Machine Learning Algorithms

The Algorithms, which were implemented in the Kaggle Project:

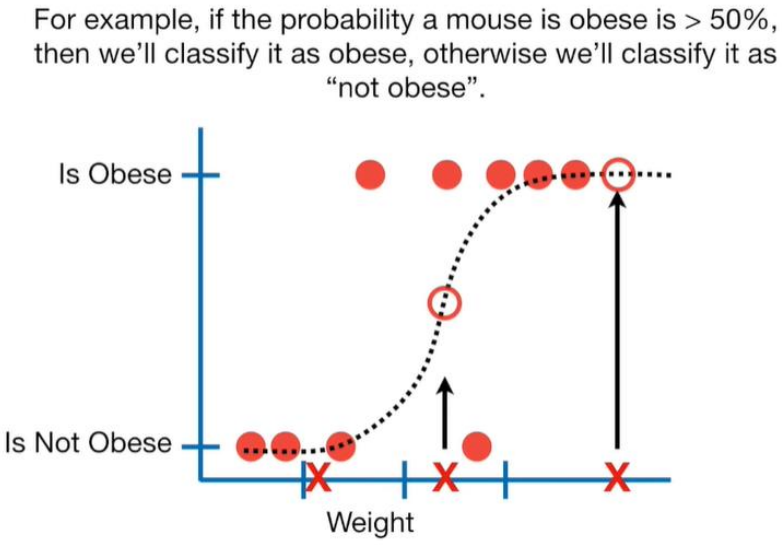
1) Logistic Regression
2) Decision Tree
3) Random Forests
4) Gradient Boosting
5) K-nearest neighbors

Logistic Regression

Logistic Regression is a Generalized Linear Model that is it is a generalization of the concepts and abilities of regular linear models.

Data is fit into linear regression model, which then be acted upon by a logistic function predicting the target categorical dependent variable.

It is often used for classification. For Example, in the below Figure we could tell whether a mouse is obese or not obese given its weight, by projecting the data point to the curve and setting up a threshold.



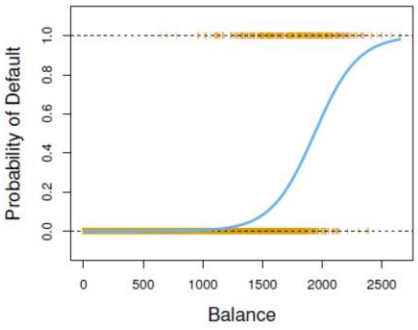
Logistic Regression fits an S shaped logistic function. The Y-axis is the probability.

Logistic regression's ability to provide probabilities and classify new samples using continuous and discrete measurements makes it a popular machine-learning algorithm.

If we remember linear regression, we take the sum of the squared residual from the data points to the fitted line. Unlike here, we have maximum likelihood.

Logistic Regression

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$$



Model

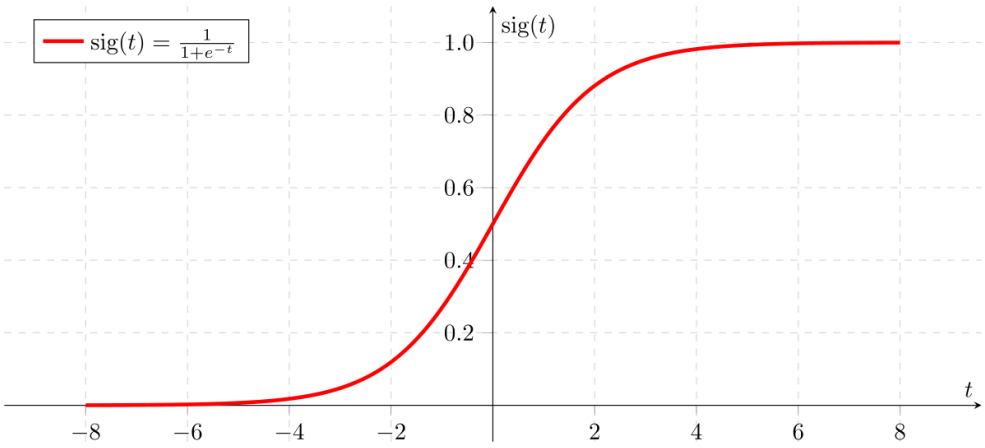
Output = 0 or 1

Hypothesis => Z = WX + B

hΘ(x) = sigmoid (Z)

This cost function is called sigmoid function or logistic function.

