What are the techniques you will implement to improve the classification Accuracy?

Ensemble methods : Bagging, Random Forest, Boosting

Bagging has a single parameter, which is the number of trees. All trees are fully grown binary tree (unpruned) and at each node in the tree one searches over all features to find the feature that best splits the data at that node.  
  
Random forests has 2 parameters:

1. The first parameter is the same as bagging (the number of trees)
2. The second parameter (unique to randomforests) is mtry which is how many features to search over to find the best feature. this parameter is usually 1/3\*D for regression and sqrt(D) for classification. thus during tree creation randomly mtry number of features are chosen from all available features and the best feature that splits the data is chosen.
3. The fundamental difference is that in Random forests, only a subset of features are selected at random out of the total and the best split feature from the subset is used to split each node in a tree, unlike in bagging where all features are considered for splitting a node.
4. Random Forests improve variance by reducing correlation between trees, this is accomplished by random selection of feature-subset for split at each node. `mtry` is the tuning parameter that defines #\_of\_features to be selected.
5. While Bagging improves variance by averaging/majority selection of outcome from multiple fully grown trees on variants of training set. It uses Bootstrap with replacement to generate multiple training sets.

<https://www.analyticsvidhya.com/blog/2017/02/introduction-to-ensembling-along-with-implementation-in-r/>

<https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/>

2. What is class imbalance problem and list out 5 methods to fix

#### **Random Under-Sampling**

#### **Random Over-Sampling**

#### **Cluster-Based Over Sampling**

#### **Informed Over Sampling: Synthetic Minority Over-sampling Technique**

**SMOTE**

<https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/>

3.Explain the concept of Gradient descent and its importance in Machine learning?

## **Gradient Descent**

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).

Gradient descent is best used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm

gradient descent is a general technique - if you know it, you can in principle solve any ML problem that comes with differentiable error function.

Many powerful machine learning algorithms use gradient descent optimization to identify patterns and learn from data. Gradient descent  powers machine learning algorithms such as linear regression, logistic regression, neural networks, and support vector machines.

Most of the data science algorithms are optimization problems and one of the most used algorithms to do the same is the Gradient Descent Algorithm.

<https://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html>

<https://www.analyticsvidhya.com/blog/2017/03/introduction-to-gradient-descent-algorithm-along-its-variants/>

<https://machinelearningmastery.com/gradient-descent-for-machine-learning/>

4. What is Bias and Variance concept in Machine learning and list out the approaches to fix them

The prediction error for any machine learning algorithm can be broken down into three parts:

* Bias Error
* Variance Error
* Irreducible Error

The irreducible error cannot be reduced regardless of what algorithm is used. It is the error introduced from the chosen framing of the problem and may be caused by factors like unknown variables that influence the mapping of the input variables to the output variable

Bias measures how far off in general these models' predictions are from the correct value.

## **Bias Error**(sse : sum of error : actual – predicted : more number of observations)

.Generally, parametric algorithms have a high bias making them fast to learn and easier to understand but generally less flexible. In turn, they have lower predictive performance on complex problems that fail to meet the simplifying assumptions of the algorithms bias.

* **Low Bias**: Suggests less assumptions about the form of the target function.
* **High-Bias**: Suggests more assumptions about the form of the target function.

Examples of low-bias machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

Examples of high-bias machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

## Variance Error(ssr: sum of square regression : difference between predicted and average value: more number of variables/features)

It makes model overfitting and create noise and sensitivity for unseen data

Variance is the amount that the estimate of the target function will change if different training data was used.

* **Low Variance**: Suggests small changes to the estimate of the target function with changes to the training dataset.
* **High Variance**: Suggests large changes to the estimate of the target function with changes to the training dataset.

Generally, nonparametric machine learning algorithms that have a lot of flexibility have a high variance. For example, decision trees have a high variance, that is even higher if the trees are not pruned before use.

Examples of low-variance machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

Examples of high-variance machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

## Bias-Variance Trade-Off

The goal of any supervised machine learning algorithm is to achieve low bias and low variance. In turn the algorithm should achieve good prediction performance.

Below are two examples of configuring the bias-variance trade-off for specific algorithms:

* The k-nearest neighbors algorithm has low bias and high variance, but the trade-off can be changed by increasing the value of k which increases the number of neighbors that contribute t the prediction and in turn increases the bias of the model.
* The support vector machine algorithm has low bias and high variance, but the trade-off can be changed by increasing the C parameter that influences the number of violations of the margin allowed in the training data which increases the bias but decreases the variance.

There is no escaping the relationship between bias and variance in machine learning.

