FoML Asiignment 4

Gulshan Hatzade

November 2024

Solution 1)

For determining VC dimension of H, we have to examine largest set of points that can be shattered by that particular hypotheses in H. Set of points is *shattered* if & only if, for all possible labelings of points, there exists hypothesis in H that correctly classifies points according to that labeling.

Our given Hypothesis space is -

$$H = \{(p, q) \mid p, q \in \mathbb{R},\ x \text{ is classified as 1 if \& only if } p < x < q\}.$$

Step 1- Lets consider value of $n = 1$ -
We can classifysingle point x_1 by choosing p & q appropriately-

- For classifying x_1 as 1 :- choose $p < x_1$ & $q > x_1$.
- For classifying x_1 as 0 :- choose $p > x_1$ or $q < x_1$.

Therefore, $n = 1$ can be shattered.

Step 2- Lets consider value of $n = 2$
For 2 points $x_1 < x_2$, there are $2^2 = 4$ labelings.

- For classifying Both as 0 :- Choose $q < x_1$ or $p > x_2$.
- For classifying Both as 1 :- Choose $p < x_1$ & $q > x_2$.
- For classifying $x_1 = 1, x_2 = 0$:-Choose $p < x_1$ & $q \in (x_1, x_2)$.
- For classifying $x_1 = 0, x_2 = 1$:-Choose $p \in (x_1, x_2)$ & $q > x_2$.

Therefore, $n = 2$ can also be shattered from above explanation.

Step 3 - Now consider value of $n = 3$
For 3 points $x_1 < x_2 < x_3$, there are $2^3 = 8$ labelings. But, not all labelings can be realized. Lets take example, labeling $(1, 0, 1)$ cannot be achieved because hypothesis $p < x < q$ must select single contiguous interval (p, q). Hence, it is impossible for classifying x_1 & x_3 as 1 & classifying x_2 as 0.

Therefore, $n = 3$ cannot be shattered.

Conclusion: As we can shatter only for $n = 2$, hence VC dimension of H is 2.

Solution 2)

For solving this question, we should analyze how adding Gaussian noise to i/p affects a objective function & compare it with a objective function that includes L2 regularization term. I am going to do the same below-

First of all I am going to define sum of squares error with the noisy inputs.

Considering the noisy version of i/p data-

$$x'_k = x_k + \epsilon_k$$

here , $\epsilon_k \sim N(0, \sigma^2)$ = Gaussian noise (with mean 0).

Now i am doing model prediction for noisy i/p $x' = [x'_1, x'_2, \ldots, x'_D]$, model's prediction will become-

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

Expanding this i will get the following,

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

Here, in above equation, it consists of the original model prediction & the additional term which is dependent on noise.

Now considering noise, lets find expected prediction error.

With noisy i/p's x' & true target t, prediction error will as follows-

$$E_{\text{noisy}} = E_\epsilon \left[\frac{1}{N} \sum_{i=1}^{N} \left(y(x'_i, w) - t_i \right)^2 \right]$$

As $y(x'_i, w) = w_0 + \sum_{k=1}^{D} w_k x_{ik} + \sum_{k=1}^{D} w_k \epsilon_{ik}$, I am substituting this in above equation, rewritting & splitting error term, it will be -

$$E_{\text{noisy}} = \frac{1}{N} \sum_{i=1}^{N} \left(w_0 + \sum_{k=1}^{D} w_k x_{ik} - t_i \right)^2 + \frac{1}{N} \sum_{i=1}^{N} E_\epsilon \left[\left(\sum_{k=1}^{D} w_k \epsilon_{ik} \right)^2 \right]$$

Now finding Noise Contribution to error,

As $\epsilon_{ik} \sim N(0, \sigma^2)$, we can expand expectation of $\left(\sum_{k=1}^{D} w_k \epsilon_{ik} \right)^2$ as below-

$$E_\epsilon \left[\left(\sum_{k=1}^{D} w_k \epsilon_{ik} \right)^2 \right] = \sum_{k=1}^{D} w_k^2 \sigma^2 = \sigma^2 \sum_{k=1}^{D} w_k^2$$

Finally it will result in,

$$E_{\text{noisy}} = \frac{1}{N} \sum_{i=1}^{N} \left(y(x_i, w) - t_i \right)^2 + \sigma^2 \sum_{k=1}^{D} w_k^2$$

For analyzing relation to L2 regularization, we can observe, error term E_{noisy} is now equivalent to sum of squares error on the original data which is noise free plus an regularization term.