Sigmoid Function



If ‘Z’ goes to infinity, Y(predicted) will become 1 and if ‘Z’ goes to negative infinity, Y(predicted) will become 0.
The output from the hypothesis is the estimated probability. This is used to infer how confident can predicted value be actual value when given an input.

Log odds transformation of p(X):

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X$$

You use the log of odds to convert the axis, and get the coefficients and intercept of the curve.

Types of logistic regression:

1) Binary – 2 possible outcomes
2) Multinomial – More than 2 categories
3) Ordinal – More than 2 categories with ordering

Like mentioned before, to classify the class of data to which it belongs to, we set a threshold, which is called the decision boundary. Decision boundary can be linear or non-linear. Polynomial order can be increased to get complex decision boundary.

Cost Function:

For logistic regression, the Cost function is defined as:

$-\log(h\vartheta(x))$ if y = 1

$-\log(1-h\vartheta(x))$ if y = 0

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

We use gradient descent to minimize the cost function.

Advantages:

1) It's simplicity makes it easy to apply as a base model and compare other complex models.
2) Performs well when dataset is linearly separable.

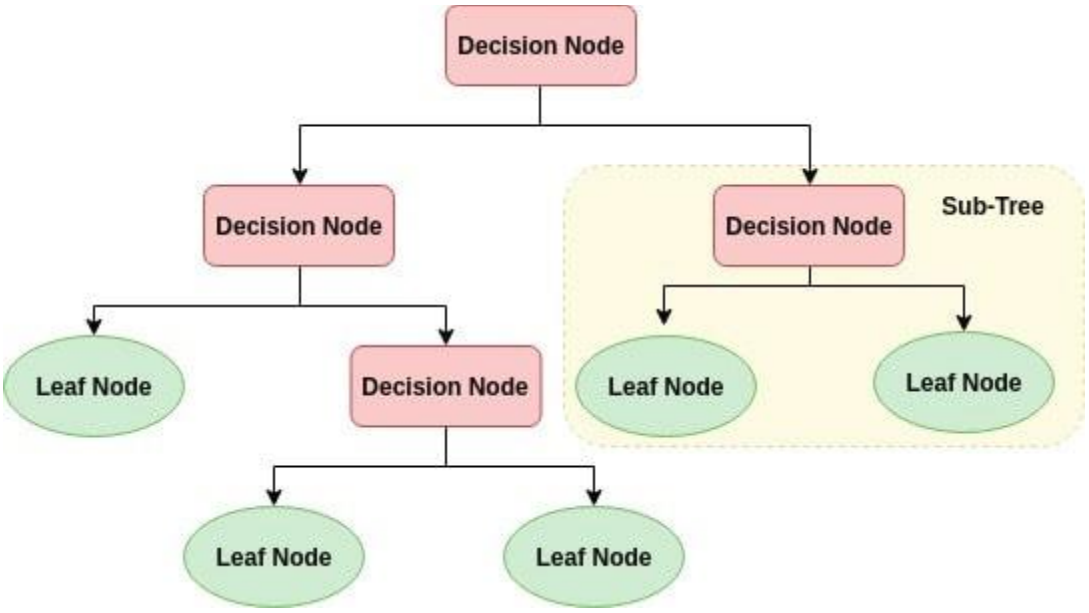
Disadvantages:

1) Overfits the data easily if the number of observations and features are less.

Decision Tree

Decision tree learning is a supervised machine learning technique for inducing a decision tree from data. It is a predictive model, which is a mapping from observations about an item to

conclusions about its target value. In the tree structures, leaves represent classifications, nodes are features, and branches represent conjunctions of features that lead to the classifications.



The top of the tree is the Root node. The other nodes are called internal nodes. Finally we have the leaf nodes at the bottom.

A Root node is the starting point followed by a split, which partitions the data, and further into child nodes and into leaves. The leaf has the final decision or result.

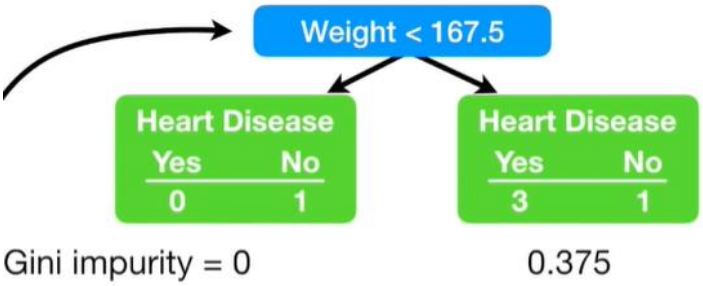
To determine which feature is best at each split, we measure and compare impurity of different splits. They are many methods to measure this such as Information gain, Gini Impurity, variance reduction etc. One of the methods which has been used in the kaggle competition as well is **Gini**.