* Increasing the bias will decrease the variance.
* Increasing the variance will decrease the bias.

[Dimensionality reduction](https://en.wikipedia.org/wiki/Dimensionality_reduction) and [feature selection](https://en.wikipedia.org/wiki/Feature_selection) can decrease variance by simplifying models. Similarly, a larger training set tends to decrease variance. Adding features (predictors) tends to decrease bias, at the expense of introducing additional variance. Learning algorithms typically have some tunable parameters that control bias and variance, e.g.:

* ([Generalized](https://en.wikipedia.org/wiki/Generalized_linear_model)) linear models can be [regularized](https://en.wikipedia.org/wiki/Regularization_(mathematics)) to decrease their variance at the cost of increasing their bias.[[11]](https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff#cite_note-11)
* In [artificial neural networks](https://en.wikipedia.org/wiki/Artificial_neural_network), the variance increases and the bias decreases with the number of hidden units.[[1]](https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff#cite_note-geman-1) Like in GLMs, regularization is typically applied.
* In [k-nearest neighbor](https://en.wikipedia.org/wiki/K-nearest_neighbor) models, a high value of *k* leads to high bias and low variance (see below).
* In [Instance-based learning](https://en.wikipedia.org/wiki/Instance-based_learning), regularization can be achieved varying the mixture of [prototypes](https://en.wikipedia.org/wiki/Prototype) and exemplars.[[12]](https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff#cite_note-12)
* In [decision trees](https://en.wikipedia.org/wiki/Decision_tree), the depth of the tree determines the variance. Decision trees are commonly pruned to control variance.[[4]](https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff#cite_note-islr-4):307

One way of resolving the trade-off is to use [mixture models](https://en.wikipedia.org/wiki/Mixture_models) and [ensemble learning](https://en.wikipedia.org/wiki/Ensemble_learning).[[13]](https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff#cite_note-13)[[14]](https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff#cite_note-14) For example, [boosting](https://en.wikipedia.org/wiki/Boosting_(machine_learning)) combines many "weak" (high bias) models in an ensemble that has lower bias than the individual models, while [bagging](https://en.wikipedia.org/wiki/Bootstrap_aggregating) combines "strong" learners in a way that reduces their variance.

<https://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning/>

<https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff>

* 1. Explain a. K-Fold validation
  2. b. Knn Classification
  3. **K-fold cross validation** : The data set is divided into *k* subsets, and the holdout method is repeated *k* times. Each time, one of the *k* subsets is used as the test set and the other *k-1* subsets are put together to form a training set. Then the average error across all *k* trials is computed. The advantage of this method is that it matters less how the data gets divided. Every data point gets to be in a test set exactly once, and gets to be in a training set *k-1* times. The variance of the resulting estimate is reduced as *k* is increased. The disadvantage of this method is that the training algorithm has to be rerun from scratch *k* times, which means it takes *k* times as much computation to make an evaluation. A variant of this method is to randomly divide the data into a test and training set *k*different times. The advantage of doing this is that you can independently choose how large each test set is and how many trials you average over.

|  |  |  |
| --- | --- | --- |
| K Nearest Neighbors - Classification |  |  |
| K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition   |  |  |  | | --- | --- | --- | | **Algorithm** |  |  | | A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor. |  |  | |  |  |  | | http://www.saedsayad.com/images/KNN_similarity.png |  |  | |  |  |  | | It should also be noted that all three distance measures are only valid for continuous variables. In the instance of categorical variables the Hamming distance must be used. It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset. |  |  | |  |  |

<https://www.analyticsvidhya.com/blog/2014/10/introduction-k-neighbours-algorithm-clustering/>

Wilson editing used to remove outliers and noise so that misclassification can be avoided.

* 1. What are the parameters that you will tune for each of the algorithms?
  2. a. Random forest : max\_features(mtry), n\_estimators(ntree), min\_sample leaf
  3. <https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/>

## b. CART : Complexity (cp)

* 1. The others could make a difference, but the complexity parameter by and large is the fundamental driver for over/under-fitting.
  2. c. C5.0 : The minCases parameter specifies the minimum number of cases.
  3. Essentially it controls the depth of the trees created by C5.0 (depth cannot be controlled directly) and hence it is intimately connected with the resulting tree complexity.