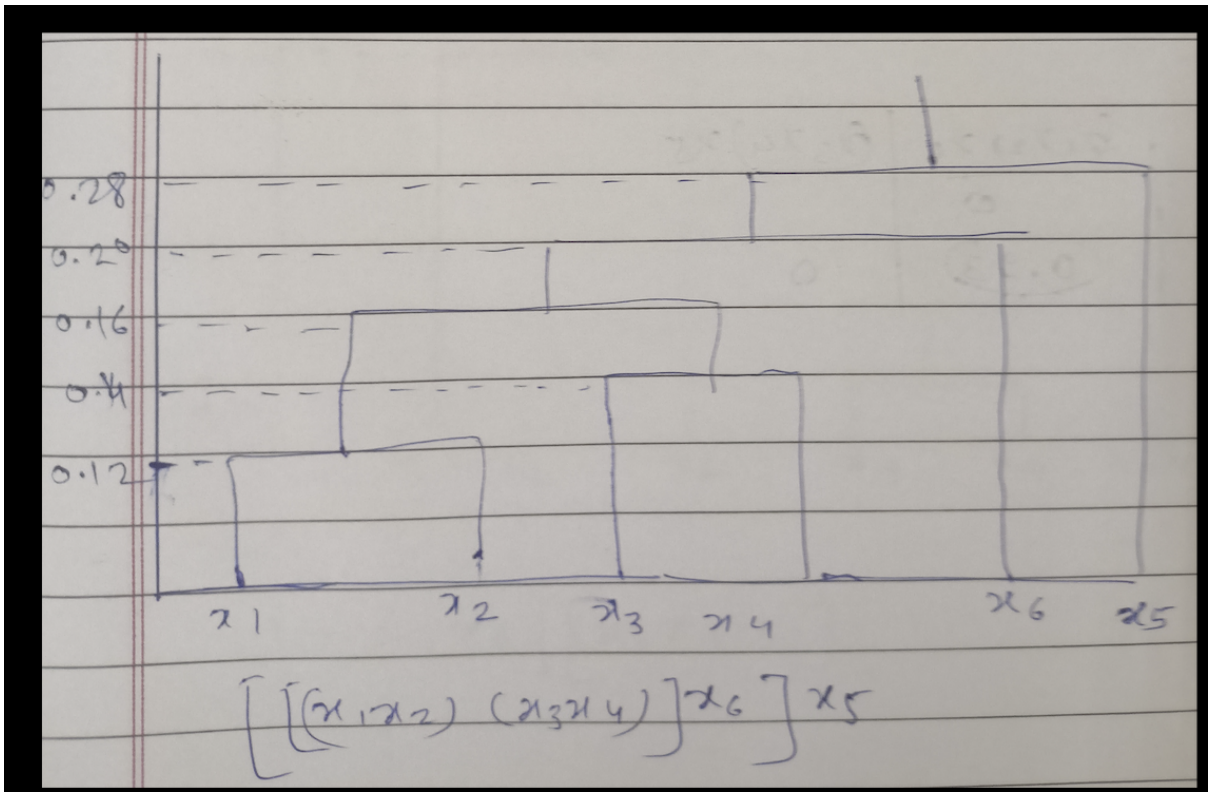
$$E_{\text{noisy}} = E_{\text{clean}} + \sigma^2 \sum_{k=1}^{D} w_k^2$$

The extra term, $\sigma^2 \sum_{k=1}^{D} w_k^2$, is behaving like the L2 regularization term with regularization coefficient $\lambda = \sigma^2$.