Represents the probability that a randomly selected sample from a node will be incorrectly classified according to the distribution of samples in the node.

The feature with the lowest Gini impurity is considered for the split.

If the node itself has the lowest impurity, that there is no point in separating the data, then it becomes the leaf node.

For numeric data, the average of data values is taken and the Gini impurity is measured. This process is repeated for a high number of data points. The one with the least impurity is taken as the cutoff.



Eventually, the weighted total Gini Impurity of the last layer goes to 0 meaning each node is completely pure and there is no chance that a point randomly selected from that node would be misclassified. While this may seem like a positive, it means that the model may potentially be overfitting because the nodes are constructed only using training data.

Splitting the space into 2 regions.

$$R_1(j,s) = \{X|X_j < s\} \text{ and } R_2(j,s) = \{X|X_j \geq s\}$$

The cut with least RSS.

$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

To avoid this, we can use a technique called pruning. Pre-pruning where we can set the splitting criteria to stop the tree to grow. Post-pruning where you grow a very large tree and prune it back to a sub tree.

Advantages:

1) Decision tree is very easy to interpret and explain.
2) It can accept both numerical and character variables.
3) It is easy to implement and quick to train.
4) It is highly flexible, and can even be trained to predict the predictions of neural network algorithm, thus opening the black box of the neural network.

Disadvantages:

1) Over fits on training data if the parameters are not tuned.
2) Has unlimited flexibility (1 leaf node for 1 observation).
3) The bias variance tradeoff is dependent solely on maximum depth of the tree.
4) Has low bias and high variance on train data.

Random Forests

The fundamental idea behind a random forest is to combine many decision trees into a single model. Individually, predictions made by decision trees may not be accurate, but combined, the predictions will be closer to the mark on average. By pooling predictions, we can incorporate much more knowledge than from one individual. That is pretty much the wisdom of crowds.

Each decision tree in the forest considers a random subset of features and only has access to a random set of the training data points.

This technique of training each tree on a random subset of the data points with replacement is called bagging, short for bootstrap aggregating.

Random sampling of data point with random sampling of subset of features together make it a random forest.

For regression, the random forest takes an average of all the individual decision tree estimates. For classification, it takes the majority vote for the predicted class.

Like discussed earlier in decision tree section, over fitting occurs when the model trains on the data and memorizes it by fitting closely, which means the model has learnt the data as well as the noise.

The model will be not able to generalize new data.

Gradient Boosting

Most of the winning models in the machine learning competitions are ensembles of different algorithms. Gradient boosting is one such particular model. In the In class kaggle competition, 3 gradient boosting based algorithms were used – XGBoost, AdaBoost, CatBoost. Adaptive boosting or AdaBoost is the first Boosting Algorithm. This algorithm shares its DNA with decision trees.

Boosting is a method of converting weak learners into strong learners. In boosting, each new tree is a fit on a modified version of the original data set.

Gradient boosting involves three elements:

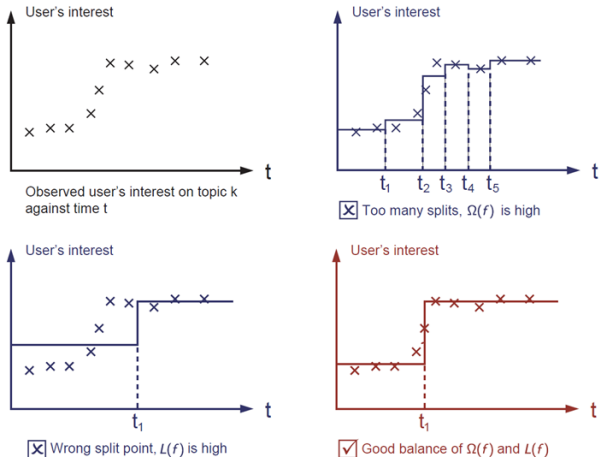
A loss function to be optimized.
A weak learner to make predictions.
An additive model to add weak learners to minimize the loss function.