1. The purpose of tuning meta parameters is to find the optimal trade-off between model complexity and the training set size and so minCases is an important parameter to tune. That said, tuning minCases is problematic under cross-validation because the number of cases in the training folds are different than the number of cases in the entire dataset so the optimal value of minCases found in cross validation will not be equal to the true optimum for the entire data set (which the final model will be trained on). To overcome this obstacle we can define minCases as a proportion of the data set size and tune the proportional parameter instead. If we define minCases as

minCases <- length(y)/splits

then as "splits" increases, so will the depth and complexity of the resulting trees. The code below customizes the standard caret functions to allow for the tuning of "splits" along with the other C5.0 meta parameters

* 1. d. Ada Boost :
  2. The important parameters to vary in an AdaBoost regressor are learning\_rate and loss

## **GBM Parameters** The overall parameters can be divided into 3 categories:

1. **Tree-Specific Parameters:** These affect each individual tree in the model.
2. **Boosting Parameters:** These affect the boosting operation in the model.
3. **Miscellaneous Parameters:** Other parameters for overall functioning.

Python attributes, r has similar one..idea remains same

1. **min\_samples\_split**
   1. Defines the minimum number of samples (or observations) which are required in a node to be considered for splitting.
   2. Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
   3. Too high values can lead to under-fitting hence, it should be tuned using CV.
2. **min\_samples\_leaf**
   1. Defines the minimum samples (or observations) required in a terminal node or leaf.
   2. Used to control over-fitting similar to min\_samples\_split.
   3. Generally lower values should be chosen for imbalanced class problems because the regions in which the minority class will be in majority will be very small.
3. **min\_weight\_fraction\_leaf**
   1. Similar to min\_samples\_leaf but defined as a fraction of the total number of observations instead of an integer.
   2. Only one of #2 and #3 should be defined.
4. **max\_depth**
   1. The maximum depth of a tree.
   2. Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.
   3. Should be tuned using CV.
5. **max\_leaf\_nodes**
   1. The maximum number of terminal nodes or leaves in a tree.
   2. Can be defined in place of max\_depth. Since binary trees are created, a depth of ‘n’ would produce a maximum of 2^n leaves.
   3. If this is defined, GBM will ignore max\_depth.
6. **max\_features**
   1. The number of features to consider while searching for a best split. These will be randomly selected.
   2. As a thumb-rule, square root of the total number of features works great but we should check upto 30-40% of the total number of features.
   3. e. XG Boost

## **Control Overfitting**

When you observe high training accuracy, but low tests accuracy, it is likely that you encounter overfitting problem.

There are in general two ways that you can control overfitting in xgboost

* The first way is to directly control model complexity
  + This include max\_depth, min\_child\_weight and gamma
* The second way is to add randomness to make training robust to noise
  + This include subsample, colsample\_bytree
  + You can also reduce stepsize eta, but needs to remember to increase num\_round when you do so.

## **Handle Imbalanced Dataset**

For common cases such as ads clickthrough log, the dataset is extremely imbalanced. This can affect the training of xgboost model, and there are two ways to improve it.

* If you care only about the ranking order (AUC) of your prediction
  + Balance the positive and negative weights, via scale\_pos\_weight
  + Use AUC for evaluation
* If you care about predicting the right probability
  + In such a case, you cannot re-balance the dataset
  + In such a case, set parameter max\_delta\_step to a finite number (say 1) will help convergence
  + <http://xgboost.readthedocs.io/en/latest/how_to/param_tuning.html>
  1. <https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/>
  2. f. Support Vector Machines

## **C**

Is a regularization parameter, that a proper value is chosen for C, the penalty factor. If it is too large, we have a high penalty for nonseparable points and we may store many support vectors and overfit. If it is too small, we may have underfitting."  
"...the coefficient *C* affects the trade-off between complexity and proportion of nonseparable samples and must be selected by the user."  
The parameter *C* controls the trade off between errors of the SVM on training data and margin maximization (*C* = ∞ leads to hard margin SVM).

**Block 2: Collaborative Filtering**

What are recommendation Engines and the types of the recommendation engines

A recommendation engine, also known as a recommender system, is software that analyzes available data to make suggestions for something that a website user might be interested in, such as a book, a video or a job, among other possibilities.

Recommendation engines use a variety of technologies and techniques that enable them to filter large amounts of data and provide a smaller, focused body of suggestions for the user. Netflix, for example, uses metadata tagging on videos in conjunction with data about user behavior to come up with recommended movies and TV shows for specific members. [LinkedIn](http://whatis.techtarget.com/definition/LinkedIn) uses the semi-structured data provided by members, including things like locations, job titles, skill sets and industries, to fuel their "Jobs you might be interested in" section.

