**The Purpose of this document is to gain overall Theoretical and Practical Knowledge of Machine Learning Algorithms to qualify for Interviews.**

**Confusion Matrix**

Sensitivity/Recall/True positive rate: TP/(TP + FN)

Precision/Positive Predictive value/false positive rate: TP/(TP+FP)

Specificity:TN/(FP + TN)

Sensitivity and recall are same thing but precision and specificity are not.

True positive = correctly identified

True negative = correctly rejected

False positive/False Alarm : A cancer screening test comes back positive, but you don’t have the disease.

False negative : A false negative is where **a negative test result is wrong**.

Explanation False Negative

False negatives create two problems. The first is a false sense of security. For example, if your manufacturing line doesn’t catch your defective items, you may think the process is running more effectively than it actually is. The second, potentially more serious issue, is that potentially dangerous situations may be missed. For example, a crippling computer virus can wreak havoc if not detected, or an individual with cancer may not receive timely treatment.

Sensitivity and Specificity explanation with Example

A model that always predicts cancer will have a sensitivity of 1 and a specificity of 0. A model that never predicts cancer will have a sensitivity of 0 and a specificity of 1. An ideal model should have both a sensitivity of 1 and a specificity of 1. In reality, however, this is unlikely to be achievable. Therefore, we should look for a model that achieves a good tradeoff between specificity and sensitivity. So which one of the two is more important? This can’t be said in general. It highly depends on the application.

If you build a photo-based skin cancer detection app, then a high sensitivity is probably more important than a high specificity, since you want to cause people who might have cancer to get themselves checked by a doctor. Specificity is a little less important here, but still, if you detect cancer too often, people might stop using your app since they unnecessarily get annoyed and scared.

An Intuitive Explanation of F-score/ balanced F-score/ F-measure

***F1 Score*** is the Harmonic mean of Recall and Precision -  2\*((precision\*recall)/(precision+recall))

Imagine that you're trying to classify politicians into two groups: those who are honest, and those who are not. I'll give you a list of 100 people, half of which are honest, and you'll give me a list of **all** the honest ones, but being careful not to accidentally include anyone who is not.

Having high ***precision*** **means** that when you do say that someone is honest, you're usually right about it. This is about how many politicians in your list are *actually* honest, out of all the ones that you've added to the list.

Having high ***recall* means** that you can identify most of the honest politicians out there. This is about how many honest politicians you've added to your list, out of all the ones that exist.

Note that these are not the same. You could add *a single honest politician* to the list. You would then have very high precision, since the only person that you've listed *is actually honest*; but you would also have very low recall: there are 50 honest politicians out there but you've only mentioned one.

Ideally, you'd want to list all honest politicians that exist while being careful to not accidentally include some who are not. If you could do that, then you'd have both high precision and high recall.

When measuring how well you're doing, it's often useful to have a single number to describe your performance. We could define that number to be, for instance, the mean of your precision and your recall. This is exactly what the F1-score is.

The only reason why we use the harmonic mean is because we're taking the average of ratios (percentages), and in that case the harmonic mean is more appropriate than the arithmetic mean.

When to use what curve ROC curve & PR curve (precision recall curve):

Particularly, if true negative is not much valuable to the problem, or negative examples are abundant. Then, PR-curve is typically more appropriate. In other words, it is used when positive class is more interesting than the negative class.

1.One example may be fraud detection, where non-fraud sample may be 10000 and fraud sample may be below 100.

2.So, if your problem involves kind of searching a needle in the haystack when for ex: the positive class samples are very rare compared to the negative clas ses, use a precision recall curve. Othwerwise use a ROC curve

<https://acutecaretesting.org/en/articles/precision-recall-curves-what-are-they-and-how-are-they-used>

Overfitting:

what overfitting actually is. Models are normally trained to fit a dataset by minimizing some loss function on a training set. There is however a limit where minimizing this training error will no longer benefit the models true performance, but only minimize the error on the specific set of data. This essentially means that the model has been too tightly fitted to the specific data points in the training set, trying to model patterns in the data originating from noise. This concept is called *overfit*.

**So, In the above explanation we covered some of the classification evaluation metrics for imbalanced data: Recall/Sensitivity, Precision, Specificity and f1 score. down the line we will see some more evaluation metrics for classification algorithms.**

 Logarithmic Loss

<http://www.exegetic.biz/blog/2015/12/making-sense-logarithmic-loss/>

Log Loss quantifies the accuracy of a classifier by penalising false classifications. Minimising the Log Loss is basically equivalent to maximising the accuracy of the classifier, Log Loss heavily penalises classifiers that are confident about an incorrect classification

ROC curve

Show down 10 algorithm with visualisation

<https://www.kaggle.com/jeffd23/10-classifier-showdown-in-scikit-learn>

stratified Cross Validation

Stratified Cross-validation violates the principal that the test labels should never have been looked at before the statistics are calculated, but this is generally thought to be innocuous as the only effect is to balance the folds, but it does lead to loss of diversity (an unwanted loss of variance). It moves even further from the Boostrap idea of constructing a sample similar to what you'd draw naturally from the whole population. Arguably the main reason stratification is important is to address defects in the classification algorithms, as they are too easily biased by over- or under-representation of classes. An algorithm that uses balancing techniques (either by selection or weighting) or optimizes a chance-correct measure (Kappa or preferably Informedness) is less impacted by this, although even such algorithms can't learn or test a class that isn't there.

Forcing each fold to have at least m instances of each class, for some small m, is an alternative to stratification that works for both Bootstrapping and CV. It does have a smoothing bias, making folds tend to be more balanced than they would otherwise be expected to be. (used for unbalanced class)

**we discussed about stratified Cross validation, log loss and implemented the code from leaf classification. after this we will discuss about cross validation and grid search validation and we will understand hyper parameter optimisation or we can say parameter tuning.**

Hyperparameter Tuning

It is always good to use a mix of random and grid searching to expose “good” regions of the hyperparameter search space.

You can use random search to find good starting points, then grid search to zoom in and find the local optima (or close to it) for those good starting points. Using the two approaches interchangeably like a manual optimization algorithm.

Grid Search Parameter Tuning

Grid search is an approach to parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid.The recipe below evaluates different alpha values for the Ridge Regression algorithm on the standard diabetes dataset. This is a one-dimensional grid search.

## Random Search Parameter Tuning

Random search is an approach to parameter tuning that will sample algorithm parameters from a random distribution (i.e. uniform) for a fixed number of iterations. A model is constructed and evaluated for each combination of parameters chosen.The recipe below evaluates different alpha random values between 0 and 1 for the Ridge Regression algorithm on the standard diabetes dataset.

