Assignment 2

Question 1:

Data pre-processing: Initially I started with exploring the data set using excel. At the time of exploration, I came across some attributes who had "?" as their values for some of the rows from the entire dataset. The columns which were having such values were **"workclass", "occupation"** and **"native-country"**. Then I decided to replace all the "?" with initially null values i.e. **""** using excel find and replace all functionality. As I have decided to do this assignment in python, I started with loading the dataset into a DataFrame and printing top 25 rows. So, in python all the null values are displayed as **"NaN"**. Further I replaced all the null values with the most frequently occurring values of that particular column i.e. mode of the column. So, null values for my **"workclass"** column were replaced with **"Private"**, for **"occupation"** they were replaced with **"Prof-specialty"** and for **"native-country"** they were replaced by **"United-States"**. As most of the data in my data is categorical and contains large number of categories per attribute. Machine learning algorithms like neural network requires the input and output to be in numbers. So, I decided to apply encoding on my dataset using various different encoding techniques. The best way to deal with ordinal features is to apply label encoding but in my dataset all the categories are of nominal type. When the data is of nominal type, we should use one hot encoding. But, the major drawback of applying one hot encoding is that it increases the number of columns in your dataset which will result in the curse of dimensionality. One hot encoding is only suitable when you have very few categorical attributes with very few categories per attribute. In my case, my dataset contains multiple numbers of attributes and each attribute has more than 7-8 categories so using one hot encoding on my dataset will result in a lot of columns. I checked the numbers of columns after applying encoding on my dataset and I got a value of 105 i.e. 105 columns and initially the number of columns in my dataset was 15. If I used label encoding on my dataset then it wont increase the number of columns in my dataset but it is more likely that our algorithm might misinterpret and would assume that my categories have actual ordering and would create the ordering accordingly. So, keeping all these things in mind I decided to apply frequency encoding to my categorical attributes. In frequency encoding I simply replaced the categorical values with the number of times they occur in that particular attribute i.e. their frequencies. Doing this have some disadvantage i.e. if you have couple of categories with same frequencies then you might turn into losing some valuable information by replacing these categories with same frequencies. To avoid that, I first checked whether the frequencies of each and every category in my dataset is different than others. I got to know that only my **"native-country"** attribute had a lot categories with same frequencies so I decided to encode that attribute using **one hot encoding** and the remaining categorical attributes like **"workclass", "education", "marital-status", "relationship", "occupation", "race", "sex"** were encoded using **frequency encoding** because none of the categories have same frequency. I then encoded my target attribute using label encoding by replacing **"<=50"** with **"0"** and **">50"** with **"1"**. Now that I am done with the encoding and dealing with the missing values, I am applying **MinMaxScaler** to normalize all the attribute values as some of the attributes have values that vary higly.

Ans a: So before choosing the evaluation measure I first checked the number of examples that belongs to each class in my dataset. There are total of **2258** examples that belongs to **"<=50"** class and **742** examples that belongs to **">50"** class. So according to this, I can say that my data is imbalanced and if I choose accuracy as an evaluation metrics then it will be biased towards the majority class for most of classifiers. In this case, I am going to use combination of **precision** and **recall** i.e. the **F1 Score**. Along with F1 Score I will be using another metrics like **Cohen's Kappa** and **Precision-Recall curve AUC** which are well suited for imbalanced data classification. To use best parameters to fit the data I am using **GridSearchCV** method in python for hyperparameter tunning. I can also use RandomizedSearchCV, but instead of using the entire search space i.e. all possible combinations of the specified parameters like GridSearchCV, it creates random search space i.e. it picks random combinations of specified parameters and then performs a randomized search. This takes less time than GridSearchCV but it will always give different best parameters. In my case, the dataset contains 3000 total examples so using GridSearchCV is acceptable as it is not taking much time for computing the best parameters. In the GridSearchCV I will be using "F1 measure" as the scoring parameter to evaluate the performance of each and every combinations of the specified parameters to get the best set of parameters.