types of the recommendation engines

through [collaborative filtering](https://en.wikipedia.org/wiki/Collaborative_filtering) or through [content-based filtering](https://en.wikipedia.org/wiki/Content-based_filtering) (also known as the personality-based approach).[[7]](https://en.wikipedia.org/wiki/Recommender_system#cite_note-7) [Collaborative filtering](https://en.wikipedia.org/wiki/Collaborative_filtering) approaches build a model from a user's past behaviour (items previously purchased or selected and/or numerical ratings given to those items) as well as similar decisions made by other users. This model is then used to predict items (or ratings for items) that the user may have an interest in.[[8]](https://en.wikipedia.org/wiki/Recommender_system#cite_note-Recommender2010-8) [Content-based filtering](https://en.wikipedia.org/wiki/Content-based_filtering) approaches utilize a series of discrete characteristics of an item in order to recommend additional items with similar properties.[[9]](https://en.wikipedia.org/wiki/Recommender_system#cite_note-mooney99-9) These approaches are often combined (see [Hybrid Recommender Systems](https://en.wikipedia.org/wiki/Recommender_system#Hybrid_recommender_systems)).

## **Description**

Collaborative filtering is one of the most effective methods to provide recommendations to a users. Based on the behavior exhibited by a set of users in buying the buying patterns of users for a set of products in their transactions, a collaborative filtering approach is implemented.

#### WHAT ARE THE DIFFERENT TYPES OF RECOMMENDATION ENGINES?

Let me introduce you to three very important types of recommender systems:

* Collaborative Filtering
* Content-Based Filtering
* Hybrid Recommendation Systems

#### WHAT IS A RECOMMENDATION ENGINE ?

**Wiki Definition:**Recommendation Engines are a subclass of [*information filtering system*](http://en.wikipedia.org/wiki/Information_filtering_system) that seek to predict the ‘rating’ or ‘preference’ that user would give to an item.

**dataaspirant Definition:** Recommendation Engine is a black box which analysis some set of users and shows the items which a single user may like.

<http://dataconomy.com/2015/03/an-introduction-to-recommendation-engines/>

1. How are KNN and collaborative filtering related to each other?

Collaborative filtering can be done using different ways.and one of the ways is using knn.

kNN is a machine learning algorithm to find clusters of similar users based on ratings, and make predictions using the average rating of top-k nearest neighbors.

With the development of electronic commerce, Collaborative Filtering Recommendation system emerge, which uses machine learning algorithms for people provide a set of N items that will be of interest. In many user-based collaborative filtering applications based on KNN(K nearest neighbor algorithm),

**Just a note**:

*KNN is non-parametric, instance-based and used in a supervised learning setting.*

It is worth noting that the minimal training phase of KNN comes both at a memory cost, since we must store a potentially huge data set, as well as a computational cost during test time since classifying a given observation requires a run down of the whole data set. Practically speaking, this is undesirable since we usually want fast responses.

*Minimal training but expensive testing.*

Does KNN work well for high dimensional datasets? Why or why not?

KNN works well with a small number of input variables (p), but struggles when the number of inputs is very large.

As the number of dimensions increases the volume of the input space increases at an exponential rate.

In high dimensions, points that may be similar may have very large distances. All points will be far away from each other and our intuition for distances in simple 2 and 3-dimensional spaces breaks down. This might feel unintuitive at first, but this general problem is called the “[Curse of Dimensionality](https://en.wikipedia.org/wiki/Curse_of_dimensionality)“.

## Best Prepare Data for KNN

* **Rescale Data**: KNN performs much better if all of the data has the same scale. Normalizing your data to the range [0, 1] is a good idea. It may also be a good idea to standardize your data if it has a Gaussian distribution.
* **Address Missing Data**: Missing data will mean that the distance between samples can not be calculated. These samples could be excluded or the missing values could be imputed.
* **Lower Dimensionality**: KNN is suited for lower dimensional data. You can try it on high dimensional data (hundreds or thousands of input variables) but be aware that it may not perform as well as other techniques. KNN can benefit from feature selection that reduces the dimensionality of the input feature space.

**Block 3: Decision Trees**

Why is tree pruning useful in Decision trees?

Over-fitting is the phenomenon in which the learning system tightly fits the given training data so much that it would be inaccurate in predicting the outcomes of the untrained data.

In decision trees, over-fitting occurs when the tree is designed so as to perfectly fit all samples in the training data set. Thus it ends up with branches with strict rules of sparse data. Thus this effects the accuracy when predicting samples that are not part of the training set.

One of the methods used to address over-fitting in decision tree is called **pruning**

**Pruning** is a technique in [machine learning](https://en.wikipedia.org/wiki/Machine_learning) that reduces the size of [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning) by removing sections of the tree that provide little power to classify instances. Pruning reduces the complexity of the final [classifier](https://en.wikipedia.org/wiki/Statistical_classification), and hence improves predictive accuracy by the reduction of [overfitting](https://en.wikipedia.org/wiki/Overfitting).