<https://www.pyimagesearch.com/2016/08/15/how-to-tune-hyperparameters-with-python-and-scikit-learn/>

Calibratedclassifiercv

calibration will improve logloss

Stability selection and recursive feature elimination - I will read

**we will work out on classification first as we have learned some of the evaluation metrics to evaluate the model:**

**Basically starting from precision, recall, F1 score, AUC score, roc curve, log loss. Above parameters can tell u whether your model is predicting well or not.**

**Decision Trees/Random forest**

“which attribute should we divide the data with?” The answer is that it should divide the data by the best attribute, but what does “best” actually mean?

For ID3, we think of best in terms of which attribute has the most *information gain*, a measure that expresses how well an attribute splits the data into groups based on classification.

To actually calculate information gain, we must first take a look at another measure, *entropy*. Entropy, in an information theory and machine learning sense, measures the homogeneity of a data set *S*’s classifications. It ranges from 0, which means that all of the classifications in the data set are the same, to log2 of the number of different classifications, which means that the classifications are equally distributed within the data set. In our *PlayTennis* example, which has 2 different classifications (*yes* and *no*), the maximum entropy of the training data is log2(2) = 1. If all of the training data tells us yes, the entropy is 0. If all of it tells us no, the entropy is still 0. If there are equal numbers of yes and no examples, the entropy is 1. Since there are 9 yes examples and 5 no examples in our table, its entropy lies somewhere between 0 and 1. We’ll have to calculate it using the formula for entropy, which is:

In this formula, *c* corresponds to the number of different classifications/classes and *pi* corresponds to the proportion of the data with the classification/classes *i*. Because our example and the basic version of the ID3 algorithm both deal with the case where classifications are either positive or negative, we can simplify the formula to:

Here, *p+* is the proportion of examples with a positive classification/classes and *p­* is the proportion of examples with a negative classification/classes.

In the case of *PlayTennis*, there are 9 *yes* rows and 5 *no* rows, which leads to an entropy of:

Information gain measures the reduction in entropy that results from partitioning the data on an attribute *A*, which is another way of saying that it represents how effective an attribute is at classifying the data. Given a set of training data *S* and an attribute *A*, the formula for information gain is:

The entropies of the partitions, when summed and weighted, can be compared to the entropy of the entire data set. The first term corresponds to the entropy of the data before the partitioning, whereas the second term corresponds to the entropy afterwards. We want to maximize information gain, so we want the entropies of the partitioned data to be as low as possible, which explains why attributes that exhibit high information gain split training data into relatively heterogeneous groups.

Entropy is computationally intensive as it involves log.

GainRatio tries to correct for information gain’s natural bias toward attributes with many possible values by adding a denominator to information gain called SplitInformation:

*Reference: Udacity Decision Trees Pdf ^*

**Combining Decision Tress/Prediction - Random Forest**

When we have multiple classifiers making predictions, we can treat each set of predictions as a column in a matrix. Here's an example where we have Decision Tree 1 ( DT1 ), Decision Tree 2 ( DT2 ), and Decision Tree 3 ( DT3 ):

Whenever we add more models to our ensemble, we just add more columns to the combined predictions. Ultimately, we don't want this matrix, though -- we want one prediction per row in the training data. To accomplish this, we'll need to create rules to convert each row of our matrix of predictions into a single number. We want to create a Final Prediction vector that looks like this:

There are many ways to get from the output of multiple models to a final vector of predictions. One method is majority voting, in which each classifier gets a "vote," and the most commonly voted value for each row "wins." This only works if there are more than two classifiers (and ideally an odd number, so we don't have to write a rule to break ties). Majority voting is what we applied in the example above.

There are three main ways to combat overfitting:

1. "Prune" the tree after we build it to remove unnecessary leaves.
2. Use ensembling to blend the predictions of many trees.
3. Restrict the depth of the tree while we're building it.

Similar to decision trees, we can tweak some of the parameters for random forests to restrict the depth of the trees:

max\_depth - Globally restricts how deep the tree can go

min\_samples\_split - The minimum number of rows a node should have before it can be split; if this is set to 2 , for example, then nodes with 2 rows won't be split, and will become leaves instead

min\_samples\_leaf - The minimum number of rows a leaf must have

min\_weight\_fraction\_leaf - The fraction of input rows a leaf must have

max\_leaf\_nodes - The maximum number of total leaves; this will cap the count of leaf nodes as the tree is being built

clf = RandomForestClassifier(n\_estimators=5, random\_state=1, min\_samples\_leaf=50, splitter="random", max\_features="auto")

We can also repeat our random subset selection process in scikit-learn. We just set the splitterparameter on DecisionTreeClassifier to "random" , and the max\_features parameter to "auto" . If we have N columns, this will pick a subset of features of size √ ̅*N* ̅, compute the Gini coefficient for each (this is similar to information gain), and split the node on the best column in the subset.

**Logistic Regression**

Must go through the link <https://machinelearningmastery.com/logistic-regression-for-machine-learning/> after reading here

Logistic regression is used to find the probability of event=Success and event=Failure. We should use logistic regression when the dependent variable is binary (0/ 1, True/ False, Yes/ No) in nature. It can represented by following equation.

Since we are working here with a binomial distribution (dependent variable), we need to choose a link function which is best suited for this distribution. And, it is [**logit**](https://en.wikipedia.org/wiki/Logistic_function)/Sigmoid function. In the equation above, the parameters are chosen to maximize the likelihood of observing the sample values rather than minimizing the sum of squared errors (like in ordinary regression).

The intuition for maximum-likelihood for logistic regression is that a search procedure seeks values for the coefficients (Beta values b0,b1..) that minimize the error in the probabilities predicted by the model to those in the data (e.g. probability of 1 if the data is the primary class).

It is enough to say that a minimization algorithm is used to optimize the best values for the coefficients

(Beta values b0,b1..) for

 your training data.

The key representation in logistic regression are the coefficients, just like linear regression. The coefficients in logistic regression are estimated using a process called maximum-likelihood estimation.

* Logistic regression doesn’t require linear relationship between dependent and independent variables.  It can handle various types of relationships because it applies a non-linear log transformation to the predicted odds ratio
* To avoid over fitting and under fitting, we should include all significant variables. A good approach to ensure this practice is to use a step wise method to estimate the logistic regression
* It requires large sample sizes because maximum likelihood estimates are less powerful at low sample sizes than ordinary least square
* The independent variables should not be correlated with each other i.e. no multi collinearity.  However, we have the options to include interaction effects of categorical variables in the analysis and in the model.
* If the values of dependent variable is ordinal, then it is called as Ordinal logistic regression
* If dependent variable is multi class then it is known as Multinomial Logistic regression.

Evaluation parameter same as classification +

AIC - akaike information criteria analogous to Rsquared: AIC is a goodness of fit measure that favours smaller residual error in the model, but penalises for including further predictors and helps avoiding overfitting.