- 1-NN Classifier: The parameters that I am specifying for GridSearchCV are:

{'n_neighbors': [1], 'metric': ['euclidean','manhattan','minkowski'], 'weights': ['uniform','distance']}
The best set of parameters that I got after performing the GridSearchCV with 12-fold cross validation is:
Best parameters for KNN= **{'metric': 'manhattan', 'n_neighbors': 1, 'weights': 'uniform'}**

Using these parameters I trained my model and then tested the model on unseen test data and got the following results:

Accuracy	79 %
F1 Score	0.5313653136531366
Precision-Recall curve AUC	0.5811186841148672
Cohen’s Kappa	0.3948538754764931
ROC AUC	0.7023144907957486

- Decision Tree Classifier: The parameters than I am specifying for GridSearchCV are:

{'criterion': ['gini','entropy'], 'max_depth': list(range(1,20)), 'max_leaf_nodes': list(range(1,10))}
The best set of parameters that I got after performing the GridSearchCV with 12-fold cross validation is:
Best parameters for DecisionTree= {'criterion': 'gini', 'max_depth': 4, 'max_leaf_nodes': 9}
Using these parameters I trained my model and then tested the model on unseen test data and got the following results:

Accuracy	86 %
F1 Score	0.6525423728813559
Precision-Recall curve AUC	0.705559796437659
Cohen’s Kappa	0.5687615040757297
ROC AUC	0.7640423835023358

- Neural Network Classifier: The parameters than I am specifying for GridSearchCV are:

{'alpha': [0.01, 0.5, 1, 5, 7], 'learning_rate_init': [0.01, 0.5, 1], 'hidden_layer_sizes': [(50,50), (100,100), (150,150)]}
The best set of parameters that I got after performing the GridSearchCV with 12-fold cross validation is:
Best parameters for NeuralNetwork= {'alpha': 0.01, 'hidden_layer_sizes': (150, 150), 'learning_rate_init': 0.01}
Using these parameters I trained my model and then tested the model on unseen test data and got the following results:

Accuracy	83 %
F1 Score	0.6194029850746268
Precision-Recall curve AUC	0.659713601158968
Cohen’s Kappa	0.5100315437208781
ROC AUC	0.7592245967545044

So, considering Cohen’s Kappa metric for all the classifier, even if the accuracy of each classifier is high, we are getting less values of kappa because it also considers the bias and evaluates the model considering both the classes instead of just considering the class with majority example. Another metrics that I am using is the Precision-Recall curver AUC because it considers severe imbalance of the dataset and highly used for imbalance classification. So, after analysing the above curve I can conclude that Decision Tree classifier for my dataset performs best because it has highest PR curve AUC and its Cohen’s Kappa value is closest to 1 as compared with other classifiers.

Ans b: Using BaggingClassifier in python I have then applied the bagging with replacement on the above classifiers with best parameters that I got from the GridSearchCV. Following is the list of members that I have passed as a paramters for **n_estimators** for the BaggingClassifier: **Members = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]**

So for my **1-NN Classifier** the performance of my model kept increasing for 2, 4, 6, and so on until I got the optimal number of estimators i.e. the members was **“n_estimators = 16”**. It gave me a **“Precision-Recall AUC of 0.5687019158808562”** and **“Cohen’s Kappa of 0.3786526234667009”.** So moving further, I used 16 as the best performing ensemble size and evaluated the performances by changing the number of instances in the bootstrap samples. For lower number of bag sizes I got best performance and then the performance decreased for later values of bag sizes. Following is the list of different values that I provided for **max_samples** parameter:

Samples = [0.10, 0.25, 0.5, 0.75, 1.0] , for the value of 0.1 i.e. **10% of the bag size** I got the best results:

Members	16
Bag Size	10 %
Accuracy	81 %

F1 Score	0.5130434782608695
Precision-Recall curve AUC	0.5831706376744544
Cohen's Kappa	0.40033195909407293

So, for my **Decision Tree Classifier** the performance of my model kept increasing for 2, 4, 6, and so on until I got the optimal number of estimators i.e. the members was **"n_estimators = 8"**. It gave me a **"Precision-Recall AUC of 0.7126795375379827"** and **"Cohen's Kappa of 0.5767718271112914".** After 8 the performance didn't change and decreased a little for the later values of members. So, moving further, I used 8 as the best performing ensemble size and evaluated the performances by changing the bag size and unlike 1-NN this model performed best for larger value of bag sizes. Following is the list of different values that I provided for **max_samples** parameter: **Samples = [0.10, 0.25, 0.5, 0.75, 1.0]** , for the value of 1 i.e. **100% of the bag size** I got the best results:

Members	8
Bag Size	100 %
Accuracy	87 %
F1 Score	0.658119658119658
Precision-Recall curve AUC	0.7126795375379827
Cohen's Kappa	0.5767718271112914

So, for my **Neural Network** the performance of my model kept increasing for 2, 4, 6, and so on until I got the optimal number of estimators i.e. the members was **"n_estimators = 20"**. It gave me a **"Precision-Recall AUC of 0.6697246098054537"** and **"Cohen's Kappa of 0.5260273521715517".** So moving further, I used 20 as the best performing ensemble size and evaluated the performances by changing the number of instances in the boostrap samples. It didn't perform well for the lower values of bag sizes. Following is the list of different values that I provided for **max_samples** parameter: **Samples = [0.10, 0.25, 0.5, 0.75, 1.0]** , for the value of 0.75 i.e. **75% of the bag size** I got the best results:

Members	20
Bag Size	75 %
Accuracy	84 %
F1 Score	0.626984126984127
Precision-Recall curve AUC	0.6713063318823208
Cohen's Kappa	0.5280255736497681

According to above results, my Decision Tree and Neural Network are performing better with higher percentage of bag size i.e. more number of instances per sample because if we reduce the variance here (increasing percentage of max_samples), then for this particular dataset BaggingClassifier is performing best as it achieve better generalisation results. On the other hand, 1-NN classifier is performing better with high variance (less percentage of max_samples) for this particular database.

Ans c: Using the best parameters for each calssifier that I got from the GridSearchCV I am evaluating their performance by applying Random subspacing technique without replacement by using BaggingClassifier in python. To apply random subspacing you just have to specify the max_features parameter. By deafult the value of max_features is 1.0 i.e. 100%. The list of members that I am passing for the BaggingClassifier as "n_estimators" are: **[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]**

So, for my **1-NN Classifier**, when I applied random subspacing with the above number of members, the optimal ensemble size for 1-NN that I got is **"n_estimators = 16"**. With this ensemble size I am getting **"Precision-Recall AUC of 0.5687019158808562"** and **"Cohen's Kappa of 0.3786526234667009"**. After getting the best ensemble size then I am checking for different values of features i.e. number of features that are used by the BaggingClassifier. Following is the list of values that I am passing for the **max_features** parameter:

Features = [0.10, 0.25, 0.5, 0.75, 1.0], after passing these, for the value of 0.25 i.e. **25% of the subspace size** I got the best results:

Members	16
Subspace Size	25 %
Accuracy	84 %
F1 Score	0.5226130653266331
Precision-Recall curve AUC	0.646659556952552
Cohen's Kappa	0.4388880138604505

So, for my **Decision Tree Classifier**, when I applied random subspacing with the above number of members, the optimal ensemble size for 1-NN that I got is **"n_estimators = 8"**. With this ensemble size I am getting **"Precision-Recall AUC of 0.7126795375379827"** and **"Cohen's Kappa of 0.5767718271112914"**. After getting the best ensemble size then I am checking for different values of features i.e. number of features that are used by the BaggingClassifier. Following is the list of values that I am passing for the **max_features** parameter:

Features = [0.10, 0.25, 0.5, 0.75, 1.0], after passing these, for the value of 1.0 i.e. **100% of the subspace size** I got the best results:

Members	8
Subspace Size	100 %
Accuracy	87 %
F1 Score	0.658119658119658
Precision-Recall curve AUC	0.7126795375379827
Cohen's Kappa	0.5767718271112914

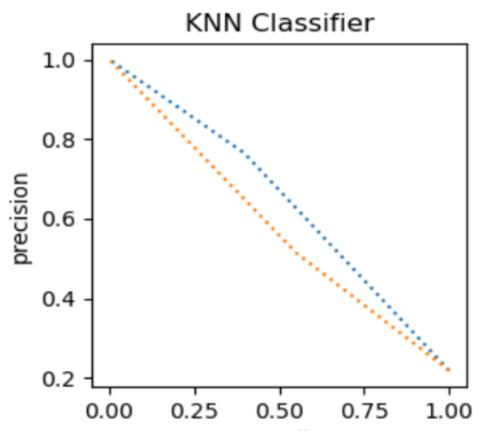
So, for my **Neural Network**, when I applied random subspacing with the above number of members, the optimal ensemble size for 1-NN that I got is **"n_estimators = 20"**. With this ensemble size I am getting **"Precision-Recall AUC of 0.6748269720101782"** and **"Cohen's Kappa of 0.533305750930906"**. After getting the best ensemble size then I am checking for different values of features i.e. number of features that are used by the BaggingClassifier. Following is the list of values that I am passing for the **max_features** parameter:

Features = [0.10, 0.25, 0.5, 0.75, 1.0], after passing these, for the value of 0.5 i.e. **50% of the subspace size** I got the best results:

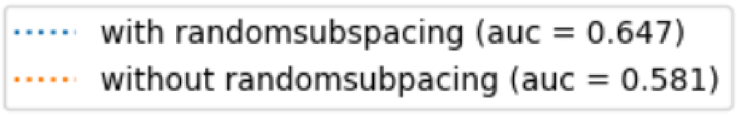
Members	20
Subspace Size	50 %
Accuracy	86 %
F1 Score	0.6413502109704641
Precision-Recall curve AUC	0.6944002352489317
Cohen's Kappa	0.5543048904114378

Ans d: According to the lectures, Decision Tree and Neural Networks will best perform with the bagging i.e. bootstrap aggregation where samples are randomely chosen with replacement from the training data. The reason for this is that bootstrap aggregation works well on unstable classifiers like decision trees and neural networks. On the other hand, KNN classifier should perform well with Random Subspacing technique without replacement because it works well for stable classifiers like KNN. It also encourages diversity as the subsets of features are selected randomly without replacement. After performing both the bootstrap aggregation and random subspacing on various classifiers the conclusion that I can draw is as follows:

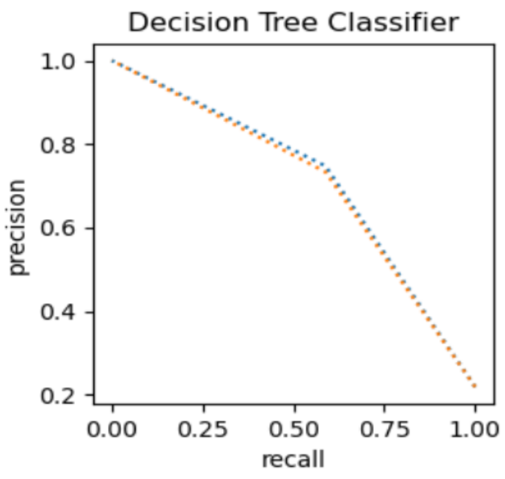
- 1-NN Classifier: This classifier performed best with random subspacing with having **16** number of estimators i.e. the ensemble size and having **25%** of the subspace size. Following chart illustrates the differences between normal 1-NN and 1-NN with Random Subspacing:



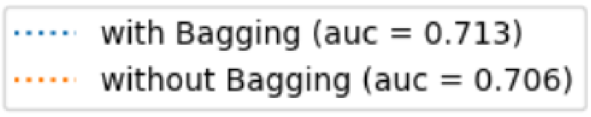
Thus, I can see that KNN actually performed best with random subspacing as I expected it to. With a subspace size of 25%, it has also encouraged the diversity by achieving more variation in choosing the feature subsets. Bootstrap aggregation also improved the performance but not as better as compared to random subspacing.



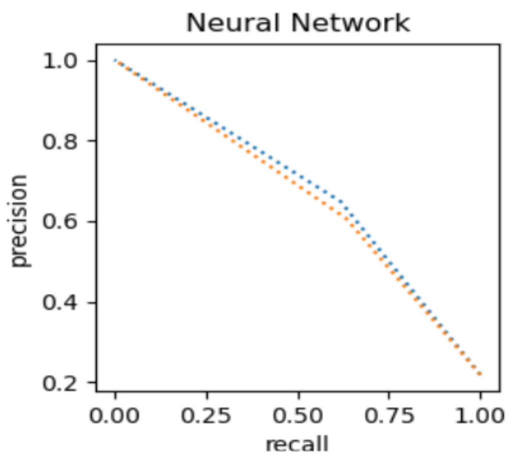
- Decision Tree: This classifier performed best with bootstrap aggregation with having **8** number of estimators i.e. the ensemble size and having **100%** of the bag size i.e max_samples. Following chart illustrates the differences between normal Decision Tree and Decision Tree with Boostrap Aggregation i.e. Bagging:



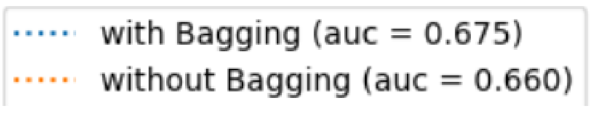
As expected, Decision Tree Classifier performed best with bootstrap aggregation by giving the highest precision recall auc score. In this case, it choose 100% of the bag size because it focuses for better generalization results. I can't see any major change when I applied random subspacing to the Decision Tree Classifier.