A weak learner is defined as one whose performance is at least slightly better than random chance. Decision trees are used as the weak learner in gradient boosting.

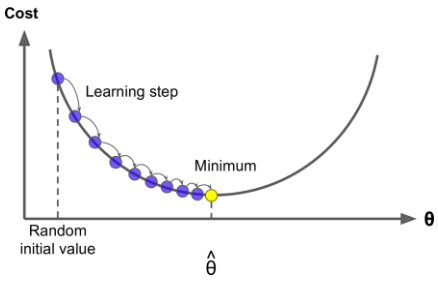
Unlike the bagging ensemble Random Forest, Trees are not made independently but sequentially. Trees are added one at a time, and existing trees in the model are not changed. A gradient descent procedure is used to minimize the loss when adding trees.

For suppose our loss function is mean squared error (MSE). We want our predictions, such that our loss function (MSE) is minimum. By using **gradient descent** and updating our predictions based on a learning rate, we can find the values where MSE is minimum.

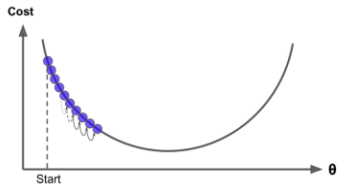
So, we are basically updating the predictions such that the sum of our residuals is close to 0 (or minimum) and predicted values are sufficiently close to actual values.



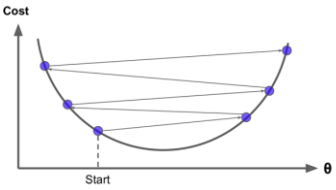
Gradient descent measures the local gradient of the loss function for a given set of parameters and takes steps in the direction of the descending gradient. Once the gradient is zero, we have reached the minimum.



An important parameter in gradient descent is the size of the steps which is determined by the learning rate. If the learning rate is too small, then the algorithm will take many iterations to find the minimum. On the other hand, if the learning rate is too high, you might jump cross the minimum and end up further away than when you started. Smaller values reduce the chance of overfitting but also increases the time to find the optimal fit.



a) too small



a) too big

Advantages:

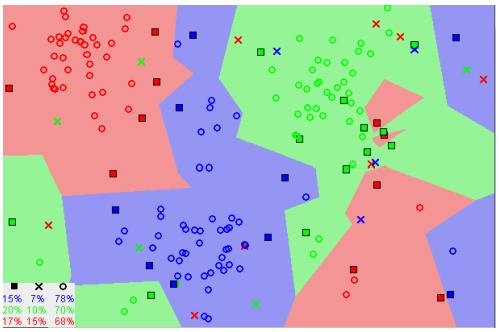
1) High predictive Accuracy.
2) Lots of flexibility - can optimize on different loss functions and provides several hyperparameter tuning options that make the function fit well.
3) Not much data preprocessing is required, works great with categorical and numerical values.
4) Handles missing data well. No imputation required.

Disdvantages:

- GBMs will continue improving to minimize all errors which can cause overfitting.
- Might require high number of trees there by increasing the computational cost.
- So many parameters to generalize, there by increasing the time for grid search.
- Interpretation is a bit complex compared to decision tree.

K-nearest neighbors

The KNN algorithm assumes that similar things are near to each other.



K is the number of neighbors you choose to fit to a datapoint. We run the KNN algorithm several times with different values of K and choose the K that reduces the number of errors we encounter while maintaining the algorithm's ability to accurately make predictions when its given test data.

As we decrease the value of K to 1, our predictions become less stable. As we increase the value of K, our predictions become more stable due to majority voting. In cases where we are taking a majority vote among labels, we usually make K an odd number to have a tiebreaker.

Randomly assigns the K to each of the observations for initial formation of the cluster and find where the variation within the cluster is the least.

$$\underset{C_1,\ldots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} \text{WCV}(C_k) \right\}.$$

For each of the K-clusters you compute a centroid. Later you assign each observation to the cluster whose centroid is closest by calculating the Euclidean distance.

$$\text{WCV}(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2,$$

Advantages

1. The algorithm is simple and easy to implement.
2. There is no need to build a model, tune several parameters, or make additional assumptions.
3. The algorithm can be used for classification, regression, and search as well.

