Statistical & Machine Learning

Individual Project Report

By Inderpreet Rana

31st March 2022

Contents

1) Logistic Regression

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable. The outcome or target variable is a binary variable and can have only two potential classes. It calculates the probability of an event occurring.

For two-class classification, Logistic Regression is one of the simplest and widely used Machine Learning methods. It's simple to set up and may be used as a starting point for any binary classification task. Its fundamental concepts are also constructive in deep learning. Logistic regression describes and estimates the relationship between one dependent binary variable and independent variables.

It's a special case of linear regression in which the target variable is categorical. The dependent variable is the log of odds. Logistic Regression predicts the probability of occurrence of a binary event utilizing a logit function.

1.1) Logistic Regression Objective Function:

The sigmoid function is the main objective function for this method.

Linear Regression Equation:

$$y = \beta 0 + \beta 1X1 + \beta 2X2 + \ldots\ldots + \beta nXn$$

Where, y is dependent variable and x1, x2 ... and Xn are independent variables.

Now when we apply the sigmoid function on linear regression, it becomes a classification function.

Sigmoid Function:

$$p = 1/1 + e^{-y}$$

Apply Sigmoid function on linear regression and it becomes the logistic regression method:

$$p = 1/1 + e^{-(\beta 0 + \beta 1X1 + \beta 2X2 \ldots\ldots \beta nXn)}$$

Properties of Logistic Regression:

- The dependent variable in logistic regression follows Bernoulli Distribution.
- Estimation is done through maximum likelihood.
- No R Square, Model fitness is calculated through Concordance, KS-Statistics.

Logistic regression is estimated using Maximum Likelihood Estimation (MLE) approach.
The MLE is a "likelihood" maximization method

Maximizing the likelihood function determines the parameters that are most likely to produce the observed data. From a statistical point of view, MLE sets the mean and variance as parameters in determining the specific parametric values for a given model. This set of parameters can be used for predicting the data needed in a normal distribution.

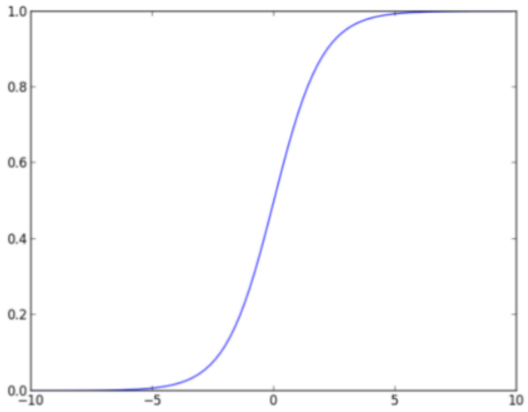
Source: https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python

1.1.1) Sigmoid Function:

The sigmoid function, also known as the logistic function, produces a 'S' shaped curve that can map any real-valued number to a value between 0 and 1. If the curve reaches positive infinity, y predicted becomes 1, and if it reaches negative infinity, y predicted becomes 0.

$$f(x) = \frac{1}{1 + e^{-(x)}}$$

The Sigmoid Function



The Sigmoid Function curve.

1.2) Types of Logistic Regression:

- **Binary Logistic Regression** is used when the target variable has only two possible outcomes, such as spam or no spam, cancer, or no cancer.

- **Multinomial Logistic Regression**: When predicting the type of wine, the target variable has three or more nominal categories.

- **Ordinal Logistic Regression** is used when the target variable has three or more ordinal categories, such as a restaurant or product rating from 1 to 5.

1.3) Advantages:

- Logistic Regression is one of the simplest machine learning algorithms and is easy to implement.
- The predicted parameters (trained weights) give inference about the importance of each feature.
- Logistic Regression proves to be very efficient when the dataset has features that are linearly separable.

1.4) Disadvantages:

- Logistic Regression is a statistical analysis model that attempts to predict precise probabilistic outcomes based on independent features. On high dimensional datasets, this may lead to the model being over-fit
- Logistic Regression requires a large dataset and also sufficient training examples for all the categories it needs to identify.
- It is required that each training example be independent of all the other examples in the dataset. If they are related in some way, then the model will try to give more importance to those specific training variables

2) Random Forest

Random forest is a method for supervised learning. It can be used for both classification and regression. It is also one of the most flexible algorithms. It creates decision trees from various samples and takes their majority vote for classification and average in case of regression. It also provides a pretty good indicator of the feature importance.