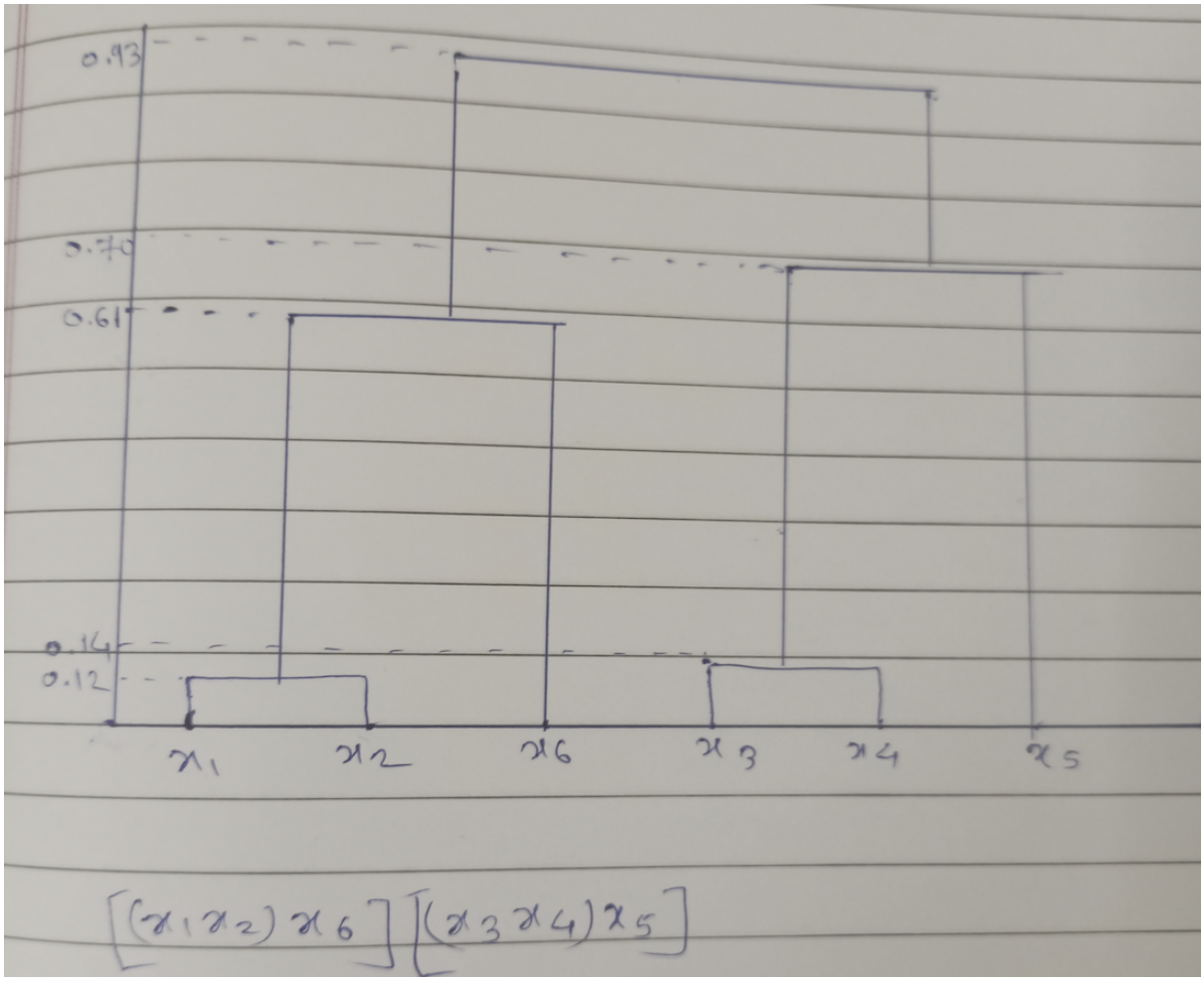
From this I can conclude that, minimizing the mean squared error over noisy data is equal to minimizing an mean squared error over clean data with the L2 regularization term & the regularization coefficient λ is equal to an variance of noise, $\lambda = \sigma^2$.