Disadvantages

1. The algorithm gets slower as the number of predictors or independent variables increase.

Kaggle Competition Setup:

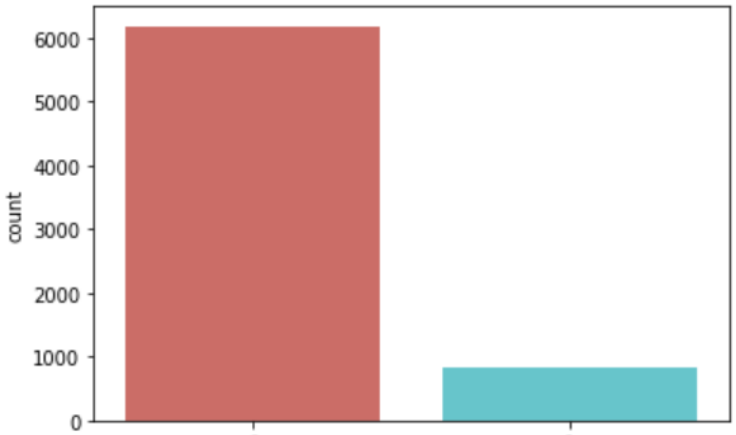
1) Preprocessing the data.
2) Feature selection.
3) Fitting the data.
4) Cross validation.

5) Hyper parameter tuning.

Basic analysis of the data:

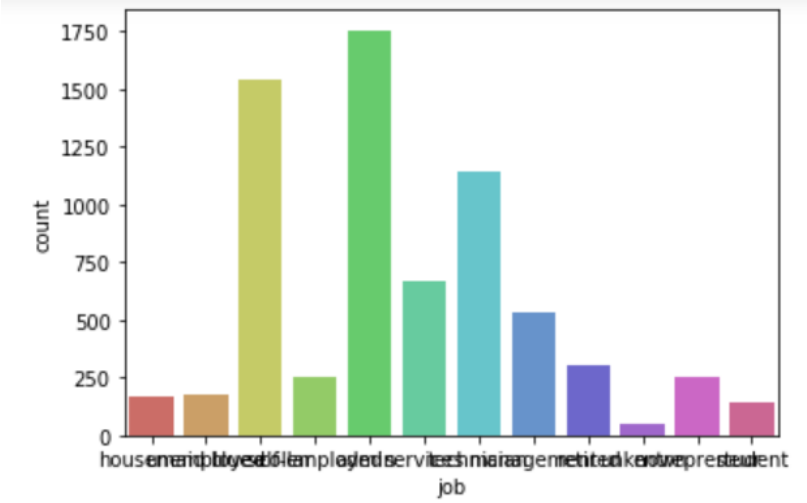
```
client_id         7000 non-null int64
age               7000 non-null int64
job               7000 non-null object
marital           7000 non-null object
education         7000 non-null object
default           7000 non-null object
housing           7000 non-null object
loan              7000 non-null object
contact           7000 non-null object
month             7000 non-null object
day_of_week       7000 non-null object
campaign          7000 non-null int64
pdays             7000 non-null int64
previous          7000 non-null int64
poutcome          7000 non-null object
emp.var.rate      7000 non-null float64
cons.price.idx    7000 non-null float64
cons.conf.idx     7000 non-null float64
euribor3m         7000 non-null float64
nr.employed       7000 non-null float64
subscribe         7000 non-null int64
```

They were no missing values present in the train data set.



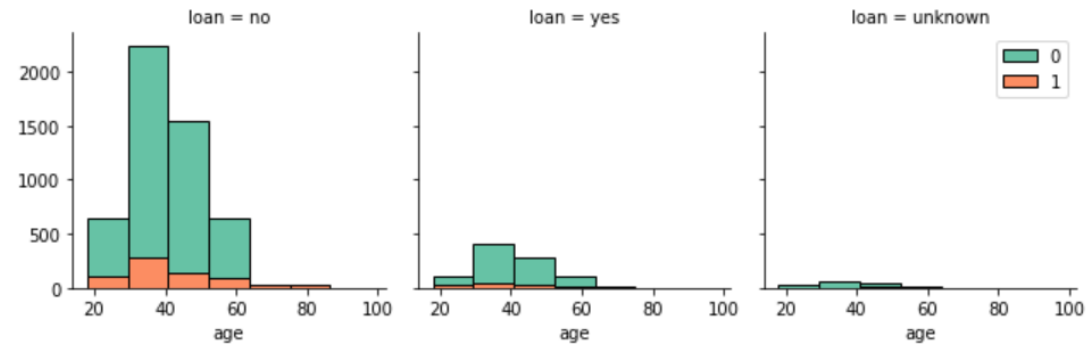

```
0    6178
1     822
```

The distribution of our target “subscribe”.