# An intuitive explanation why multicollinearity is a problem in Linear regression  -This part we will add when we will go through Linear regression

Consider the simplest case where Y is regressed against X and Z and where X and Z are highly positively correlated. Then the effect of X on Y is hard to distinguish from the effect of Z on Y because any increase in X tends to be associated with an increase in Z.

Another way to look at this is to consider the equation. If we write Y= b0 + b1X + b2Z + e, then the coefficient b1 is the increase in Y for every unit increase in X while holding Z constant. But in practice, it is often impossible to hold Z constant and the positive correlation between X and Z mean that a unit increase in X is usually accompanied by some increase in Z at the same time.

A similar but more complicated explanation holds for other forms of multicollinearity. we can see collinearity with the help of collinear matrix (heatmap)

Regression guide

<https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/>

**SVM (Support Vector Machines)**

If the data are *linearly separable*, or in other words, can be separated completely into their groups by a dividing hyperplane, then SVM finds the equation of the hyperplane that best divides the groups.

we label the classes for each of the data points x*i* as being­ 1 or -1, i.e. *yi*∈{­-1,1}. Our hyperplane function has the equation wTx + b and is defined such that for all points that have a class *yi* = ­1. Here, x*i* are the input variables

whereas *yi*is the target variable.

and for points with a class *yi* = 1,

In our training data, we should have no points in between the hyperplanes wTx + b = ­-1 and wTx + b = 1, a region called the ‘margin’. The dividing plane is the function wTx + b = 0 and we classify new points by their sign: *y*ˆ*i* = *sign*(wTx + b).

Ideally, we want to select parameters for the hyperplane that maximize the size of the margin. Basically, we minimise W to achieve larger margin between two hyperplanes. we approach solving the problem by noting that minimising ||w|| is equivalent to minimising

So, Converting problem into a quadratic programming optimisation problem which we can easily solve. The Lagrange multipliers α transform our optimization problem into one of maximizing the output of

while satisfying the constraints that all 0≤α*i*and ∑α*iyi*= 0. To provide some context for interpreting this, think of the multipliers α as weights on data points. From the constraint ∑α*iyi*= 0 , the sum of the weights on the points categorised as *yi* = ­-1 should be equal to those categorised as *yi* = 1. As for w(α), the second term controls the summed weights in the first term from getting too large. The second term takes into account the categories of each pair of points (*yiyj* = 1 if they are in the same class, ­1 if they differ) and a measure of similarity (evoked by x*i*Tx*j*) i.e. kernel function

When we obtain the optimal Lagrange multipliers, it turns out that most of the weights α*i* are equal to zero. **The points/coefficients that have non­ zero weight are the only points that contribute to the calculation of w**, and all in fact fall on the margin, satisfying *yi*(wTx*i* + b) = 1. **These points are the *support vectors* for the model.**

If we have data that is mostly linearly separable, we can consider using soft­margin SVMs, relaxing the criteria that all points are correctly classified. If we have data that is separable in a nonlinear fashion, we can consider using *kernel functions* to be able to capture a nonlinear dividing curve between classes. Typically, we make considerations of both kernel function and value of soft­margin parameter to perform classification tasks.

In a soft­margin SVM, we do not require the data to be completely linearly separable and allow for some points to be classified incorrectly. We provide for each point a non­negative slack variable ξ*i*that illustrates to what degree each point is misclassified: *yi*(wTx*i* + b) ≥ 1- ξ*i*. If a point is classified correctly on its side of the margin, then ξ*i* = 0. If a point gets placed within the margin or in the wrong­ classed region, then ξ*i* takes on positive value proportional to the point’s distance from its desired marginal hyperplane.

Our optimization problem now has to balance the size of the errors we make: our goal is to minimise 1/2||w||^2 + *C*∑ξ*i*, where *C* is a regularization parameter that tells us the weight we want to put on misclassification errors. With smaller values of *C*, we punish errors less, thus increasing the size of the margin. Larger values of *C* result in narrower margins; the limit of *C* as it tends towards infinity is that any misclassification error is punished to an extent that we effectively have our original hard­ margin SVM.

For data that is separable, but not linearly, we can use a kernel function to capture a non-linear dividing curve. The kernel function should capture some aspect of similarity in our data; it also represents domain knowledge regarding the structure of the data.

*you                                                                                               For more detail please go through Udacity pdf*

Support Vector Classification

**kernel** : string, optional (default=’rbf’)

Specifies the kernel type to be used in the algorithm. It must be one of ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable. If none is given, ‘rbf’ will be used.

Nu-Support Vector Classification

Similar to SVC but uses a parameter to control the number of support vectors. Here, nu replaces C.

Support Vector Regression

*class*sklearn.svm.**SVR**(*kernel=’rbf’*, *degree=3*, *gamma=’auto’*, *coef0=0.0*, *tol=0.001*, *C=1.0*, *epsilon=0.1*, *shrinking=True*, *cache\_size=200*, *verbose=False*, *max\_iter=-1*)[[source]](https://github.com/scikit-learn/scikit-learn/blob/a24c8b46/sklearn/svm/classes.py#L741)[¶](http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html#sklearn.svm.SVR)

Here, C and epsilon are regularisation term which is discussed in above literature.

Nu Support Vector Regression

Similar to NuSVC, for regression, uses a parameter nu to control the number of support vectors. However, unlike NuSVC, where nu replaces C, here nu replaces the parameter epsilon of epsilon-SVR.

**Linear Regression**

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<http://www.ritchieng.com/machine-learning-evaluate-linear-regression-model/>

<https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/>

Regression kaggle

<https://www.kaggle.com/sahababu/comprehensive-data-exploration-with-python?scriptVersionId=2072252> in a light of above document, which explains about the assumptions of regression. Practicing one example of advance regression from kaggle will suffice.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Linear regression equation looks like this

Here, we have Y as our dependent variable (Sales), X’s are the independent variables and all thetas are the coefficients. Coefficients are basically the weights assigned to the features, based on their importance.

linear regression with only one feature, i.e., only one independent variable.

or Y = mx + c (Straight line equation)

Sum of squared residuals/errors:

where, h(x) is the value predicted by us,  h(x) =Θ1\*x +Θ0, y is the actual values and m is the number of rows in the training set.

Cost Function:

It is just similar to sum of squared errors, with just a factor of 1/2m is multiplied in order to ease mathematics.

So in order to improve our prediction, we need to minimize the cost function, same as minimising the squared error . For this purpose we use the gradient descent algorithm. (Later we will discuss the gradient descent algorithm)

**R-Square**: It determines how much of the total variation in Y (dependent variable) is explained by the variation in X (independent variable). Mathematically, it can be written as:

The value of R-square is always between 0 and 1, where 0 means that the model does not explain any variability in the target variable (Y) and 1 meaning it explains full variability in the target variable.

