Assignment-9
Model Selection & Optimization

Harshvardhan Patidar
Department of Artificial Intelligence
Indian Institute of Technology Hyderabad
ai24btech11015@iith.ac.in

The aim of Model Selection & Optimization is to select a hypothesis which will fit or generalize well to future data. We assume that the future examples will be simpler to the given past data, called the stationarity assumption. The goal is to select such a hypothesis that minimizes the error rate, or the ratio of $h(x) \neq y$ for some (x, y) to total samples.

For better selection of the hypothesis, the data set is split into three parts:

1. **Training set** : for training the model
2. **Validation set** : to evaluate the different models and choose the best performing one.
3. **Test set** : for final unbiased evaluation of the final model.

But in cases where the data is limited, we can use the k-fold cross-validation technique. In this, the data is divided into k subsets. We then train the model k times, each time holding out one of the subset for validation purposes. The average score of these k folds is then used to estimate the model's performance. Generally, we take k to be 5 or 10. The most extreme case, when $k = n$ is called leave-one-out cross-validation or LOOCV.

Model Selection

To choose the best model we can use a simple algorithm which takes input a learner algorithm, which takes a hyperparameter, say size. For polynomials, size can be degree. The algorithm starts with the smallest value of size, and slowly increases it to obtain more complex models. In the end, the algorithm selects the model that has the lowest average error rate on the validation data which was taken aside.

Error Rates to Loss

This section discusses ways to classify errors according to how severe they are. For example, It's better to mark a spam mail as a non-spam mail rather than marking a non-spam as a spam mail. So, a program which classifies with a higher error rate but most of the errors are of marking spam as non-spam, is definitely way better than a model with lower error rate but, most of them being marking non-spam as spam, which is unfeasible. For this case, we can define a loss function, in which we can deduct 10 points for marking non-spam as spam compared to deducting only 1 point for marking a spam as a non-spam.

Instead, if we have real valued outputs, we can consider

1. $\mathbf{L_1}(\mathbf{y}, \hat{\mathbf{y}}) : L_1(y, \hat{y}) = |y - \hat{y}|$
2. $\mathbf{L_2}(\mathbf{y}, \hat{\mathbf{y}}) : L_1(y, \hat{y}) = (y - \hat{y})^2$

where $L(y, \hat{y})$ is the loss function

For the case of discrete-valued outputs, we can use

$$\mathbf{L_{0/1}}(\mathbf{y}, \hat{\mathbf{y}}) = 0 \text{ if } y = \hat{y}, \text{ else } 1$$

The learning agent chooses the best hypothesis that minimizes the expected loss over all the data. That is it tries to minimize its generalization loss. We need a probability distribution $\mathbf{P}(X, Y)$ to calculate it. The generalization loss is given by:

$$\text{GenLoss}_L(h) = \sum_{(x,y)\in E} L(y, h(x)) \cdot P(x, y)$$

and our hypothesis h^* in this case would be

$$h^* = \arg\min_{h\in H} \text{GenLoss}_L(h)$$

But in case if we don't have the probability distribution $\mathbf{P}(X, Y)$, then we can use empirical loss which is given by

$$\text{EmpLoss}_{L,E}(h) = \frac{1}{N} \sum_{(x,y)\in E} L(y, h(x))$$

And h^* will be

$$\hat{h}^* = \arg\min_{h\in H} \text{EmpLoss}_{L,E}(h)$$

The difference in h^* from the true function f is mainly due to following reason:

1. **Unrealizability** : It is possible that our true function f is not present in our selected hypothesis space $\mathcal{H}$.
2. **Variance** : if unrealizability isn't an issue, then only after training over a very big data is that variance becomes low, else the function would be mostly overfitting.
3. **Noise** : There might be noise in the function f, which means it might output different values for the same x.
4. **Computational intractability** : It might be also the case that $\mathcal{H}$ is very large and that we are only going through a small portion of the space but not complete. In this case, its not guaranteed that the found hypothesis h is the best one.

Traditionally we used to do small-scale learning, in which the training set used to be comparatively smaller, because of which 1^{st} and 2^{nd} type of errors were prominent. But nowadays, since we have very large data sets, the 3^{rd} and 4^{th} type of errors are more common as we don't have enough computational power to go through all of the hypothesis space.

Regularization

Another way to find our hypothesis h is to use the method of regularization, which tries to minimize a cost function, which depends upon empirical loss and the complexity of the hypothesis.

$$\text{Cost}(h) = \text{EmpLoss}(h) + \lambda\text{Complexity}(h)$$
$$\hat{h}^* = \arg\min_{h\in\mathcal{H}} \text{Cost}(h).$$

Here λ is a hyperparameter which works as a conversion rate between complexity and loss. This process of explicitly penalizing complex hypotheses is called Regularization. We try to keep our model midway being too simple and too complex. Being simple is not feasible as it might cause it to face underfitting, or not being able to find patterns in data. Whereas, being complex is not feasible as this might make the model memorize the data, hence decreasing the performance of the model on the unseen data.

One more notable way to reduce the complexity is feature selection. We try to reduce the number of attributes we provide as input by discarding the non-important ones. The Minimum Description Length (MDL) principle says that the hypothesis which can be expressed in the least number of bits in its Turing code is the most efficient hypothesis.

Hyperparameter Tuning

This is the process of selecting the best value of the hyperparameter in supervised learning. There are few methods for achieving this depending on the data set and the model being used.

1. **Hand Tuning** : It is the simplest method of hyperparameter tuning. Choose it by your prior knowledge, check its error rate, guess a new parameter value, again check its error rate, and repeat till you get the best result.
2. **Grid Search** : Useful in cases where you have only a few hyperparameters each with a small number of possible values. In this approach, all combinations of values are tried and the best performing hypothesis is selected.
3. **Random Search** : it's a good way to handle continuous values. A value is randomly picked up from the sample until you are willing to spend time and computational resources.
4. **Bayesian Optimization** : In this method, we consider hyperparameter tuning also as a machine learning task. The model learns from its previous choice of hyperparameter and tries to choose a better hyperparameter in the next iteration. It makes a tradeoff between
   (a) Exploitation : Trying close values to previous good results.
   (b) Exploration : Trying new values.
5. **Population-based Training (PBT)** : In this method, we train a few models each with varying hyperparameter values. Then, we train a second generation of models, whose hyperparameter values are based on the previous successful values. This way we get the best model.