic model($P(\hat{y} = 1|x_1, x_2)$) and cross-entropy loss(L_θ) as follows:

$$P(\hat{y} = 1|x_1, x_2) = \sigma(f_\theta(x_1, x_2)) = \frac{1}{1+\exp^{-f_\theta(x_1,x_2)}} = \frac{1}{1+\exp^{(1-1.5x_1-0.5x_2)}}$$

$$L_\theta = -\sum_{i=1}^{n} \left[y_i \log(P(\hat{y} = 1|x_1, x_2)) + (1 - y_i) \log(1 - P(\hat{y} = 1|x_1, x_2)) \right]$$ where $P(\hat{y} = 1|x_1, x_2)$ is defined above

5.2 Logistic model after first iteration

I have used stochastic gradient decent(SGD) and after the first iteration, I have got $\theta_0 = -1.01(bias), \theta_1 = 1.578$ *and* $\theta_2 = 0.543$. So using these values we can write the updated logistic regression model as follows:

$$P(\hat{y} = 1|x_1, x_2) = \sigma(f_\theta(x_1, x_2)) = \frac{1}{1+\exp^{-f_\theta(x_1,x_2)}} = \frac{1}{1+\exp^{(1.01-1.578x_1-0.543x_2)}}$$

5.3 Accuracy, precision and recall

My model converged after **3003 epochs** and final θ values are $\theta_0 = -13.723(bias), \theta_1 = 19.862$ *and* $\theta_2 = 5.162$. Predicted results for the 6 samples in test dataset is **[1, 1, 0, 1, 1, 1]**.

Accuracy, Precision and Recall are **0.666**, **0.6** and **1.0** respectively after evaluating on test data. As we can see in figure 2 also that first two test data points(red ones) are classified wrongly as the two points are in the wrong side of the decision boundary.

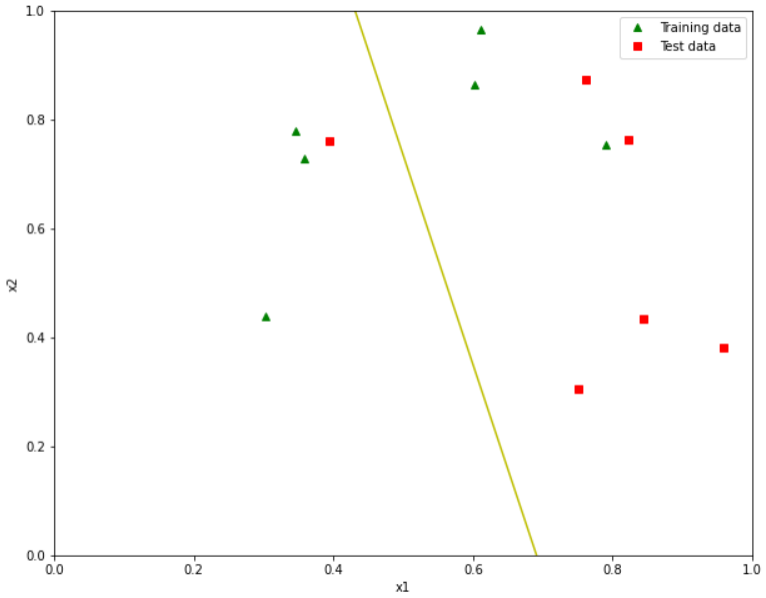

Figure 2: Logistic regression decision boundary

6 Kaggle - Taxi Fare Prediction

In this problem we have to predict the taxi fairs in New York city. It is regression problem and have **55M training samples but only 7 features**, so proper data preprocessing and analysis becomes crucial. The test data has

around 10K samples to predict and the evaluation metric used is **root-mean-square error**.

The **best 2** RMSE score that I have got are using **Random forest** and **LightGBM** which are **3.151** and **3.287** respectively(prediction results are submitted in .csv file). Interestingly **the best 3 models(Random forest, LightGBM and XGBoost)** that achieved the best RMSE scores are **all ensemble methods**.