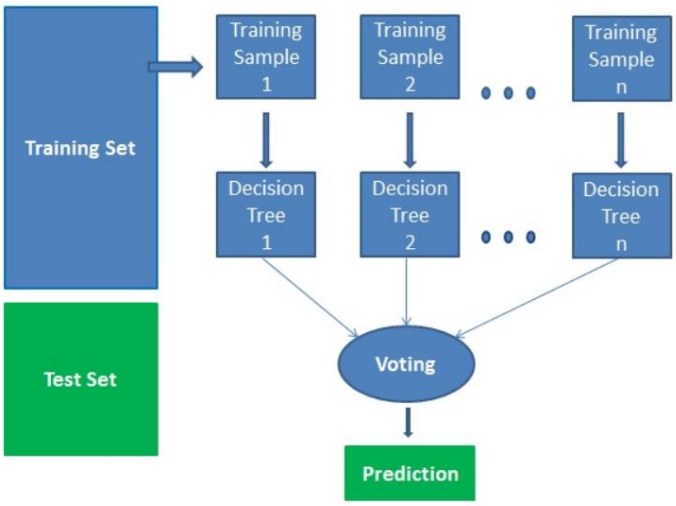
It's technically an ensemble method of decision trees created on a randomly split dataset (based on the divide-and-conquer methodology). The forest is the name given to a group of decision tree classifiers. Individual decision trees are created for each attribute using an attribute selection indicator such as information gain, gain ratio, or Gini index.

Each tree is based on a separate random sample. Each tree votes on a classification challenge, and the most popular class is picked as the output. In the case of regression, the ultimate result is the average of all the tree outputs. In comparison to other non-linear classification algorithms, it is both simpler and more powerful.

2.1) Steps involved in random forest

Random forest basically works in 4 steps:

- Select random samples from a given dataset.
- Construct a decision tree for each sample and get a prediction result from each decision tree.
- Perform a vote for each predicted result.
- Select the prediction result with the most votes as the final prediction.



A visualization of how random forest works internally

2.2) Advantages:

- Random forests are considered as a highly accurate and robust method because of the number of decision trees participating in the process.
- The algorithm is applicable to classification and regression.
- The relative feature importance may be obtained, which aids in the selection of the most contributing features for the classifier.

Source: https://www.datacamp.com/community/tutorials/random-forests-classifier-python

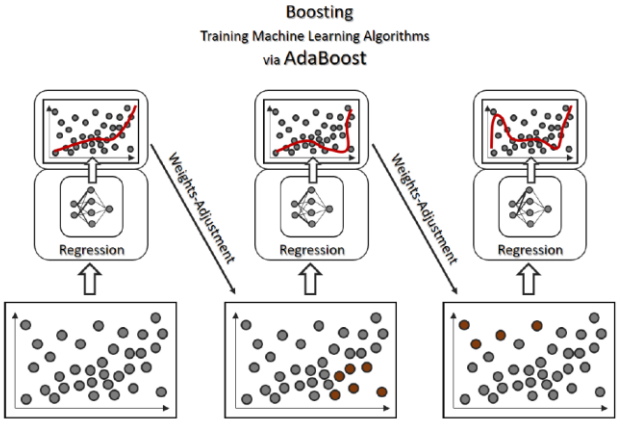
2.3) Disadvantages:

- In comparison to a decision tree, where you can readily decide by following the path in the tree, the model is difficult to interpret.
- Because it has several decision trees, random forests take a long time to provide predictions. Every time it makes a prediction, all the trees in the forest must make a prediction for the same supplied input and then vote on it. This entire procedure is time consuming.

3) AdaBoost

AdaBoost also called Adaptive Boosting is an ensemble learning method (sometimes known as "meta-learning") that was developed to improve the performance of binary classifiers. AdaBoost utilizes an iterative technique to learns from the errors of weak classifiers and transforming them into strong ones.