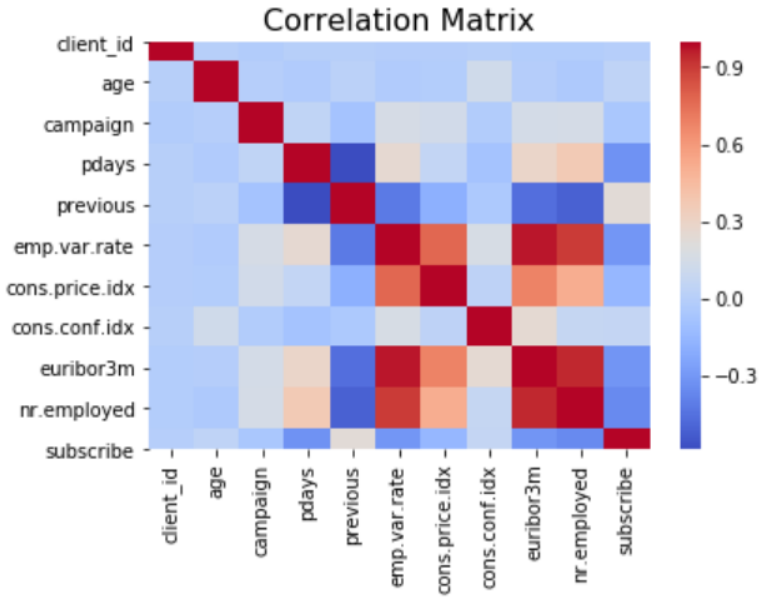



```
admin.           1756
blue-collar      1541
technician       1143
services          668
management        534
retired           306
self-employed     256
entrepreneur      253
unemployed        176
housemaid         172
student           146
unknown            49
```

The distribution of classes in the variable job.



The distribution of users given the variable loan and the target.



The correlation between the variables amongst them and with the target.

The train and validation set is split in the ratio 7:3.

```
train
0    4312
1     588
Name: subscribe, dtype: int64 (4900, 62)
validation
0    1866
1     234
Name: subscribe, dtype: int64 (2100, 62)
```

After trying to develop different features by feature engineering, the results ended up being pretty much the same, so addition of variables has been removed from the code and the original variables are used.

Feature Selection

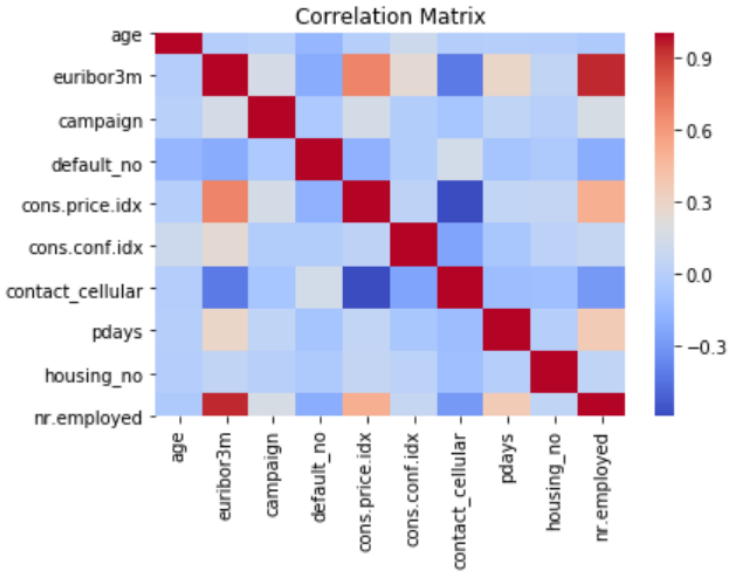
It makes more sense to select the features before dummying the categorical variables, because if you do it after dummying, you are basically selecting the classes of the features but not the features. After trying both the techniques, a decision was made by comparing the results.

Features were selected after dummying the variables.

LGBM is used for the feature selection based on the feature importance after fitting the data. The reason to use this algorithm is because of its ability be both the fastest and most flexible amongst the gradient boosting family. It comes as all-inclusive package for covering the wide range of complexity settings that can occur in real world datasets, especially heterogenous ones.