6.1 Data preprocessing

6.1.1 Data cleaning

I have only preprocessed **20M training samples** out of 55M as handling large dataset is difficult and processing time increases significantly. There were around **139 missing values** in training data and no missing value in test data. So I have removed the rows for these 139 missing values as we have quite large training set.

Negative fairs are not realistic so I have removed the entries where "fare_amount" is less than 0.

There are many rides where **"passenger_count" is 208** which is also not realistic for a taxi, so I have removed these rows.

We know that **latitude** is always in the range between -90 and +90, similarly **longitude** is always in the range between -180 and +180. I have removed the records having any other values for **"pickup_latitude", "pickup_longitude", "dropoff_latitude" and "dropoff_longitude"**.

6.1.2 Feature engineering

I have extracted the features **"Year", "Month", "Date", "Day of Week" and "Hour"** information from "pickup_datetime" column as date-time information is hard to directly digest for the models. Then finally have **removed "key" and "pickup_datetime" column**.

Additionally I have created a **"Distance" column** from pickup and drop off coordinates given using **"Haversine distance"** as distance would be an important contributing factor to the taxi fair. After all these preprocessing, we have **11 features** in the final dataset to train on.

6.2 Evaluating different models

6.2.1 Using simple Linear Regressor

Using simple linear regressor from sklearn and **10M training samples**, I have got a RMSE of **9.326** on test data. Clearly a simple linear regressor could not capture the complex pattern of this problem using linear model.

6.2.2 Using Polynomial Regressor

Using linear regressor from sklearn with polynomial features of **degree 3** and **1M training samples**, I have got a RMSE of **4.972** on test data. Polynomial regressor worked way better than the linear regressor even with just 1M training samples. I could not try higher degrees and with more training samples as training was taking a lot of time.

6.2.3 Using Support Vector Regressor(SVR)

Using a support vector regressor(SVR) from sklearn with **RBF kernel** and with **only 10K training samples**, I have got a RMSE score of **only 9.824**. It is even worse than linear regressor, possible that the model is overfitting and need proper tuning of C and ϵ. Small number of training samples can also impact.

6.2.4 Using Random Forest Regressor

Using a Random forest regressor from sklearn, I have got the **best results** among all. I have made 3 different submissions with modifications on hyper-parameters and number of training samples, best RMSE that I have got is **3.151** with **5M training samples**.

Random forest being an **ensemble(bagging)** method, by nature it can handle over-fitting and noise points better than others, that would probably be the main reason for the least error. Also Random forest works well with large dataset and I have used **200 estimators(quite high and generally higher is better)** in the best submission.

6.2.5 Using XGBoost

Like Random forest, XGBoost(extreme gradient boosting) is another **ensemble method**. Using **5M training samples** and **100 estimators**, I have got a RMSE of **3.381**. This is the **3rd best result** that I have got.

We know that XGBoost works in the additive way and in forward stage-wise manner where weak learners try to reduce the remaining errors(residuals) and that is how it aggregates the results of **multiple learners**. Also weak learners by nature are **immune to overfitting**. XGBoost still didn't perform as good as Random forest and the reason can be the **noise** present in the data.

6.2.6 Using LightGBM

LightGBM stands for **Light Gradient Boosting Machine** originally developed by Microsoft. It is a gradient boosting(so ensemble) framework that uses tree based learning algorithms and one of the main feature of LightGBM is **faster training with the support of parallel, distributed, and GPU learning**.

Using **10M training samples** and **200 estimators**, LGBMRegressor had a RMSE of **3.287** which is the **2nd best** among all. It worked better than XGBoost, may be because I could train with larger training data set.

6.2.7 Using Neural Network

Finally I have tried a **4-layer**(of dimension 11-20-20-1) fully connected feed-forward network with **20M training samples, "Relu" activation(for all layers other than output), "Adam" optimizer and 50 epochs**. I have got the RMSE of **4.276** which is average, not as good as previous 3 ensemble methods discussed.