This algorithm constructs a model and assigns equal weights to all data points. It then it applies a higher weight to the incorrectly categorized points. In the following model, all points with higher weights are given more importance. This process continues to train models until a smaller error is received.



3.1) Objective Function and inner workings:

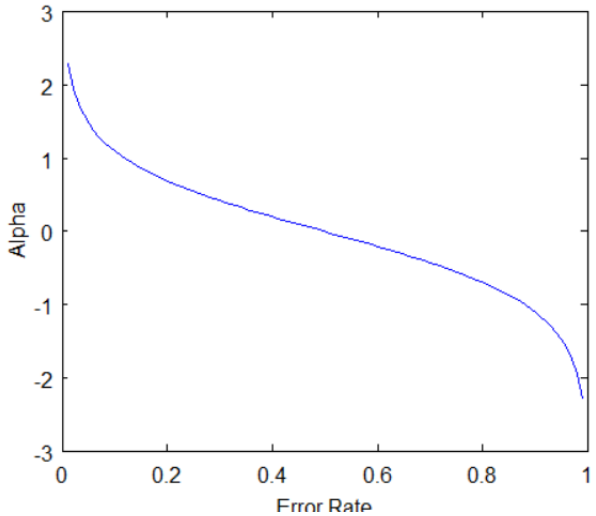
The formula boosting uses to calculate the weights is

$$w(x_i, y_i) = \frac{1}{N}, \; i = 1, 2, \ldots . n$$

Where N is the total number of datapoints
Once the weights are calculated, in the next step the 'Influence' for this classifier in classifying the datapoints is calculated using this formula:

$$\frac{1}{2}\log\frac{1 - Total\ Error}{Total\ Error}$$

Where the total error is the sum of all the sample weights of misclassified data points, the total error is always in the range of 0 and 1, where 0 Indicates perfect stump and 1 indicates bad stump.



From the graph above we can see that when there is no misclassification then we have no error. Similarly, when the classifier predicts half right and half wrong then the Total Error is 50% and the importance of the classifier is 0. If all the samples have been incorrectly classified, the error is very high, and the alpha value will be a negative integer.

After finding the importance of the classifier and total error we need to finally update the weights and for this, we use the following formula:

$$New\ sample\ weight\ =\ old\ weight\ *\ e^{\pm Amount\ of\ say\ (\alpha)}$$

Alpha is -ve when the sample is correctly classified and +ve when they are classified correctly.

To sum up, this algorithm follows the following steps and keeps iterating until the error is minimized

1) Assign equal weights to all the datapoints
2) Find the stump that does the best job classifying the new collection of samples by finding their Gini Index and selecting the one with the lowest Gini index
3) Calculate the “Amount of Say” and “Total error” to update the previous sample weights.
4) Normalize the new sample weights.

3.2) Advantages:

- It is an Improvement over bagged trees by decorrelating the trees.
- It randomly selects the ‘n’ predictors at every tree split.
- AdaBoost can be used to improve the accuracy of your weak classifiers hence making it flexible.

Reference: https://www.analyticsvidhya.com/blog/2021/09/adaboost-algorithm-a-complete-guide-for-beginners/

3.3) Disadvantages:

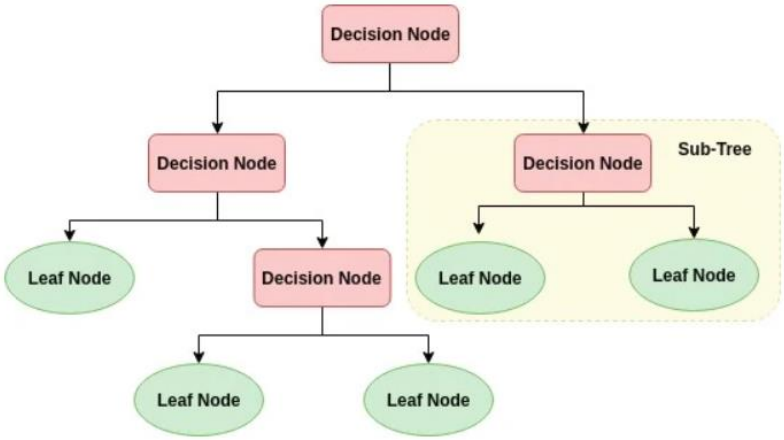
- AdaBoost is slower and takes a long time to run compared to random forest or other tree based methods.
- AdaBoost can be extremely sensitive to noise and outliers.