In this case, R² is 32%, meaning, only 32% of variance in target variable(sales) is explained by independent variable(establishment and the MRP) In other words, if you know year of establishment and the MRP, you’ll have 32% information to make an accurate prediction about its sales.

**Adjusted R-Square**:

The only drawback of R2is that if new predictors (X) are added to our model, R2 only increases or remains constant but it never decreases. We can not judge that by increasing complexity of our model, are we making it more accurate?

That is why, we use “Adjusted R-Square”.

The Adjusted R-Square is the modified form of R-Square that has been adjusted for the number of predictors in the model. It incorporates model’s degree of freedom. The adjusted R-Square only increases if the new term improves the model accuracy.

where, R2 = Sample R square , p = Number of predictors, N = total sample size

**concordance correlation coefficient:**

**Assumptions of Linear Regression**

1) Variables are Normally Distributed

Regression assumes that variables have normal distributions. Non-normally distributed variables (highly skewed or kurtotic variables, or variables with substantial outliers) can distort relationships and significance tests. There are several pieces of information that are useful to the researcher in testing this assumption: visual inspection of data plots, skew, kurtosis, and P-P plots give researchers information about normality, and Kolmogorov-Smirnov tests provide inferential statistics on normality. Outliers can be identified either through visual inspection of histograms or frequency distributions, or by converting data to z-scores.

Removal of univariate and bivariate outliers can reduce the probability of Type I and Type II errors, and improve accuracy of estimates. However, it is not always desirable to remove outliers. In this case transformations (e.g., square root, log, or inverse), can improve normality, but complicate the interpretation of the results, and should be used deliberately and in an informed manner.

2) Assumption of a Linear relationship between the Independent and Dependent variables

**How to check:** Look for residual vs fitted value plots (explained below).

Standard multiple regression can only accurately estimate the relationship between dependent and independent variables if the relationships are linear in nature.

            Example of curvilinear and linear relationships with standardized residuals by standardized predicted

plt.scatter(pred\_cv, (pred\_cv - y\_cv), c='b')

3) Autocorrelation

The presence of correlation in error terms drastically reduces model’s accuracy. This usually occurs in time series models where the next instant is dependent on previous instant. If the error terms are correlated, the estimated standard errors tend to underestimate the true standard error.

If this happens, it causes confidence intervals and prediction intervals to be narrower. Narrower confidence interval means that a 95% confidence interval would have lesser probability than 0.95 that it would contain the actual value of coefficients. Let’s understand narrow prediction intervals with an example:

For example, the least square coefficient of X¹ is 15.02 and its standard error is 2.08 (without autocorrelation). But in presence of autocorrelation, the standard error reduces to 1.20. As a result, the prediction interval narrows down to (13.82, 16.22) from (12.94, 17.10).

Also, lower standard errors would cause the associated p-values to be lower than actual. This will make us incorrectly conclude a parameter to be statistically significant.

**How to check:** Look for Durbin – Watson (DW) statistic. It must lie between 0 and 4. If DW = 2, implies no autocorrelation, 0 < DW < 2 implies positive autocorrelation while 2 < DW < 4 indicates negative autocorrelation. Also, you can see residual vs time plot and look for the seasonal or correlated pattern in residual values.

                                                    Explanation DW Test:  <http://www.statisticshowto.com/durbin-watson-test-coefficient/>  and in sklearn you can find this value in OLS, In OLS (ordinary least square) summary one of the parameter is Durbin-Watson. The screenshot is pasted below for reference.

4) Multicollinearity

This phenomenon exists when the independent variables are found to be moderately or highly correlated. In a model with correlated variables, it becomes a tough task to figure out the true relationship of a predictors with response variable. In other words, it becomes difficult to find out which variable is actually contributing to predict the response variable.

                                                                                Another point, with presence of correlated predictors, the standard errors tend to increase. And, with large standard errors, the confidence interval becomes wider leading to less precise estimates of slope parameters.

Also, when predictors are correlated, the estimated regression coefficient of a correlated variable depends on which other predictors are available in the model. If this happens, you’ll end up with an incorrect conclusion that a variable strongly / weakly affects target variable. Since, even if you drop one correlated variable from the model, its estimated regression coefficients would change. That’s not good!

**How to check:** You can use scatter plot to visualize correlation effect among variables. Also, you can also use VIF factor. VIF value <= 4 suggests no multicollinearity whereas a value of >= 10 implies serious multicollinearity. Above all, a correlation table should also solve the purpose.

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

5) Assumptions of Homoscedasticity

Homoscedasticity means that the variance of errors is the same across all levels of the IV. When the variance of errors differs at different values of the IV, heteroscedasticity is indicated. Slight heteroscedasticity has little effect on significance tests; however, when heteroscedasticity is marked it can lead to serious distortion of findings and seriously weaken the analysis thus increasing the possibility of a Type I error.

When the null hypothesis is true and you reject it, you make a type I error.

When the null hypothesis is false and you fail to reject it, you make a type II error.

                                                            Examples of homoscedasticity and heteroscedasticity

Ideally, residuals are randomly scattered around 0 (the horizontal line) providing a relatively even distribution. Heteroscedasticity is indicated when the residuals are not evenly scattered around the line. There are many forms heteroscedasticity can take, such as a bow-tie or fan shape. In cases where skew is present in the independent variables, transformation(log,root,square) of variables can reduce the heteroscedasticity.

We can easily check Heteroscedasticity by looking at residual vs fitted values plot.

plt.scatter(pred\_cv, (pred\_cv - y\_cv), c='b')

*For more information visit:*[*http://pareonline.net/getvn.asp?n=2&v=8*](http://pareonline.net/getvn.asp?n=2&v=8)

<https://www.analyticsvidhya.com/blog/2016/07/deeper-regression-analysis-assumptions-plots-solutions/>

**Lasso (L1), Ridge (L2) & Elastic Net Regression**

Basically there are two methods to overcome overfitting,

* Reduce the model complexity
* Regularization

Here we would be discussing about Regularization in detail and how to use it to make your model more generalized. In regularization, what we do is normally we keep the same number of features, but reduce the magnitude of the coefficients/weight

 to overcome overfitting.

Let us take a look at the coefficients of feature in our regression model.

checking the magnitude of coefficients

predictors = x\_train.columns

coef = Series(lreg.coef\_,predictors).sort\_values()

coef.plot(kind='bar', title='Modal Coefficients')

We can see that coefficients of Outlet\_Identifier\_OUT027 and Outlet\_Type\_Supermarket\_Type3(last 2) is much higher as compared to rest of the coefficients. Therefore the total sales of an item would be more driven by these two features.