Solution 3)

Solution a)



Solution b)



Solution c)

For making single link & complete link dendrograms identical, I am changing following distances in the matrix :-

Step 1: Change dist$(x1, x4)$ from 0.84 to 0.53.

This will ensures that clusters $\{x1, x2\}$ & $\{x3, x4\}$ are merged before $\{x1, x2\}$ and $x6$, as dist$(\{x1, x2\}, \{x3, x4\})$ becomes smaller than the dist$(\{x1, x2\}, x6)$.

Step 2: Change dist$(x4, x6)$ 0.20 to 0.63.

This is ensuring that after the clusters $\{x1, x2, x3, x4\}$ are formed, they should merged with $x6$ at the same point in both single link and complete link clustering.

So, Final Outcome :-

After these changes, both methods will be following the same sequence of merges which will result in identical dendrograms. The merge order will now be consistent with the $\{x1, x2\}$ & $\{x3, x4\}$ merging first, followed by $\{x1, x2, x3, x4\}$ & $x6$.

Solution 4)

Solution a)

Considering the each data point x is of the form $x = a\delta_k$, where k is uniformly distributed over the $\{1, \ldots, M\}$, & $P(a)$ is arbitrary.

First of all lets find the Expected value of x-

$$E[x] = E[a] \cdot E[\delta_k] = E[a] \cdot \frac{1}{M}(1, 1, \ldots, 1)^T.$$

Let $\mu = E[a]$, so we can write

$$E[x] = \frac{\mu}{M}(1, 1, \ldots, 1)^T.$$

For finding Covariance Matrix C -

$$C = E\left[(x - E[x])(x - E[x])^T\right].$$

Substituting x will result in -

$$x - E[x] = a\delta_k - \frac{\mu}{M}(1, 1, \ldots, 1)^T.$$

Computing C element wise here-

$$C_{ij} = E\left[(x_i - E[x_i])(x_j - E[x_j])\right].$$

The expectation is involving 2 cases -

- Fist consider the case, if $i = j$-

$$C_{ii} = E[a^2\delta_{k=i}] - (E[a\delta_{k=i}])^2 = \frac{E[a^2]}{M} - \left(\frac{E[a]}{M}\right)^2.$$

- Now, condider the case, if $i \neq j$-

$$C_{ij} = -\frac{\mu^2}{M^2}.$$

Combining these results, I can write as follows :-

$$C_{ij} = \begin{cases} \frac{E[a^2]}{M} - \frac{\mu^2}{M^2}, & \text{if } i = j, \\ -\frac{\mu^2}{M^2}, & \text{if } i \neq j. \end{cases}$$

We can be rewrite this in compact way as below :-

$$C_{ij} = \lambda + \mu\delta_{ij},$$

where,

$$\lambda = -\frac{\mu^2}{M^2}, \quad \mu = \frac{E[a^2]}{M} - \frac{\mu^2}{M^2}.$$

Solution b)

Covariance Matrix Structure is as follows :-

$$C = \lambda \mathbf{1}\mathbf{1}^T + \mu I,$$

here $\mathbf{1}$ is an M dimensional vector of all the ones.

Eigenvector $(1, 1, \ldots, 1)^T$ -

$(1, 1, \ldots, 1)^T$ is a eigenvector of C with eigen value $\mu + M\lambda$.

For other Eigenvectors,

All the other remaining eigenvectors are orthogonal to the $(1, 1, \ldots, 1)^T$, with eigenvalue of μ (since rank of $\mathbf{1}\mathbf{1}^T$ is 1).