4) Decision Tree

A decision tree is a tree structure that looks like a flowchart, with an internal node representing a feature (or attribute), a branch representing a decision rule, and each leaf node representing the outcome. The root node is the node at the top of a decision tree.

It learns to partition based on attribute value. It splits the tree recursively, which is known as recursive partitioning. This flowchart-like structure aids you in making decisions. It's a flowchart diagram-style depiction that easily replicates human level thinking. As a result, decision trees are simple to grasp and interpret.

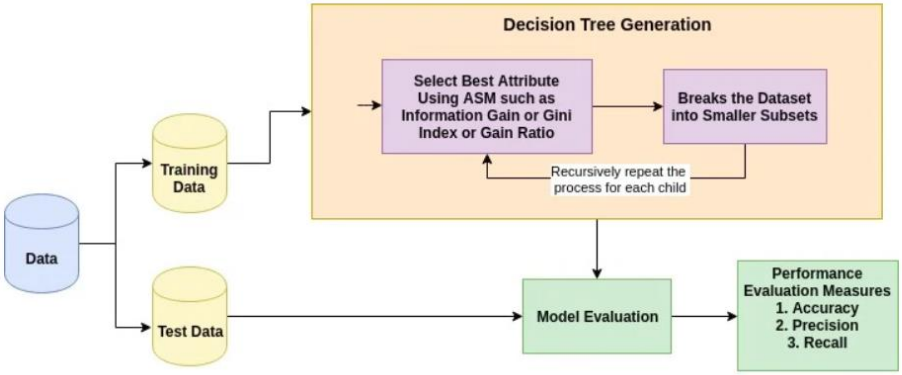
The decision tree is a distribution-free or non-parametric method, which does not depend upon probability distribution assumptions. Decision trees can handle high dimensional data with good accuracy.



4.1) Objective Function and inner workings:

The basic idea behind any decision tree algorithm is:

- Select the best attribute using Attribute Selection Measures to split the records.
- Make that attribute a decision node, then break the dataset into smaller subsets.
- Starts tree building by repeating this process recursively for each child until one of the conditions matches:
    - All the tuples belong to the same attribute value.
    - There are no more remaining attributes.
    - There are no more instances.



4.1.1) Information Gain

It works on the concept of entropy which determines the impurity level of the input data set.

Entropy is a term used in science and mathematics to describe the unpredictability or impurity in a system. It refers to the impurity in a group of examples.

Entropy is reduced because of information gain. Based on specified attribute values, information gain computes the difference between entropy before split and average entropy after split of the dataset. The information gain decision tree algorithm is used by the Iterative Dichotomiser decision tree technique.

$$\text{Info}(D) = -\sum_{i=1}^{m} pi \log_2 pi$$

Where, Pi is the probability that an arbitrary tuple in D belongs to class C_i. Further.

$$\text{Info}_A(D) = \sum_{j=1}^{V} \frac{|Dj|}{|D|} \text{ X Info}(D_j)$$

$$\text{Gain}(A) = \text{Info}(D) - \text{Info}_A(D)$$

Where,

- Info(D) is the average amount of information needed to identify the class label of a tuple in D.
- |Dj| / |D| acts as the weight of the j^{th} partition.
- $\text{Info}_A(D)$ is the expected information required to classify a tuple from D based on the partitioning by A.

The attribute A with the highest information gain, Gain(A), is chosen as the splitting attribute at node N().