- Neural Network: This classifier performed best with bootstrap aggregation with having **20** number of estimators i.e. the ensemble size and having **100%** of the bag size i.e max_samples. Following chart illustrates the differences between normal Decision Tree and Decision Tree with Boostrap Aggregation i.e. Bagging:



As expected, Neural Network performed best with bootstrap aggregation by giving the better precision recall auc score than the normal neural network i.e. without bootstrap aggeration. In this case, it choose 75% of the bag size because it focuses for better generalization results. When I applied random subspacing on neural network, it didn't give any better results than bootstrap aggregation.



Therefore, the results that I got were pretty much similar to what I expected to happen. I was expecting Decision Tree Classifier and Neural Network to perform more better when bootstrap aggregation was applied, instead there was just a little improvement in the performance. In the case of random subspacing, the performance of KNN Classifier was highly improved as per the expectations.

Question 2:

Ans a: Interpretability is nothing but explaining the entire process of the machine learning models i.e. why a model makes such predictions, also extracting knowledge about the relationships that are present in the data and the relationships that are learned by the model. It can also be said as a part of drawing insights for the ML models. There are different types of models that you might have to explain to any layman. For instance, if you want to interpret linear regression models which is a supervised technique then it might be easy for you explain the predictions that your linear model is making. This is because it is user-friendly and your predicited output Y is completely dependent on the values of X. So, it is easy to interpret the predictions i.e. because of this value of X, we are getting this value of Y. If we consider neural network, then it may have many hidden layers and large numbers of neurons and it is very difficult to explain which neuron is performing which task. These models are often called as black-boxes where it is difficult for us to interpret the model and explain the prediction to a layman. Suppose if a model is predicting class A for a particular query and there is similar example in training data which belongs to class B then it might be that the model is incorrectly predicting for the given query. So in this case, there might be a chance that your model is overfitted or your model is unfamiliar with new data i.e. not up-to-date. So the old training examples with the combination of new data available should be modified and used to re-train your model and also take into consideration that the examples you choose to train your model should result into least number of False Negatives and False positives. Feature importance is one of the interpretability methods in ML. We can easily find out weights of various features in our model by using feature importance ranking techniques like coefficient of determination (rsquare) in which the correlation between each feature with the target feature is calculated. Ridge regression and lasso are another techniques which derives weights of different features during training based on their relative.

Ans b: The difference between actual output and predicted output is the error of our model and the is calculated by the summation of reducible and irreducible error. So our bias and variance are two components of reducible error. If there is high bias and high variance then our model is not consistent and also the predictions are far away from the actual value. When there is high bias and low variance then the model is consistent but predictions are far way from the actual values. For low bias and high variance our model predictions are close to the target but are highly variable. For low bias and low variance our model is consistent and also predicting very close to the actual values. So the right balance between the bias and variance is nothing but the bias-variance trade-off. We have to make sure that the model is neither overfitted nor underfitted in any case. Decision Tree generally suffer from high variance as a large tree will try to adjusts the prediction according to every single input. Decision Trees generally tends to learn the training set along with the noise but then perform poorly on the unseen data. On the other hand, linear regression and logistic regression suffers from high bias because they are more prone to underfit. Bagging technique is mostly helpful for reducing the vairance. For instance, Random forest can reduce the high variance by combining the output of multiple decision trees and getting an optimal soluton. In the case of high bias, boosting technique is more helpful because it creates a model that is less biased than the weak learner that are used to create the model.