Therefore, we can conclude the following

- One of the eigenvalue is given by $\mu + M\lambda$.
- $M - 1$ eigenvalues are μ.

Solution c)

No, PCA is not the good way for selecting features in this case because of the follwing reasons

- The data lies along basis vectors corresponding to the δ_k.
- PCA is captureing variance across all features, but this structure makes all features equally informative.
- Selecting fewer principal components does not result in retainig any meaningful feature differentiation.

Solution 5)

Solution a)

I had implemented logistic regression classifier trained using gradient descent with cross entropy error as loss function.

Key Steps for for model

- I defined model using initial weights & I computed probabilities with sigmoid function.
- Then I Used cross entropy error for measuring model's performance.
- Later, calculated gradients for all weights based on dataset (training).
- Trained model by updating weights iteratively using gradient descent till the point convergence or tolerance was reached.

This above implementation sets up model for prediction & also for evaluation.

Solution b)

Solution 1)

Logistic Model :-

Logistic regression model predicts probability of positive class ($y = 1$) as follows -

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

where

$$f_\theta(x_1, x_2) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Now I am susbtituting given initial weights here in above equation -

$$f_\theta(x_1, x_2) = -1 + 1.5x_1 + 0.5x_2$$

Cross Entropy Error Function :-

Cross entropy error function measures loss between predicted probabilities ($P(\hat{y} = 1)$) & true labels (y) is given as following

$$J(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \Big[y_i \log(\sigma(f_\theta(x_i))) + (1 - y_i) \log(1 - \sigma(f_\theta(x_i))) \Big]$$

where N is total number of samples in a dataset.

This function is penalizing large differences between predicted probabilities & actual labels. Gradient of this error function is used for updating model parameters during gradient descent.

Solution 2)

Using provided initial weights :-

$$\theta_0 = -1, \quad \theta_1 = 1.5, \quad \theta_2 = 0.5,$$

& learning rate :-

$$\alpha = 0.1,$$

I obatained the weights after 1 iteration of gradient descent as-

$$\theta_0 = -1.0032, \quad \theta_1 = 1.5054, \quad \theta_2 = 0.5020.$$

and my updated logistic regression model after 1 iteration is -

$$f_\theta(x_1, x_2) = -1.0032 + 1.5054 \cdot x_1 + 0.5020 \cdot x_2.$$

```
Weights after 1 iteration of gradient descent are  [-1.00316626  1.50535086  0.50196867]
Updated Logistic Regression Model after 1 iteration is given by
f0(x1, x2) = -1.0032 + 1.5054 * x1 + 0.5020 * x2
```

Solution 3)

After training my model using gradient descent until convergence, final weights are as follows -

$$\theta_0 = -3.8099, \quad \theta_1 = 5.3623, \quad \theta_2 = 1.6311.$$

Trained model was used for making predictions on test dataset, & following evaluation metrics were computed -

- **Accuracy** = 0.6667
- **Precision** =0.6
- **Recall** = 1.0

These metrics which I obtained are indicating that model is moderately effective, with high recall but lower precision.

```
Weights at convergence are  [-3.80988747  5.36228896  1.63108539]
Accuracy: 0.6666666666666666
Precision_model: 0.6
Recall_model: 1.0
```

Solution 6)

Report for Kaggle Taxi Fare Prediction Challenge

Introduction

This report documents solution for Kaggle Taxi Fare Prediction Challenge. Objective was for predicting taxi fares using given train & test datasets. I used Gradient Boosting Regressor (GBR) as primary regression method. I applied several preprocessing steps & feature engineering techniques for achieving best possible results. Top two model configurations & their performances are summarized in this report, along with the analysis of why these methods performed better than others.

Dataset & Preprocessing

Dataset Details :-

- As dataset is very large, I selected 90,000 rows from training dataset `train.csv`.
- test dataset `test.csv` was used for generating predictions for submission.