Tree pruning methods address this problem of overfittingthe data.

Extra notes

Tree pruning attempts to identify and remove such branches, with the goal of improving classification accuracy on unseen data.

Pruned trees

• These tend to be smaller and less complex and, thus, easier to comprehend.

• They are usually faster and better at correctly classifying independent test data than unpruned trees.

• Pruned trees tend to be more compact than their unpruned counterparts

There are two common approaches to tree pruning:

1. prepruning :

♣ In the pre-pruning approach, a tree is “pruned” by halting its construction early (e.g. by deciding not to further split or partition the subset of training tuples at a given node).

♣ When constructing a tree, measures such as statistical significance, information gain, Gini index, and so on can be used to assess the goodness of a split.

♣ If partitioning the tuples at a node would result in a split that falls below a pre specified threshold, then further partitioning of the given subset is halted.

♣ There are difficulties, however, in choosing an appropriate threshold.

♣ High thresholds could result in oversimplified trees, whereas low thresholds could result in very little simplification.

1. post pruning.

♣ The second and more common approach is post pruning, which removes subtrees from a “fully grown” tree.

♣ A subtree at a given node is pruned by removing its branches and replacing it with a leaf.

♣ The leaf is labeled with the most frequent class among the subtree being replaced.

♣ The cost complexity pruning algorithm used in CART is an example of the post pruning approach.

♣ The basic idea is that the simplest solution is preferred.

♣ Unlike cost complexity, pruning does not require an independent set of tuples.

♣ Post pruning leads to a more reliable tree.

Machine learning is a problem of trade-offs. The classic issue is overfitting versus underfitting. Overfitting happens when a model memorizes its training data so well that it is learning noise on top of the signal. Underfitting is the opposite: the model is too simple to find the patterns in the data. Simplicity versus accuracy is a similar consideration.

<https://www.displayr.com/machine-learning-pruning-decision-trees/>

* *Minimum error*. The tree is pruned back to the point where the cross-validated error is a minimum. *Cross-validation* is the process of building a tree with most of the data and then using the remaining part of the data to test the accuracy of the tree.
* *Smallest tree*. The tree is pruned back slightly further than the minimum error. Technically the pruning creates a tree with cross-validation error within 1 standard error of the minimum error. The smaller tree is more intelligible at the cost of a small increase in error.
* None.
* A decision tree is a machine learning algorithm that partitions the data into subsets. The partitioning process starts with a binary split and continues until no further splits can be made. Various branches of variable length are formed.
* The goal of a decision tree is to encapsulate the training data in the smallest possible tree. The rationale for minimizing the tree size is the logical rule that the simplest possible explanation for a set of phenomena is preferred over other explanations. Also, small trees produce decisions faster than large trees, and they are much easier to look at and understand. There are various methods and techniques to control the depth, or prune, of the tree.
* **Pruning.** The shortening of branches of the tree. Pruning is the process of reducing the size of the tree by turning some branch nodes into leaf nodes, and removing the leaf nodes under the original branch. Pruning is useful because classification trees may fit the training data well, but may do a poor job of classifying new values. Lower branches may be strongly affected by outliers. Pruning enables you to find the next largest tree and minimize the problem. A simpler tree often avoids over-fitting.

1. What is one problem associated with using information gain in decision tree learning?

Information gain is biased toward choosing features with a large number of values – The selection of a feature that is non-optimal for predication can result in overfitting.

Although information gain is usually a good measure for deciding the [relevance](https://en.wikipedia.org/wiki/Relevance) of an attribute, it is not perfect. A notable problem occurs when information gain is applied to attributes that can take on a large number of distinct values. For example, suppose that one is building a decision tree for some data describing the customers of a business. Information gain is often used to decide which of the attributes are the most relevant, so they can be tested near the root of the tree. One of the input attributes might be the customer's credit card number. This attribute has a high mutual information, because it uniquely identifies each customer, but we do *not* want to include it in the decision tree: deciding how to treat a customer based on their credit card number is unlikely to generalize to customers we haven't seen before ([overfitting](https://en.wikipedia.org/wiki/Overfitting)).

[Information gain ratio](https://en.wikipedia.org/wiki/Information_gain_ratio) is sometimes used instead. This biases the decision tree against considering attributes with a large number of distinct values. However, attributes with very low information values then appeared to receive an unfair advantage.

* GAINsplit is penalized when large number of small partitions are produced by the split!
* SplitINFO increases when a larger number of small partitions is produced
* Designed to overcome the disadvantage of Information Gain