Name: Md. Mubasir

Email address: talk2mubasir0587@gmail.com

Contact number: 9831958386

Anydesk address: 151 986 252

Date: 16th Jul 2020

Self Case Study -1: Don't Overfit! II

Overview

One of the main objectives of predictive modelling is to build a model that will give accurate predictions on unseen data which will only be possible when we make sure that they have not overfitted the training data. Before going to understand how to ensure that our model hasn't overfitted training data, let's first get aware of the reasons which lead the model to get overfitted.

There are many reasons for the same, yet we would like to point out some major reasons. First, being of fewer data points in training samples, second is the dataset being imbalanced, and last but not the least, complex nature of model.

In this case study, we are going to handle the same problem of overfitting and the challenging part is to make a model with 250 data points in the train set and predict the binary target accurately for 19750 unseen data points in the test set. The dataset has 300 features of a continuous variable.

The dataset is obtained from the link: https://www.kaggle.com/c/dont-overfit-ii/data

Files

- train.csv - the training set. 250 rows.
- test.csv - the test set. 19,750 rows.
- sample_submission.csv - a sample submission file in the correct format

Columns

- id- sample id
- target- a binary target of mysterious origin.
- 0-299- continuous variables.

Evaluation Metric: AUROC

Research-Papers/Solutions/Architectures/Kernels

Underfitting and Overfitting in machine learning and how to deal with it !!!: (Blog:https://towardsdatascience.com/underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6fe4a8a49dbf):

In this blog, an overfitting Machine Learning model as a *subject matter expert* is proposed, which gives the correct results only to those data points by which it has been trained. Also, there are several approaches listed to tackle overfitting.

Firstly, during the learning of a model, *Cross-Validation* of data was used to ensure that the model is not getting overfitted. If we are training the Neural Network model, we should stop the training as soon as we observe that the performance of cross-validation data is not upgrading. This is called *Early Stopping*.

Adding *Regularization* term into the cost function will help to minimize the weight of less important features and prevent the model from overfitting.

The model which is trained with data points having a large number of features and fewer numbers of rows is prone to overfit. To avoid this, we can select only a set of important features and train models on these features only. This is called *Features-Selection*.

An Overview of Overfitting and its Solutions: (Paper:https://iopscience.iop.org/article/10.1088/1742-6596/1168/2/022022/pdf):

This paper presented some of the heuristic techniques to avoid overfitting in great detail and gave reasons for how and when to apply them.

Network-reduction:

We can reduce the complexity of a model by eliminating the less significant and irrelevant data (i.e. noise), which will help the model to prevent overfitting and perform well on unseen data. This is called *pruning,* which is widely used in the decision tree algorithm. There are two types of pruning methods that have been proposed: one is *pre-pruning* and the other is *post-pruning.*

The pre-pruning method try to stop the tree-building process early before it produces leaves with very small samples. At each stage of splitting the tree, we check the cross-validation error. If the error does not decrease significantly enough, we stop. This is also called Early-Stopping in the learning algorithm.

Post-pruning involves cutting back the tree. After a tree has been built, it is pruned back to the point where the cross-validated error is minimum.

Expansion of the training data:

As the limited data leads the model to overfit, we should expand out our data which might help the model to avoid overfitting. The techniques that have been proposed in this paper include adding some random noise to an existing dataset, Re-acquiring some data from existing data set through some processing, and Producing some new data based on the distribution of the data set.

Regularization:

A large number of features makes the model more complex. Instead of discarding some features arbitrary, if we add a regularization term into cost function, it will cut down the effect of useless features by minimizing their weights. L1 regularization term sets the weight of the less important feature to zero whereas L2 regularization makes that type of weight too small.

Overfitting in Making Comparisons Between Variable Selection Methods: (Paper:http://www.jmlr.org/papers/volume3/reunanen03a/reunanen03a.pdf):

This paper has covered two major methods of feature-selection, namely sequential forward selection (SFS) and sequential forward floating selection (SFFS). The results obtained on the sonar and the mushroom dataset were covered, which consequently showed that SFFS produces a better result than SFS. In the mushroom dataset, SFFS gave better results on test data even by half of the features, as compared to a number of features selected by SFS.