4.1.2) Gini index

Gini method is another metric that decision tree algorithm used to create data split levels.

$$\text{Gini}(D) = 1 - \sum_{i=1}^{m} \text{Pi}^2$$

For each attribute, the Gini Index takes into account a binary split. You can compute a weighted sum of each partition's impurity. If data D is partitioned into D1 and D2 by a binary split on attribute A, the Gini index of D is:

$$\text{Gini}_A(D) = \frac{|D1|}{|D|} \text{Gini}(D_1) + \frac{|D2|}{|D|} \text{Gini}(D_2)$$

In the case of a discrete-valued attribute, the subset with the lowest gini index for that attribute is picked as a splitting attribute. In the case of continuous-valued attributes, the technique is to choose one of each pair of nearby values as a viable split-point, with the point with the lowest gini index chosen as the splitting point.

$$\Delta Gini(A) = Gini(D) - Gini_A(D).$$

The attribute with minimum Gini index is chosen as the splitting attribute.

Source: https://www.datacamp.com/community/tutorials/decision-tree-classification-python

4.3) Advantages:

- Decision trees need less effort for data preparation during pre-processing than other methods.
- A decision tree does not need data scaling.
- A decision tree approach is very simple to communicate to technical teams and stakeholders alike.

4.4) Disadvantages:

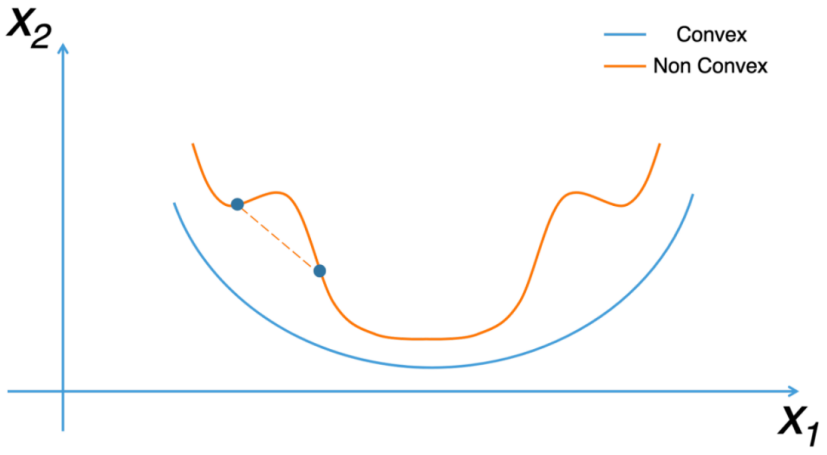
- A minor change in the data can result in a significant change in the structure of the decision tree, resulting in instability.
- The training period for a decision tree is often longer.
- It is highly variable on each run, hence it is recommended to set a seed value for reproducibility.

5) SGD Regressor

Gradient descent is a fundamental algorithm in machine learning and deep learning. It is a highly effective optimization technique capable of training linear regression, logistic regression, and neural network models.

Gradient descent is an optimization algorithm. It is used to quickly find the minimum value of a function. The definition of gradient descent is straightforward. It is a method for calculating the minimum of a convex function. It does this by changing the parameters of the function in question iteratively. It is a method that is utilized in applications such as linear regression.

A convex function is a function that looks like a valley with a global minimum in the center. Conversely, a non-convex function is a function that has several local minima, and the gradient descent algorithm should not be used on these functions at the risk of getting stuck at the first minima encountered.



The SGD provides individual parameter updates for each training example. It allows attention to be paid to each example, ensuring that the process is error-free. Depending on the problem, this can help the SGD become faster compared to batch gradient descent. Its regular updates provide us with detailed improvement rates.

However, these updates are computationally expensive, especially when compared to the stepwise descent strategy. Furthermore, the frequency of updates can result in noisy gradients and prevent the error rate from reducing. Instead, the error rate skyrockets, posing a long-term concern.

Basically, how this method works is: Suppose you are in a valley with beautiful mountains:

1) From your current position, you look all around you for the direction of where the slope is going down the hardest.
2) Once you find that direction, you follow it for a distance and repeat step 1.