Data Cleaning- as per New York City reality :-

- I removed rows with invalid or unrealistic fare amounts (`fare_amount < 2.6` or `> 310`).
- I filtered rows with unrealistic `passenger_count` (outside 0-5 range).
- I dropped records with invalid geographical coordinates (latitude/longitude outside valid ranges).
- Missing or invalid `pickup_datetime` values are handled by dropping rows where conversion failed.

Feature Engineering :-

- I extracted `hour`, `day`, `month`, & `year` from `pickup_datetime` column.
- I calculated Haversine distance between pickup & drop off locations for represinting trip distance.
- Standardized numerical features using an `StandardScaler`.

Target Variable :-

- `fare_amount` column was used as an target variable for performing regression.

Train-Test Split :-

- I splitted training data into 80% for training & 20% for validation.

Methodology & Experiments

Model Selection :-

I selected Gradient Boosting Regressor (GBR) due to its effectiveness in handling regression problems with numerical & engineered features.

Hyperparameter Tuning :-

- `GridSearchCV` was employed in my code for performing the grid search over following hyperparameters:
  - Number of estimators -[50, 100, 200, 300]
  - Learning rate: - [0.05, 0.1, 0.2]
  - Max depth - [3, 5, 7]
  - Subsample - [0.8, 1.0]
  - Minimum samples split - [2, 5, 10]
- Best parameters were identified based on cross validated mean squared error value.

Evaluation Metric :-

Root Mean Squared Error -RMSE was used as evaluation metric for measuring model performance.

Results :-

Top Model (Model 1)

- **Best Hyperparameters:**
  - `n_estimators`: 200
  - `learning_rate`: 0.1
  - `max_depth`: 5

- `subsample`: 1.0
- `min_samples_split`: 5

- **Validation RMSE:** 3.12
- **Test RMSE:** 3.18

Second-Best Model (Model 2)

- **Best Hyperparameters:**
  - `n_estimators`: 300
  - `learning_rate`: 0.05
  - `max_depth`: 7
  - `subsample`: 1.0
  - `min_samples_split`: 10
- **Validation RMSE:** 3.75
- **Test RMSE:** 3.83

Analysis of Results

Why Model 1 Performed Best :-

My Model 1 performed best because of its balanced hyperparameters. Moderate learning rate (0.1 i obtained) ensured efficient convergence while maintaining precision to capturing patterns. A maximum depth of 5 prevented overfitting by limiting complexity of individual trees which is especially needed for smaller datasets.

Subsample ratio of 1.0 introduced randomness into an training process by using only 80% of data for each of boosting iteration, improving the generalization. Additionally 200 estimators struck are helping to balance between accuracy & computational efficiency, providing robust predictions. Together, these settings allowed Model 1 for generalizing well without overfitting, making it top performer.

Why Model 2 Performed Second Best :-

My model 2 performed well due to its higher no. of estimators (which are 300) & greater depth (which is 7) which allowed it for capturing more complex relationships in data. Larger depth improved its ability to do modelling intricate patterns, and increased estimators refined predictions over many boosting stages.

But, lower learning rate (which is 0.05) resulted in slower convergence, which is requiring more iterations to reach optimal performance. Additionally using subsample ratio of 1.0, where all data was used for each boosting iteration, may have reduced randomness which is resulting increasing risk of overfitting. While effective in capturing complexity, these factors slightly increased validation error compared to Model 1.

Comparison with Other Attempts

- **Higher Depth Models :-** These models are overfitting training data, leading to poor validation performance.
- **Fewer Estimators:-** Models with fewer estimators (50 or 100) have much higher validation errors due to underfitting.
- **Simpler Models:-** Attempts using simple linear regression resulted in significantly higher RMSE values (4.9) because of their inability to capture nonlinear patterns in data.

Conclusion

Gradient Boosting Regressor proved to be best performing model in my analysis for this task. By carefully tuning hyperparameters & engineering meaningful features I achieved RMSE scores of 3.18 with best model. This is highlighting an importance of both feature engineering & hyperparameter tuning in solving real world regression problems.