*How can we reduce the magnitude of coefficients in our model? For this purpose, we have different types of regression techniques which uses regularization to overcome this problem. So let us discuss them.*

You can see that, as we increase the value of alpha, the magnitude of the coefficients decreases, where the values reaches to zero but not absolute zero.

While practicing machine learning, you may have come upon a choice of the mysterious L1 vs L2. Usually the two decisions are : 1) L1-norm vs L2-norm loss function; and 2) L1-regularization vs L2-regularization.

## *As An Error Function*

L1-norm loss function is also known as least absolute errors (LAE)/Sum of absolute errors. It is basically minimizing the sum of the absolute differences (S) between the target value (Yi) and the estimated values (f(xi)):

L2-norm loss function is also known as least squares error (LSE)/Sum of squared error. It is basically minimizing the sum of the square of the differences (S) between the target value (Yi) and the estimated values (f(xi):

***As Regularization***

Regularization is a very important technique in machine learning to prevent overfitting. Mathematically speaking, it adds a *regularization* *term* in order to prevent the coefficients to fit so perfectly to overfit. The difference between the L1 and L2 is just that L2 is the sum of the square of the weights, while L1 is just the sum of the weights. As follows:

A regression model that uses L1 regularization technique is called Lasso Regression and model which uses L2 is called Ridge Regression.

The key difference between these two is the penalty term.

Ridge regression adds “squared magnitude” of coefficient as penalty term to the loss function.

our idea was to basically minimize the cost function, such that values predicted are much closer to the desired result.

Now take a look back again at the cost function for ridge regression:

Lasso Regression (Least Absolute Shrinkage and Selection Operator) adds “absolute value of magnitude” of coefficient as penalty term to the loss function. In other words, lasso regression is quiet similar to that of ridge only difference being instead of adding squares of theta, we will add absolute value of Θ.

Here too, λ is the hypermeter, whose value is equal to the alpha in the Lasso function.

Lasso selects only some feature while reduces the coefficients of others to zero. This property is known as feature selection and which is absent in case of ridge.

Elastic net is basically a combination of both L1 and L2 regularization. So if you know elastic net, you can implement both Ridge and Lasso by tuning the parameters. So it uses both L1 and L2 penality term, therefore its equation look like as follows:

So, if you look at the code above, we need to define alpha and l1\_ratio while defining the model. Alpha and l1\_ratio are the parameters which you can set accordingly if you wish to control the L1 and L2 penalty separately. Actually, we have

*alpha* = a + b           and     l1\_ratio =  a / (a+b)

where, a and b weights assigned to L1 and L2 term respectively. So when we change the values of alpha and l1\_ratio, a and b are set accordingly such that they control trade off between L1 and L2 as:                                                            a \* (L1 term) + b\* (L2 term)

LassoCV, RidgeCV, ElasticNetCV : Lasso linear model with iterative fitting along a regularization path,The best model is selected by cross-validation.

**Stochastic Gradient Descent f*or Linear Regression***

*SGD is indeed a technique that is used to find the minima of a function.*

Gradient descent is an optimization algorithm that finds the optimal weights (a,b) that reduces prediction error in this case SSE.

So a,b are nothing but θ1 and θ2, those are weight which will minimise the magnitude of coefficient to overcome overfitting

Lets now go step by step to understand the Gradient Descent algorithm:

**Step 1:**Initialize the weights (a & b) with random values and calculate Error (SSE)

             To fit a line Ypred = a + b X, start off with random values of a and b and calculate prediction error (SSE)

**Step 2:**Calculate the error gradient w.r.t the weights

∂SSE/∂a = – (Y-YP)

∂SSE/∂b = – (Y-YP)X

Here, SSE=½ (Y-YP)2 = ½(Y-(a+bX))2

∂SSE/∂a and ∂SSE/∂b are the **gradients** and they give the direction of the movement of a,b w.r.t to SSE.

**Step 3:**Adjust the weights with the gradients to reach the optimal values where SSE is minimized

We need to update the random values of a,b so that we move in the direction of optimal a, b.

Update rules:

* a – ∂SSE/∂a
* b – ∂SSE/∂b

So, update rules:

1. New a = a – r \* **∂SSE/∂a =**0.45-0.01\*3.300 = 0.42
2. New b = b – r \* **∂SSE/∂b=**0.75-0.01\*1.545 = 0.73

here, r is the learning rate = 0.01, which is the pace of adjustment to the weights.

**Step 4:**Use new a and b for prediction and to calculate new Total SSE

You can see with the new prediction, the total SSE has gone down (0.677 to 0.553). That means prediction accuracy has improved.

**Step 5:**Repeat step 3 and 4 till the time further adjustments to a, b doesn’t significantly reduces the error. At that time, we have arrived at the optimal a,b with the highest prediction accuracy.

This is the Gradient Descent Algorithm. This optimization algorithm and its variants form the core of many machine learning algorithms like Neural Networks and even Deep Learning.

*Reference:*[*https://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html*](https://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html)

Stochastic Gradient Descent is sensitive to feature scaling, so it is highly recommended to scale your data. Note that the same scaling must be applied to the test vector to obtain meaningful results. This can be easily done using **StandardScaler**:

*The possible values are ‘squared\_loss’, ‘huber’, ‘epsilon\_insensitive’, or ‘squared\_epsilon\_insensitive’*

*The ‘squared\_loss’ refers to the ordinary least squares fit. ‘huber’ modifies ‘squared\_loss’ to focus less on getting outliers correct by switching from squared to linear loss past a distance of epsilon. ‘epsilon\_insensitive’ ignores errors less than epsilon and is linear past that; this is the loss function used in SVR. ‘squared\_epsilon\_insensitive’ is the same but becomes squared loss past a tolerance of epsilon.*

*The goal of all supervised machine learning algorithms is to best estimate a target function (f) that maps input data (X) onto output variables (Y). This describes all classification and regression problems.*

*Some machine learning algorithms have coefficients that characterize the algorithms estimate for the target function (f). Different algorithms have different representations and different coefficients, but many of them require a process of optimization to find the set of coefficients that result in the best estimate of the target function.*

*Common examples of algorithms with coefficients that can be optimized using gradient descent are Linear Regression and Logistic Regression.*

The loss function to be used. Defaults to ‘hinge’, which gives a linear SVM.

The possible options are ‘hinge’, ‘log’, ‘modified\_huber’, ‘squared\_hinge’, ‘perceptron’.

The ‘log’ loss gives logistic regression, a probabilistic classifier. ‘modified\_huber’ is another smooth loss that brings tolerance to outliers as well as probability estimates. ‘squared\_hinge’ is like hinge but is quadratically penalized. ‘perceptron’ is the linear loss used by the perceptron algorithm.

Here, Cost function is nothing but sigmoid function as in Linear regression it was SSE. main objective is to minimize the cost function.