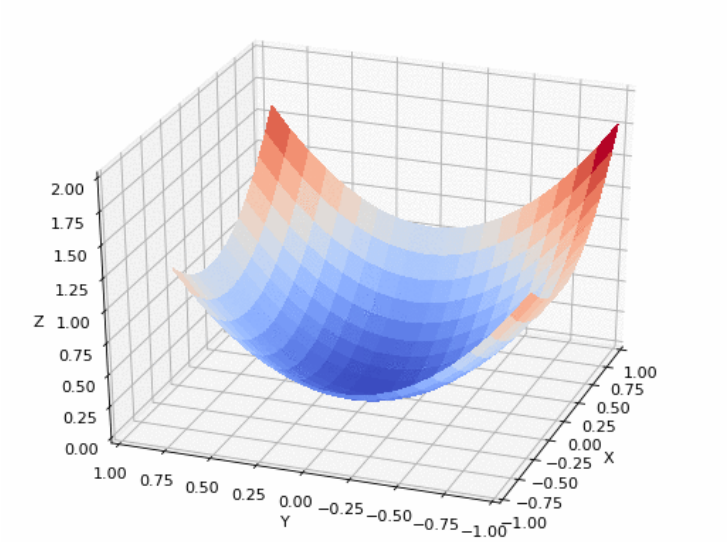
By repeating steps 1 and 2 in a loop, you are sure to converge on the minimum of the valley. This strategy is nothing more or less than the gradient descent algorithm. To put the above 2 steps in simple words:

Step 1: Determine the cost function's derivative:

- We begin with a random starting location (as if we were lost in the vally) and then measure the slope value at that point. In mathematics, a slope is calculated by taking the function's derivative.

Step 2: Revise the model parameters.

- Then we advance by a given distance d in the slope direction, which descends, but not by the same distance this time. This distance is referred to as the "learning rate."



The result of this operation is to modify the value of the parameters of our model (our coordinates in the valley change when we move).

5.1) Objective Function:

Both statistical estimation and machine learning consider the problem of minimizing an objective function that has the form of a sum:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w),$$

where the parameter w that minimizes Q(w) is to be estimated. Each summand function Q_i is typically associated with the i^{th} observation in the data set (used for training).

5.2) Advantages:

- Larger datasets may converge faster because the parameters are updated more frequently.
- Because of the frequent updates, the steps made towards the minima of the loss function include oscillations that can aid to get out of the loss function's local minimums.
- Because only one sample is processed at a time, it is computationally fast.

5.2) Disadvantages:

- It lacks the benefit of vectorized operations because it only works with a single sample at a time.
- The steps taken towards the minima are highly loud as a result of the frequent updates. This frequently causes the gradient descent to veer in unexpected directions.

6) Data Overview

Bank Marketing Data Set: The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

The data has 21 columns and 20,000 rows in total. This includes 20 independent variables and 1 dependent variable 'subscribed' with the following distribution of the target variable.

Subscribed	Count	Percentage Count
Yes	2,271	11.4%
No	17,729	88.6%

7) Data Pre-Processing

1) The data has a total of 1,730 NA's. None of them were dropped in pre-processing.
2) NA's were imputed based on the most occurring values and a Missing flag was added.
3) The columns Age, campaign, pdays and previous had outliers. They were fixed using scaling.
4) The categorical variables were encoded using Ordinal Encoder.
5) Data was split in train and test before any feature engineering to avoid any data leakage with the test size of 20%.