Ans c: Following are the limitations of llyods algorithm:

- **Sensitive to outliers**: If there an outliers in the data and then it will skew your cluster to a alrge extent. A single outlier can increase the size of a cluster.
- **Sensitive to initialization of the initial centroids**: The time taken to reach the local minima depends on the initial centroids value. Different values lead to different solutions.
- **Spherical clusters**: Since euclidean distance is used to calculate the distance between the points and centroid, the algorithm treats the data to be isotropic, so, the clusters are of spherical shape. It is very difficult to cluster data that have complex geometrical shapes.
- **It creates different clusters for same data**: If centroids are choosen at random, everytime it will give different clusters.
- **It also takes a lot of time to converge i.e it requires many iterations.**

So to choose best values for centroids k-means ++ algorithm is used. In this, an initial random point is choosen as a single centroid. The distances of all the data points to the centroid is calculated. The data point that have large distance from the current centroid have higher probability of getting picked as the next centroid. This means that the probabilty of picking a point as a cluster is directly proportional to the distance between that point and the previously picked centroid. So the reason behind k-means ++ algorithm is that it ensures that the centroids that are picked are far away from each other. So by doing this, the chances of picking intial centroids that lie in different clusters increases. Also, since the centroids are choosen from the data points, each of the centroid will have some points associated with it. This resolves the issue of initailizing optimal intial centroids.

Ans d: The R suqare measure can be best understaood by a following example:

Let **y = mx+b**, be the line that is drawn through the data such that using this line we can calculate the error i.e. the difference between actual and predicted values of **y**. Using this we can check whether the particular line best fits all the data points or not. So error can be calculated as the difference between the actual y values and the predicted y values on the line and is given by **Squared Error = summation of (ActualYi - PredictedYi)^2** . Now that we have calculated the error, we will check how good this line fits the data by calculating the percent of variation in y that is decsribed by the variation in x or the regression line. So we get the total variation in y by:

Variation in y = summation of (ActualYi - Ymean)^2. Then we calculate the total variation that is not not described by the line y = mx+b i.e. the sum of squared error. So following equation gives us the percent of variation in y that is described by variation in x or the regression line: **1 - (sum of squared error / total variation in y).** This equation is then called the **coefficient of determination or R square measure**. It is used to determine how the line fits the model. If the value of R square measure is close to one then it means the line is a better fit and if the value is close to 0 then it means that the line is not a good fit.

Ans e: Precision is the number of examples that your model accurately predicts i.e the accuracy of how your model correctly predicts positive class. On the other hand, recall is the proportion of postive examples that are correctly predicted. So in the case of binary classification, we get the probabilties whether a particular example belong to positive or negative class. After getting this probabilities we can determine which class a particular example belongs to. This can be done by setting a threshold value, this value depends on the type of problem we are looking at i.e. based on our needs. Then the probabilities of each example will then be compared with the defind threshold and based on that we can determine the class. For lowest value of threshold the number of false positives will be large so my precision is less and recall will be high where the number of false negative is less. Precision starts increasing as we increase the threshold which reduces the number of false positives. Also the recall starts decreasing as we increase the value of threshold where the number of false negatives will increase. So whenever we want to achieve high precision it will give us a low recall value and vice versa. Suppose for example. I want to predict whether a patients have cancer when they actually do, then my model will need to predict with high precision which in turn will reduce the recall as I increase the threshold and if my model is predicting that patients do not have cancer but actually they do, then this will be very bad. So in this case instead of having higher threshold values I can lower the threshold so that my model predicts that the patients might have cancer and then they can have their further checkup. This will result in higher recall and lower precision.

Ans f: A neural network can have multiple layers of neurons and these neurons accept inputs and then process it and provides output with the help of activation function. Each connection in the neural networks is assigned a weight and when we multiple this weight by the input value it gives importance of that particular relationship in the neuron. So learning of the parameter values i.e. training the neural networks is done in an iterative fashion of going forward in the networks and then returning in backward manner. Initially in **forward propagation** all the training data is passed and it crosses the entire network. In this process all the neurons from each layers applies some transforamtion to the input they receive from the previous layers and passes the output to the next layer. Once the input crosses the entire networks i.e. reaches the final layer, it gives us an output based on the input values. This output is then compared with the actual values by using a **loss function** to get the error. We then aim to lower this error as much as possible. Once we have our loss calcualted the next phase is the **backward propagation**. In this phase the loss that is calcualted propagates to all the neurons starting from the final layer and in a backward manner. Neurons in all the hidden layers get a fraction of the loss based on their contribution towards the output. This information in passed till the first layer in the network. Using this information the weights of connections between neurons can be adjusted. The process of changing the weights can then be done by a technique known as gradient descent. In this technique, the weights are changed in small increments by calculation the derivatives of the loss function.