Also for the fact that most of the kaggle competitions have been aced with this algorithm because in general the datatsets don't involve features that would be easily mapped to the target.

Selected features and their correlation after running the LGBM algorithm.



Fitting

Fitting the base models with default hyper parameters on the training and validation set. The results are as follows :

Training Set:

	tree	logistic	randomForest	boostedTree	svm	neuralNet	neighbors	adaboost	linearDiscriminant	Quadratic	xboost	catboost
Accuracy	0.986531	0.893061	0.972653	0.912245	0.890000	0.523061	0.908571	0.894286	0.883878	0.866531	0.945306	0.928367
AUC	0.999115	0.767965	0.996857	0.828493	0.677612	0.651245	0.919969	0.804969	0.760019	0.781758	0.964400	0.892010

Validation Set:

	tree	logistic	randomForest	boostedTree	svm	neuralNet	neighbors	adaboost	linearDiscriminant	Quadratic	xboost	catboost
Accuracy	0.835238	0.902857	0.883810	0.900476	0.901905	0.531429	0.889048	0.897143	0.899048	0.887619	0.898095	0.897619
AUC	0.626514	0.783281	0.753679	0.805761	0.698583	0.642681	0.722487	0.798588	0.780267	0.787300	0.777942	0.808537

After running on the base models without hyper parameter tunning, the CatBoost model has the best AUC.

Report after performing cross validation on all the base models :

```
Model Report:  DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=
None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
Accuracy : 0.9865
AUC (Train): 0.999115
CV Score : Mean - 0.6183054 | Std - 0.01557817 | Min - 0.5908194 | Max - 0.6342463

----------------------------------**********----------------------------------
-----


Model Report:  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept
=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
Accuracy : 0.8931
AUC (Train): 0.767965
CV Score : Mean - 0.7506971 | Std - 0.03469487 | Min - 0.7095616 | Max - 0.7954206

----------------------------------**********----------------------------------
-----


Model Report:  RandomForestClassifier(bootstrap=True, class_weight=None, criterion='g
ini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
Accuracy : 0.9716
AUC (Train): 0.995837
CV Score : Mean - 0.7320606 | Std - 0.0223537 | Min - 0.7038293 | Max - 0.771319

----------------------------------**********----------------------------------
-----


Model Report:  GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
Accuracy : 0.9122
AUC (Train): 0.828493
CV Score : Mean - 0.784422 | Std - 0.02824203 | Min - 0.7520229 | Max - 0.821754
```

```
-----------------------------------**********-------------------------------
-----


Model Report:  SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=True, random_state=None, shrinking=True, tol=0.001,
    verbose=False)
Accuracy : 0.89
AUC (Train): 0.677612
CV Score : Mean - 0.6328137 | Std - 0.05900587 | Min - 0.5360415 | Max - 0.7055199

-----------------------------------**********-------------------------------
-----


Model Report:  MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta
_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_iter=200, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
Accuracy : 0.8849
AUC (Train): 0.755744
CV Score : Mean - 0.656424 | Std - 0.1227592 | Min - 0.4268348 | Max - 0.7951748

-----------------------------------**********-------------------------------
-----


Model Report:  KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski
',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
Accuracy : 0.9086
AUC (Train): 0.919969
CV Score : Mean - 0.7091629 | Std - 0.02725986 | Min - 0.6819858 | Max - 0.7565057

-----------------------------------**********-------------------------------
-----


Model Report:  AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_
rate=1.0,
                   n_estimators=50, random_state=None)
Accuracy : 0.8943
AUC (Train): 0.804969
CV Score : Mean - 0.778557 | Std - 0.02673526 | Min - 0.7455909 | Max - 0.817502

-----------------------------------**********-------------------------------
-----


Model Report:  LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=N
one,
                           solver='svd', store_covariance=False, tol=0.0001)
Accuracy : 0.8839
```

```
AUC (Train): 0.760019
CV Score : Mean - 0.755193 | Std - 0.03513661 | Min - 0.7081871 | Max - 0.7930021

------------------------------------**********------------------------------
-----


Model Report:  QuadraticDiscriminantAnalysis(priors=None, reg_param=0.0,
                               store_covariance=False, tol=0.0001)
Accuracy : 0.8665
AUC (Train): 0.781758
CV Score : Mean - 0.7752025 | Std - 0.02836658 | Min - 0.7324003 | Max - 0.8112958

------------------------------------**********------------------------------
-----


Model Report:  XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='binary:logistic', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=None,
              validate_parameters=False, verbosity=None)
Accuracy : 0.9453
AUC (Train): 0.964400
CV Score : Mean - 0.7469788 | Std - 0.01359171 | Min - 0.7237839 | Max - 0.7621521

------------------------------------**********------------------------------
-----


Model Report:  <catboost.core.CatBoostClassifier object at 0x00000144852C7888>
Accuracy : 0.9284
AUC (Train): 0.892010
CV Score : Mean - 0.7772238 | Std - 0.02356154 | Min - 0.7478137 | Max - 0.8157148

------------------------------------**********------------------------------
-----
```