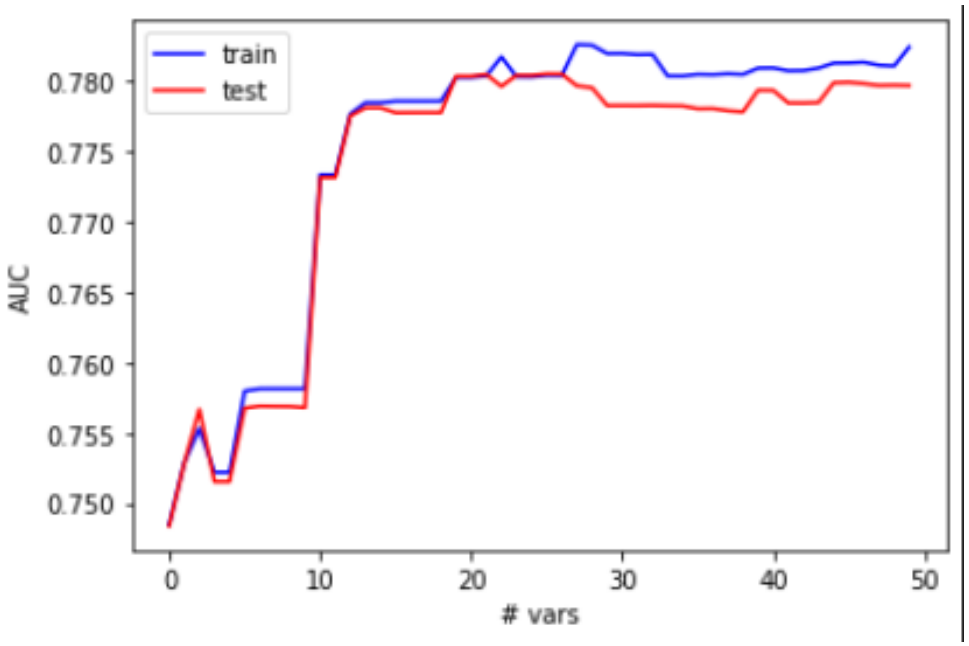
8) Feature Engineering

1) Polynomial terms of degree 2 and 3 were added to numerical variables.
2) Categorical variables were mapped using decision tree based re-mapping and were grouped based on the last node as the category.
3) Numerical variables were categorized and grouped in buckets and then binned in categories.
4) Weight of Evidence was applied to the categorical variables and were dummy encoded based on the incident rates per category.
5) Any missing value created by the above steps was imputed with the mean.
6) Total number of features engendered using the above techniques was 770.
7) All new featured with low or no variance and duplicate values were dropped.

9) Feature Selection

Fisher Score was used to do variable selection. Different thresholds were tried; however, the best results were with the top 15 variables based on the fisher score.

As we can see from the following graph, the AUC starts to stabilize after 15 variables.



Further, top 35 variables were kept for further experimentation as there is slight variation in the AUC when top 30-35 variables were used based on the fisher score.

Further Kendall, Pearson and Spearman's correlation scores were used to select more features to experiment with later. The intersection of the features based on all 3 correlations were selected based on the cutoff value of 0.8.

10) Modeling Experimentation

10.1) Initial Benchmark

The five models selected for this project were first ran with all the engineered features with default parameters (All featured before variable selection).

	Train_Accuracy	Train_AUC	Test_Accuracy	Test_AUC
DT	1.000000	1.000000	0.83600	0.634771
Random Forest	0.999938	1.000000	0.89275	0.772321
Logistic Regression	0.904250	0.818633	0.89475	0.780463
ADABoost	0.901500	0.811320	0.89675	0.780094
SGD Classifier	0.886437	0.500000	0.88650	0.500000

We can see from the above table that:

1) Decision Tree is highly overfitted.
2) Random Forest is also overfitting.
3) Logistic Regression also has a minor overfit.
4) AdaBoost has the similar performance as Logistic Regression
5) SGD Classifier does not overfit, however the performance is very bad at 50% AUC and 88% Accuracy.

10.2) Performance after Feature Selection

The same five models were re-initialized and re-fitted with the top 35 features selected using Fisher Score with the default parameters.

	Train_Accuracy_F	Train_AUC_F	Test_Accuracy_F	Test_AUC_F
DT	0.923813	0.858448	0.89250	0.701971
Random Forest	0.923750	0.850021	0.89350	0.764386
ADABoost	0.900813	0.798792	0.89725	0.787708
Logistic Regression	0.900250	0.781084	0.89725	0.780051
SGD Classifier	0.899625	0.758697	0.89725	0.736467

From above table we can see that:

1) Decision Tree is still overfitting, but overfitting is less compared to no feature selection.
2) Random Forest also improves on overfitting, but overfitting is still present.
3) AdaBoost is performing similarly if we look at the AUC, however, the overfitting has gone down if we look at the accuracy score.
4) Logistic Regression is performing really well.
5) SGD Classifier also shows signs of overfitting based on AUC, however the accuracy seems to be good.