**K-Nearest Neighbour (KNN)**

<https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/>

From the above link write more point for knn, so it will give the sense of completeness in knn and evaluation parameter is same as what we used for regression and classification.

Knn is a supervised algorithm used for regression as well as for classification purposes. It use distance metrics to find the nearest neighbour, It uses several distance metrics such as Euclidean, chebysev, Manhattan, minkowaski (Default), Hamming distance and so on. knn uses different type of algorithm according to the dataset when auto option is chosen in the sklearn. It selects from the given set of algorithms like Brute, Kdtree, Balltree. Each have there own pros and cons.

**Clustering**

CLUSTER analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group

(called cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics

The goal of partitioning clustering algorithms is to split the data set into clusters of objects, such that:

* the objects in the same cluster are similar as much as possible,
* and the objects in different clusters are highly distinct

*A.    Types of Clustering*

*Cluster:* It is said to be *“Collection of data objects”.*

Where the two types of similarities of clustering’s are:

* *Intraclass similarity* - Objects are similar to objects in same cluster
* *Interclass dissimilarity* - Objects are dissimilar to objects in other clusters

## Types of clustering algorithms

* **Connectivity models:** As the name suggests, these models are based on the notion that the data points closer in data space exhibit more similarity to each other than the data points lying farther away. These models can follow two approaches. In the first approach, they start with classifying all data points into separate clusters & then aggregating them as the distance decreases. In the second approach, all data points are classified as a single cluster and then partitioned as the distance increases. Also, the choice of distance function is subjective. These models are very easy to interpret but lacks scalability for handling big datasets. Examples of these models are ***hierarchical clustering*** algorithm and its variants.
* **Centroid models:** These are iterative clustering algorithms in which the notion of similarity is derived by the closeness of a data point to the centroid of the clusters. ***K-Means clustering*** *algorithm* is a popular algorithm that falls into this category. In these models, the no. of clusters required at the end have to be mentioned beforehand, which makes it important to have prior knowledge of the dataset. These models run iteratively to find the local optima.
* **Distribution models:** These clustering models are based on the notion of how probable is it that all data points in the cluster belong to the same distribution (For example: Normal, Gaussian). These models often suffer from overfitting. A popular example of these models is ***Expectation-maximization*** algorithm which uses multivariate normal distributions.
* **Density Models:**These models search the data space for areas of varied density of data points in the data space. It isolates various different density regions and assign the data points within these regions in the same cluster. Popular examples of *density models are* ***DBSCAN*** *and OPTICS.*

*B.   Methods of Clustering*

k means clustering, EM Clustering, Affinity Propagation, DBSCAN (Density-Based Spatial Clustering of Applications with Noise), Hierarchal Agglomerative Clustering

Above are the types of clustering which we will understand

**K Means Clustering**

df = nba, nba df consist only two cols ppg and atr, col name = ppg, atr, ppg: point per game, atr: assist turn over, plt.scatter(nba[‘ppg’], nba[‘atr’])

We named the 5 rows (centroid values) as clusters (0, 1, 2,3 and 4)

Suppose we want number of cluster = 5, so in centroid base clustering we will randomly choose 5 centroids values (nba[‘ppg’], nba[‘atr’]).

We will calculate euclidean distance of each centroid(i.e those 5 rows ) with rest of the values in the column one by one, whichever centroids(rest of rows) is near to the clusters(0,1,2,3,4). According we will group those centroid values in the clusters.

Then we will calculate cluster wise mean of nba[‘ppg’] and nba[‘atr’]. And we will obtain 5 values.

Previously we did randomly in step 1 , but this time we will repeat step (2 &3) with Intial value what we obtained in step 4

# How can we choose a "good" K for K-means clustering?

A quick (and rough) method is to take the square root of the number of data points(i.e. no. of observations or no. of rows) divided by two, and set that as the number of clusters. The elbow method and kernel method  work more precisely, but the number of clusters can also *depend on your problem*.

In general, there is no method for determining the exact value of *K*, but an accurate estimate can be obtained using the following techniques.

One of the metrics that is commonly used to compare results across different values of *K* is the mean distance between data points and their cluster centroid. Since increasing the number of clusters will always reduce the distance to data points, increasing *K* will *always* decrease this metric, to the extreme of reaching zero when *K* is the same as the number of data points. Thus, this metric cannot be used as the sole target. Instead, mean distance to the centroid as a function of *K* is plotted and the "elbow point," where the rate of decrease sharply shifts, can be used to roughly determine *K* (see plot below).

A number of other techniques exist for validating *K*, including cross-validation, information criteria, the information theoretic jump method, the silhouette method, and the G-means algorithm.

In addition, monitoring or visualizing the distribution of data points across groups is a useful and easy way to gain insight into how the algorithm is splitting the data for each *K*.

*C. Methods of cluster validation*

1. **Internal cluster validation**, which uses the internal information of the clustering process to evaluate the goodness of a clustering structure without reference to external information. It can be also used for estimating the number of clusters and the appropriate clustering algorithm without any external data.
2. **External cluster validation**, which consists in comparing the results of a cluster analysis to an externally known result, such as externally provided class labels. It measures the extent to which cluster labels match externally supplied class labels. Since we know the “true” cluster number in advance, this approach is mainly used for selecting the right clustering algorithm for a specific data set.
3. **Relative cluster validation**, which evaluates the clustering structure by varying different parameter values for the same algorithm (e.g.,: varying the number of clusters k). It’s generally used for determining the optimal number of clusters.

Internal validation measures reflect often the **compactness**, the **connectedness** and the **separation** of the cluster partitions.

1. **Compactness** or cluster cohesion: Measures how close are the objects within the same cluster. A lower **within-cluster variation** is an indicator of a good compactness (i.e., a good clustering). The different indices for evaluating the compactness of clusters are base on distance measures such as the cluster-wise within average/median distances between observations.
2. **Separation**: Measures how well-separated a cluster is from other clusters. The indices used as separation measures include:

* distances between cluster centers
* the pairwise minimum distances between objects in different clusters
* **Connectivity**: corresponds to what extent items are placed in the same cluster as their nearest neighbors in the data space. The connectivity has a value between 0 and infinity and should be minimized.

**Silhouette coefficient:**

The silhouette analysis measures how well an observation is clustered and it estimates the **average distance between clusters**. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters.

<https://www.coursera.org/learn/cluster-analysis/lecture/vPsaH/6-8-relative-measures>

This measure has a range of [-1, 1]

Silhouette coefficients (as these values are referred to as) near +1 indicate that the sample is far away from the neighbouring clusters i.e model is better. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighbouring clusters i.e. model is poor and negative values indicate that those samples might have been assigned to the wrong cluster.