Sequential Forward Selection:

In this method, the best single feature was first selected and paired with unselected features, and the best pair features were selected again. Again, a triplet with the best pair and unselected features were made, and then the best triplet features were selected. This process went on until the desired number of features got selected.

The drawback of this method is that, once a feature gets selected, we can't remove it even if we could get good results by another, comparatively smaller subset of features. This limitation does not occur in SFFS.

Sequential Forward Floating Selection:

SFFS starts from an empty set. After each forward step, it performs a backward step as long as the objective function keeps on upgrading. The crux of this approach is to use a backtracking algorithm to select the best and the smallest subset of features.

The downside of this method is that it is expensive in terms of computation.

Generalization Improvement in Multi-Objective Learning: (Paper:https://static.aminer.org/pdf/PDF/000/265/793/a_noise_metric_on_binary_training_inputs_and_a_framework.pdf):

Unlike the above strategies to handle overfitting, this paper has mentioned the heuristic method called the *noise injection(jitter)* to get rid of such problems. In this paper, it has been justified that SO (single-objective, e.g. Square mean error) algorithm is much more susceptible to overfitting than MO (multi-objective e.g. ROCAUC) algorithm. This is because, in ROCAUC, the cost function algorithm tries to maximize TPR and minimize FPR simultaneously which works as a regulator and resists it to get overfitted. The second important thing that has been mentioned, is that the neural networks are more robust towards small changes in training patterns. Thus, perturbation of training data by injecting noise and training such data with neural nets make the model overfit free and perform better on test data.

Following are two approaches to perturbing training data which are mentioned in this paper:

Additive Gaussian Noise:

In each generation, a new artificial training data set is generated by adding a normal distributed random number to each of the elements of the input.

Adding Noise by Means of Averaging:

In this approach, new artificial patterns are generated by averaging the original patterns within a local neighbourhood.

Regularization Method: Noise for improving Deep Learning models: (Blog:https://towardsdatascience.com/noise-its-not-always-annoying-1bd5f0f240f):

This blog has covered the Keras implementation of the fitter(noise-injection) approach and performed grid-search to know the best amount of noise for the model. Firstly, 100 binary classification data points were synthesized using sklearn. Then, the train set was made 30% and the test set 70% to forcibly overfit the model.

A baseline model without the fitter was made at first, so as to compare with the fitter model later, that got 100% train accuracy and 50% test accuracy. Then, 3 kinds of architectures were made. The first architecture kept the Gaussian noise layer as the input layer. The second one created a hidden layer as a noise layer, and in the last architecture, both the input noise layers and hidden noise layers were added. The three architectures resulted in the test accuracy of 64%, 67%, and 68% respectively. Eventually, a

grid-search was performed that resulted in the best noise value of 0.001, and the test accuracy of 83% with hidden layer noise architecture.

Just Don't Overfit: (Blog:https://medium.com/analytics-vidhya/just-dont-overfit-e2fddd28eb29):

It used the same problem as that of ours. The crux approach of this blog is to use LASSOCV (Least Absolute Shrinkage and Selection Operator Cross-Validation). It is made up of two terms LASSO and CV (where CV is Cross-Validation). The main function of the LASSO regressor is to add some bias term to minimize the variance of the model. It performs L1 regularization, i.e. it adds a penalty equivalent to an absolute value of the magnitude of coefficients. Its objective function is:

$$\text{Minimization objective} = \text{LS Obj} + \alpha * (\text{sum of absolute value of coefficients})$$

Where, the only parameter is alpha. More the value of alpha, more the feature's weight gets to zero.

Use of this algorithm lead to a score of 8.43 on the Kaggle leader board.