10.2) Cross Validation

Further a 10 folds cross validation was performed on the entire dataset for the top 2 models to verify the AUC.

1) Logistic Regression with variable selection and 10 folds cross validation gave the mean AUC of 77.9%. This seems to be very close to the train and test performance.
2) AdaBoost with variable selection and 10 folds cross validation gave the mean AUC of 79%. This also aligns with the train and test performance.

Next, the models were re-initialized and re-fitted with the features selected using Fisher's score, and the correlation score based on Kendall, Pearson and Spearman's correlation.

	Train_Accuracy_FP	Train_AUC_FP	Test_Accuracy_FP	Test_AUC_FP
DT	0.940438	0.931265	0.88250	0.630850
Random Forest	0.940438	0.918221	0.88250	0.734139
ADABoost	0.901000	0.798024	0.89800	0.787114
Logistic Regression	0.900250	0.784316	0.89825	0.781717
SGD Classifier	0.886250	0.771722	0.88675	0.761106

From above table we can see that:

1) Decision Tree performance went down when more variables were added.
2) Random Forest performance went down when more variables were added.
3) AdaBoost seems to be performing ok as well
4) Logistic Regression also seems to be performing good as well.
5) SGD Classifier also shows signs of overfitting

10.2) Model Evaluation & Comparison

Evaluating AUC:

	DT	Random Forest	ADABoost	Logistic Regression	SGD Classifier
Train_AUC_All_Features	1.000000	1.000000	0.811320	0.818633	0.500000
Test_AUC_All_Features	0.634771	0.772321	0.780094	0.780463	0.500000
Train_AUC_Fisher_Score	0.858448	0.850021	0.798792	0.781084	0.758697
Test_AUC_Fisher_Score	0.701971	0.764386	0.787708	0.780051	0.736467
Train_AUC_Fisher_Corr_Score	0.931265	0.918221	0.798024	0.784316	0.771722
Test_AUC_Fisher_Corr_Score	0.630850	0.734139	0.787114	0.781717	0.761106

When comparing the AUC, we see that a lot of feature engineering is not always helpful, as model's performance with all engineered features is worse and tend to overfit a lot.

For this dataset **Logistic Regression** seems to be performing best after feature selection followed by **AdaBoost.**

References:

Class content shared by **Minh Phan -** IÉSEG School of Management

ML & Data Pipeline Notebook shared by **Minh Phan -** IÉSEG School of Management

https://hastie.su.domains/ISLR2/ISLRv2_website.pdf

https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python

https://www.datacamp.com/community/tutorials/random-forests-classifier-python

https://www.datacamp.com/community/tutorials/decision-tree-classification-python

https://www.datacamp.com/community/tutorials/adaboost-classifier-python

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

https://towardsdatascience.com/how-to-make-sgd-classifier-perform-as-well-as-logistic-regression-using-parfit-cc10bca2d3c4

https://stackoverflow.com/questions/48220125/how-to-generate-roc-curve-with-cross-validation-using-sgd-classifier-loss-hinge

https://www.datacamp.com/community/tutorials/tutorial-gradient-descent

https://stackabuse.com/calculating-pearson-correlation-coefficient-in-python-with-numpy/

https://programtalk.com/python-examples/skfeature.function.similarity_based.fisher_score.fisher_score/

https://stats.stackexchange.com/questions/277123/fisher-score-feature-selection-implementation

https://stackoverflow.com/questions/52741236/how-to-calculate-p-values-for-pairwise-correlation-of-columns-in-pandas

https://stackoverflow.com/questions/67460329/sgd-classifier-precision-recall-curve

https://stackoverflow.com/questions/33547965/computing-auc-and-roc-curve-from-multi-class-data-in-scikit-learn-sklearn

https://stackoverflow.com/questions/55250963/how-to-get-probabilities-for-sgdclassifier-linearsvm

https://stackoverflow.com/questions/50830059/python-pandas-extracting-numbers-within-text-to-a-new-column

https://stackoverflow.com/questions/33604139/how-to-split-pandas-column-by-a-delimiter-and-select-preferred-element-as-the-re