The 5 selected models for grid search:

1) Logistic Regression
2) Decision Tree
3) Random Forests
4) CatBoost (Gradient Boosting variant)
5) K-nearest neighbors

All the models have been tuned on various hyper parameters of the algorithm.

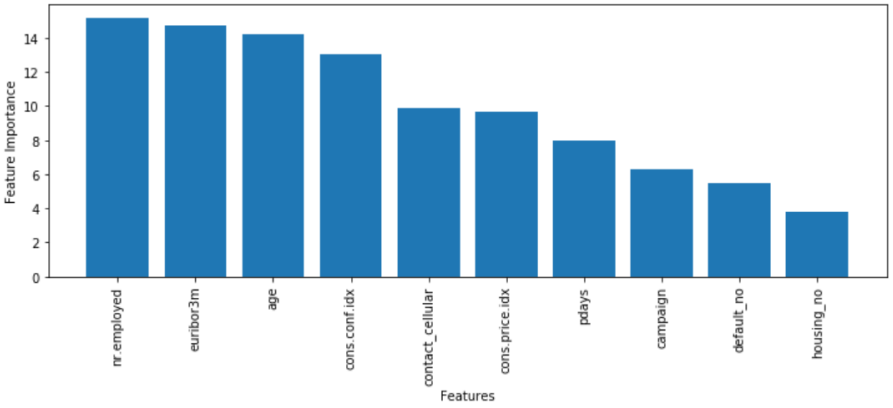
After the grid search and cross validation. CatBoost model had the best cv score.

```
{'depth': 5, 'iterations': 600, 'learning_rate': 0.05},
 0.7812443663008256)
```

** For computational purpose the code of grid search for other algorithms has been removed**

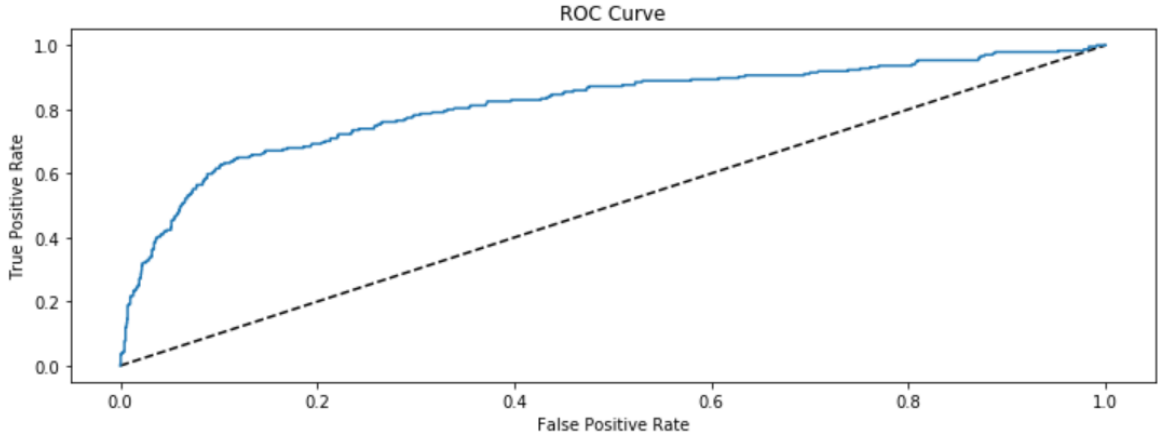
In depth analysis of CatBoost model:

Feature Importance :



Result on validation:

```
<catboost.core.CatBoostClassifier object at 0x0000014489A24C48>
[[1845   21]
 [ 183   51]]
AUC score:  0.812881890052308
Logloss :  0.2686845646954953
```



At the end you can find other models and a lot of experimentation done prior to finalizing the model to use.