**algorithm** : “auto”, “full” or “elkan”, default=”auto”

K-means algorithm to use. The classical EM-style algorithm is “full”. The “elkan” variation is more efficient by using the triangle inequality, but currently doesn’t support sparse data. “auto” chooses “elkan” for dense data and “full” for sparse data.

## EM Clustering

Expectation-maximization (EM) is a method to find the maximum likelihood estimator of a parameter θ of a probability distribution.

So, with K-Means clustering each point is assigned to just a single cluster, and a cluster is described only by its centroid. This is not too flexible, as we may have problems with clusters that are overlapping, or ones that are not of circular shape. With [EM Clustering](http://docs.rapidminer.com/studio/operators/modeling/segmentation/expectation_maximization_clustering.html), we can now go a step further and describe each cluster by its centroid (mean), covariance (so that we can have elliptical clusters), and weight (the size of the cluster). The probability that a point belongs to a cluster is now given by a multivariate Gaussian probability distribution (multivariate - depending on multiple variables). That also means that we can calculate the probability of a point being under a Gaussian ‘bell’, i.e. the probability of a point belonging to a cluster.

We now start the EM procedure by calculating, for each point, the probabilities of it belonging to each of the current clusters (which, again, may be randomly created at the beginning). This is the E-step. If one cluster is a really good candidate for a point, it will have a probability close to one. However, two or more clusters can be acceptable candidates, so the point has a distribution of probabilities over clusters. This property of the algorithm, of points not belonging restricted to one cluster is called “soft clustering”.

Affinity Propagation

Rather than requiring that the number of clusters be respecified as in k means clustering, affinity propagation takes as input a real number s(k,k) say  -50 for each data point k so that data points with larger values of s(k,k) are more likely to be chosen as exemplars. These values are referred to as “preferences.” The number of identified exemplars (number of clusters) is influenced by the values of the input preferences, but also emerges from the message-passing procedure. If a priori, all data points are equally suitable as exemplars, the preferences should be set to a common value—this value can be varied to produce different numbers of clusters. The shared value could be the median of the input similarities (resulting in a moderate number of clusters) or their minimum (resulting in a small number of clusters).

There are two kinds of message exchanged between data points, and each takes into ac- count a different kind of competition. Mes- sages can be combined at any stage to decide which points are exemplars and, for every other point, which exemplar it belongs to. The“responsibility” r(i,k), sent from data point i to candidate exemplar point k, reflects the ac- cumulated evidence for how well-suited pointk is to serve as the exemplar for point i, taking into account other potential exemplars for point i. The “availability” a(i,k), sent from candidate exemplar point k to point i, reflects the accumulated evidence for how appropriate it would be for point i to choose point k as its exemplar, taking into account the support from other points that point k should be an exemplar. r(i,k) and a(i,k) can be viewed as log-probability ratios. To begin with, the availabilities are initialized to zero:a(i,k) = 0. Then, the responsibilities are com- puted using the rule

In the first iteration, because the availabilities are zero, r(i,k) is set to the input similarity between point i and point k as its exemplar, minus the largest of the similarities between point i and other candidate exemplars. [*http://www.psi.toronto.edu/affinitypropagation/FreyDueckScience07.pdf*](http://www.psi.toronto.edu/affinitypropagation/FreyDueckScience07.pdf)

DBSCAN

Hierarchal Agglomerative clustering

Hierarchical clustering

Hierarchical clustering, as the name suggests is an algorithm that builds hierarchy of clusters. This algorithm starts with all the data points assigned to a cluster of their own. Then two nearest clusters are merged into the same cluster. In the end, this algorithm terminates when there is only a single cluster left.

The results of hierarchical clustering can be shown using dendrogram. The dendrogram can be interpreted as:

At the bottom, we start with 25 data points, each assigned to separate clusters. Two closest clusters are then merged till we have just one cluster at the top. The height in the dendrogram at which two clusters are merged represents the distance between two clusters in the data space.

The decision of the no. of clusters that can best depict different groups can be chosen by observing the dendrogram. The best choice of the no. of clusters is the no. of vertical lines in the dendrogram cut by a horizontal line that can transverse the maximum distance vertically without intersecting a cluster. In the above example, the best choice of no. of clusters will be 4 as the red horizontal line in the dendrogram below covers maximum vertical distance AB.

Two important things that you should know about hierarchical clustering are:

* This algorithm has been implemented above using bottom up approach. It is also possible to follow top-down approach starting with all data points assigned in the same cluster and recursively performing splits till each data point is assigned a separate cluster.
* The decision of merging two clusters is taken on the basis of closeness of these clusters. There are multiple metrics for deciding the closeness of two clusters :
* Euclidean distance: ||a-b||2 = √(Σ(ai-bi))
* Squared Euclidean distance: ||a-b||22 = Σ((ai-bi)2)
* Manhattan distance: ||a-b||1 = Σ|ai-bi|
* Maximum distance:||a-b||INFINITY = maxi|ai-bi|
* Mahalanobis distance: √((a-b)T S-1 (-b))   {where, s : covariance matrix}

Difference b/w Hierarchical and K means clustering

Hierarchical clustering can’t handle big data well but K Means clustering can. This is because the time complexity of K Means is linear i.e. O(n) while that of hierarchical clustering is quadratic i.e. O(n2).

<http://www.psi.toronto.edu/affinitypropagation/faq.html>

Evaluation parameter

Estimated number of clusters: 6

Homogeneity: 1.000

Completeness: 0.801

V-measure: 0.890

Adjusted Rand Index(ARI): 0.819   range b/w 0-1

Normalized mutual information(NMI): 0.75   range b/w 0-1

Adjusted Mutual Information: 0.799     range b/w 0-1

Silhouette Coefficient: 0.574   (However, doesn't work well in practice. silhouette score computes the compactness of a cluster. while compact clusters are good, compactness doesn't allow complex shape )

Data Cleaning Kaggle Notebook

1. <https://www.kaggle.com/sahababu/data-cleaning-challenge-handling-missing-v-6d857e/edit>

2. <https://www.kaggle.com/dansbecker/handling-missing-values>

**Ways to Handle imbalance Data:**

<https://blog.dominodatalab.com/imbalanced-datasets/>

<https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets>

Bootstrap Sampling

SMOTE

Tomek links

Stratified CV

Precision-Recall (PR) curve is recommended over ROC for imbalanced data.