Don't Overfit! — How to prevent Overfitting in your Deep Learning Models: (Blog:https://towardsdatascience.com/dont-overfit-how-to-prevent-overfitting-in-your-deep-learning-models-63274e552323):

This blog has tried to train a deep neural network model to avoid the overfitting of the same dataset we have. First, a feature selection using RFE (Recursive Feature Elimination) algorithm is performed. Then, a simple NN model with 2 hidden layers and 2 dropout layers with early stopping is trained, that resulted in 80% test accuracy.

First Cut Approach

There are several methods that have already been applied, giving some results. Yet, out of many relatable techniques that we learnt from various research papers; we would use the important conceptual ideas that'd guide us in our project. Our approach is categorized into two approaches: first one being machine learning approach, and another being the neural network approach.

Machine Learning Approaches:

ML Baseline Model:

Our baseline model would be a Logistic Regression model with all the features, without applying anything else. The purpose of constructing this model is to juxtapose with the models that would be tried to improve later on.

First Approach: Random Sampling

As we can observe, along with the limited data points in the train set, it is also decently imbalanced, with the ratio of 9:16. Thus, our first and foremost approach would involve balancing the dataset by combining Random Oversampling and Under sampling, which might consequently give an interesting result.

A modest amount of oversampling could be applied to the minority class to improve the bias towards these examples, whilst applying a modest amount of under sampling to the majority class as well, so as to reduce the bias on that class. This could result in an improved overall performance as compared to performing one or the other techniques in isolation.

For example, if we have a dataset with a 9:16 class distribution, we might first apply oversampling to increase the ratio to 3:4 by duplicating examples from the minority class, following with the application of under sampling to further improve the ratio to 1:1 by deleting examples from the majority class.

This could be implemented using an imbalanced-learn by using a ***RandomOverSampler*** with sampling_strategy set to 0.75 (75%), followed by a ***RandomUnderSampler*** with a sampling_strategy set to 1.0 (100%)

We would compare the performance of this model with the baseline model.

Second Approach: Feature Selection

In the train set, we have 300 features that are quite a lot for few training samples. Thus, instead of using all the features, it would be better to use only the most important ones. This, on one hand, would make the training process notably faster, and on the other hand, it might help to prevent overfitting because the model doesn't need to learn as many features.

As we have learned that SFFS works better than SFS, here we would throw a light on both of them. There is a python library *mlxtent* that contains the function **SequentialFeatureSelector** which would work as an SFFS if the floating parameter is true. Else, it works as SFS.

There are filter methods of feature selections that could be used. The scikit-learn library has provided many feature selection algorithms through feature_selection modules like **SelectKBest, SelectPercentil, chi2, f_classif, RFE, etc.** These algorithms would help us to select the best features. We would train the ML model on top of that and check the performance. Finally, the resulting score would be compared with the model based on the original dataset.

Third Approach: Feature Engineering

Feature engineering is an essential part of building an intelligent system. It is an art as well as a science, which is the reason Data Scientists often spend 70% of their time in the data preparation phase before modelling. As prof. Andrew Ng. said "Coming up with features is difficult, time-consuming, requires expert knowledge. 'Applied machine learning' is basically feature engineering." That is why we should apply some feature engineering to our project.

As we know that all features values are continuously distributed, we can convert continuous data into discrete data by using the *binning* technique. We would do fixed-width and adaptive binning for some of the features that are skewed in nature.

Neural Network Learning Approaches:

NN Baseline Model:

We would construct a simple model with two hidden layers and 8-6 neurons in each layer without any dropout layers. This would be done just for the comparison of the performance with the other better models.

First Approach: Dropout layers and Weight decay(i.e regularizer)

In this approach, we would try to make the baseline architecture quite complex by adding dropout layers, regularizer term, and bias term to the layers to resist it to the overfitting case.

Second Approach: Injecting Noise (Fitter)

This approach would be based upon the research paper that is mentioned above. We would add the Gaussian noise hidden layer and search for the best noise value through grid-search, followed by training with some dropout layers.