Mathews correlation coefficient (MCC)

Cohen’s kappa

Class weighted/Cost sensitive learning  <https://medium.com/coinmonks/practical-tips-for-class-imbalance-in-binary-classification-6ee29bcdb8a7>

Better go with GBM(Gradient Boosted machines) for imbalanced data.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Linear Programming (Linear Optimization) - <https://www.analyticsvidhya.com/blog/2017/02/lintroductory-guide-on-linear-programming-explained-in-simple-english/>

# Importance of Distance Metrics in Machine Learning Modelling

<https://towardsdatascience.com/importance-of-distance-metrics-in-machine-learning-modelling-e51395ffe60d>

# Cosine Similarity – Understanding the math and how it works (with python codes)

<https://www.machinelearningplus.com/nlp/cosine-similarity/>

## Understanding Gradient Boosting Machines: Lightgbm and Xgboost

Boosting refers to a group of algorithms which transforms weak learner to strong learners.

Boosting algorithms are classified into:

* Gradient Boosting
* XGBoost
* AdaBoost etc.

## Elements in Gradient Boosting Algorithm

Basically, Gradient boosting Algorithm involves three elements:

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

***My only goal is to gradient boost over myself of yesterday. And to repeat this every day with an unconquerable spirit.***

With each passing day, we aim to improve ourselves by focusing on the mistakes of yesterday.

And you know what? — GBMs do that too!

GBMs do it by creating an ensemble of predictors. Each one of those predictors is **sequentially** built by focusing on the mistakes(misclassified data) of the previous one.

### What’s an ensemble?

It is simply a group of items viewed as a whole rather than individually.

A GBM basically creates lots of individual predictors and each of them tries to predict the true label. Then, it gives its final prediction by averaging all those individual predictions (note however that it is not a simple average but a weighted average).

* That is in fact what an ensemble method is. And random forests and gradient boosting machines are just 2 types of ensemble methods.

### One important difference between the two is that the predictors used in Random forest are independent of each other whereas the ones used in gradient boosting machines are built sequentially where each one tries to improve upon the mistakes made by its predecessor.

***Base Learner***

***Q:*Okay, so how does the algorithm decide the number of predictors to put in the ensemble?**

* It does not. We do. And that brings us to our first important parameter — n\_estimators : We pass the number of predictors that we want the GBM to build inside the n\_estimators parameter. The default number is 100.

So, let’s talk about these individual predictors now.

In theory, these predictors can be any regressor or classifier but in practice, decision trees give the best results.

The sklearn API for LightGBM provides a parameter- boosting\_typeand the [API](https://www.zeolearn.com/magazine/how-to-manage-api-secrets-tokens?utm_source=blog&utm_medium=internallink&utm_campaign=blogpromo)for XGBoost has parameter- booster to change this predictor algorithm. You can choose from —  gbdt, dart, goss, rf (LightGBM) or gbtree, gblinear or dart (XGBoost). [Notehowever that a decision tree, almost always, outperforms the other options by a fairly large margin. The good thing is that it is the default setting for this parameter; so you don’t have to worry about it.]

### Creating weak predictors

We also want these predictors to be weak**.**A **weak predictor** is simply a prediction model that performs better than random guessing.

**Q: Wait a second.. that seems backwards. Don’t we want to have strong predictors that can make good guesses?**

* Nope. We want the individual predictors to be weak so that the overall ensemble becomes strong. This is because every predictor is going to focus on the observations that the one preceding it got wrong. When we use a weak predictor, these mislabelled observations tend to have some learnable information which the next predictor can learn. Whereas if the predictor were already strong, it would be likely that the mislabelled observations are just noise or nuances of that sample data. In such a case the model will just be overfitting to the training data.

Also note that if the predictors are just too weak, it might not even be possible to build a strong ensemble out of them.

Now back to creating a weak predictor.. this seems like a good area to *hyperparameterise*.

These are the parameters that we need to tune to make the right predictors (which are decision trees):

* **max\_depth**(both XGBoost and LightGBM)**:**This provides the maximum depth that each decision tree is allowed to have. A smaller value signifies a weaker predictor.
* **min\_split\_gain** (LightGBM),gamma (XGBoost): Minimum loss reduction required to make a further partition on a leaf node of the tree. A lower value will result in deeper trees.
* **num\_leaves** (LightGBM): Maximum tree leaves for base learners. A higher value results in deeper trees.
* **min\_child\_samples** (LightGBM): Minimum number of data needed in a child (leaf). According to the LightGBM docs, this is a very important parameter to prevent overfitting.

**Note:**These are also the parameters that you can tune to control overfitting.

The subtree marked in red has a leaf node with 1 data in it. So, that subtree can’t be generated as 1 < `min\_child\_samples` for the above case

### Subsampling

Even after we do all this, it might just happen that some trees in the ensemble are highly correlated.

**Q: Excuse me, what do you mean by highly correlated trees?**

* I mean decision trees that are similar in structure because of similar splits based on same features. This means that the ensemble as a whole is going to store less amount of information than what it could have stored if the trees were different. So we want our trees to be as little correlated as possible.

To combat this problem, we subsample the data rows and columns before each iteration and train the tree on this subsample. These are the relevant parameters to look out for:

* **subsample** (both XGBoost and LightGBM): This specifies the fraction of rows to consider at each subsampling stage. By default, it is set to 1, which means no subsampling.
* **colsample\_bytree** (both XGBoost and LightGBM): This specifies the fraction of columns to consider at each subsampling stage. By default, it is set to 1, which means no subsampling.
* **subsample\_freq** (LightGBM): This specifies that bagging should be performed after every k iterations. By default, it is set to 0. So make sure that you set it to some non-zero value if you want to enable subsampling.

That is it. Now you have a good overview of the whole story of how a GBM works. There are 2 more important parameters though which I couldn’t fit into the story. So, here they are —

* **learning\_rate** (both XGBoost and LightGBM): It is also called shrinkage. The effect of using it is that learning is slowed down, in turn requiring more trees to be added to the ensemble. This gives the model a regularisation effect.
* **class\_weight** (LightGBM): This parameter is extremely important for multi-class classification tasks when we have imbalanced classes. I recently participated in a Kaggle competition where simply setting this parameter’s value to balanced caused my solution to jump from the top 50% of the leaderboard to the top 10%.

You can check out the sklearn API for LightGBM [here](https://lightgbm.readthedocs.io/en/latest/Python-API.html#scikit-learn-api) and that for XGBoost [here](https://xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.XGBRegressor).

## Finding the best set of hyperparameters

You can use sklearn’s RandomizedSearchCV in order to find a good set of hyperparameters. It will randomly search through a subset of all possible combinations of the hyperparameters and return the best possible set of hyperparameters(or at least something close to the best).

But if you wish to go even further, you could look around the hyperparameter set that it returns using GridSearchCV. Grid search will train the model using every possible hyperparameter combination and return the best set. Note that since it tries every possible combination, it can be expensive to run

*Reference :*[*https://www.zeolearn.com/magazine/a-guide-to-understanding-gradient-boosting-machines-lightgbm-and-xgboost*](https://www.zeolearn.com/magazine/a-guide-to-understanding-gradient-boosting-machines-lightgbm-and-xgboost)