CSE 601: DATA MINING AND BIOINFORMATICS

FALL 2019

PROJECT 3: CLASSIFICATION ALGORITHMS

REPORT

Keerthana Baskaran (50288944)

Sriram Venkataramanan(50289666)

Vivek Nagaraju (50288711)

TASKS TO BE IMPLEMENTED

- Implement three classification algorithms by yourself: **Nearest Neighbor, Decision Tree,** and **Naïve Bayes**.
- Implement **Random Forests** based on your own implementation of Decision Tree.
- Adopt 10-fold **Cross Validation** to evaluate the performance of all methods on the provided two datasets in terms of **Accuracy, Precision, Recall**, and **F-1 measure**.
- Kaggle Competion

DATASET:

Each line represents one data sample. The last column of each line is class label, either 0 or 1. The rest columns are feature values, each of them can be a real-value (continuous type) or a string (nominal type).

PERFORMANCE METRIC: Before we see how to calculate both the coefficients, we need to understand 4 terms related to it.

- **True Positives:** Relevant data retrieved.
  It means two points are in same clusters according to the ground truth value given to us (our input file) and we also classify them in the same cluster. (Eg p is 1 and q is also 1)
- **True Negatives:** Irrelevant data omitted.
  It means two points are not in same cluster according to the ground truth value given to us (our input file) and we also classify them in different clusters. (Eg p is 0 and q is also 0)
- **False Positives:** Irrelevant data retrieved
  It means two points are not in same cluster according to the ground truth value given to us (our input file) but, we classify them in same cluster (Eg p is 1 and q is 0)
- **False Negatives:** Relevant data omitted.
  It means two points are in same cluster according to the ground truth value given to us (our input file) but, we classify them in different clusters. (Eg p is 0 and q is 1)

		Predicted	
		TSS	FSS
Actual	TSS	TP	FN
	FSS	FP	TN

ACCURACY:

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations.

$$Accuracy = (TP+TN)/(TP+FP+FN+TN)$$

PRECISION:

It is the ratio of correctly predicted positive observations to total number of positive predicted observations.

$$Precision = TP/(TP+FP)$$

RECALL:
It is the ratio of correctly predicted positive observations to all the observations in actual class yes.

$$Recall = TP/(TP+FN)$$

F1 – Measure:
It considers both precision and recall. It is the harmonic mean of precision and recall. It is the best if there is some sort of balance between precision and recall in the system. Oppositely F1 score isn't so high if one measure is improved at the expense of the other.

$$F1\ Score = 2*(Recall * Precision) / (Recall + Precision)$$

10 FOLD CROSS VALIDATION:

In every algorithm we applied 10 fold cross validation i.e split the data as 90 percent training data and the rest 10 percent as testing data. At the end we obtained the average of the metrics discussed above.

TASK 1:

1) K NEAREST NEIGHBOURS

It is a simplest classification technique when there is little or no prior knowledge about the distribution of the data. K in KNN refers to number of nearest neighbours.
It is a type of instance based/lazy algorithm.

FEATURES OF KNN ALGORITHM:

1) Supervised machine learning algorithm: The target variable is known already.
2) Lazy algorithm: It does not have a training step. Every data point will be used only at the time of prediction. Since there is no training step, the prediction step is costly. It is an eager learner algorithm, as it eagerly learns during the training step.
3) Classification and Regression: Used for both the purposes.
4) Feature similarity: It is the metric used to cluster the new point that will fall into.

GENERIC ALGORITHM:

- Initially, we have to decide the value for K. Usually K should be an add number.
- Once a new point comes, find the distance of the new point to each and every point in the training data.
- Then identify the K nearest neighbours to the new data point.
- In order to classify the point, count the number of data point in each category among the k nearest neighbours. The new data point will belong to the class that has more number of neighbours.
- In order to obtain regression, the value for the new data point will be the average of k neighbours.

CHOOSING THE VALUE OF K

- The choice of K will have a drastic impact on the results obtained from KNN. Hence the value of K has to be chosen very carefully.
- Choosing better value of K will improve accuracy and precision of the results obtained.
- Choosing lesser value of K might sometimes result in predicting with less accurate information and choosing bigger value of K might result in including outliers and which will result in misclassification of data.
- Hence it is very essential to choose the right value for K which can be found by re iterating the date many times for different values of K.

DISTANCE METRICS USED IN KNN

- **Euclidean distance, Manhattan distance, Hamming Distance, Minkowski Distance, Chebychev Distance**

In our code implementation we have used Euclidean distance.

ALGORITHM IMPLEMENTATION OF OUR CODE:

1) Initially we obtain the file name and the value of K from the user. The value of K is nothing but number of nearest neighbours to be considered.
2) Once the file is obtained, it is converted into a np array using np.asarray() function.
3) We pre-processed the data i.e. the points are converted into a matrix form using np.matrix() and the ground truth from the input file is stored in a variable separately.
4) While pre processing, the categorical data is checked and respective columns are stored separately.
5) While pre processing, we normalize the data using the Min Max Normalisation technique. It linearly transforms x to y = (x-min)/max-min where min and max are the minimum and maximum values in X, where X is the set of observed values of x.
6) Then we split the data according to a 10 fold cross validation technique, where 90 % is used for training data and 10% is used for test data at each iteration.
7) knn_implement(train_data,train_label,test_data,k) is called on the split data.
    - For each Test data, the distance is calculated from every training data using the function np.linalg.norm(traning_data[i] -test_data[i]). It uses the Euclidean distance.
    - Once the distances are calucalted, the list is sorted.
    - From the sorted list top K training data indices are obtained. These are the top K nearest neighbours for that test data.
    - Once the indices are obtained, we check the training label and assign the max of training label to the predicted test label.
    - The process is repeated for each of the test record in K fold set.
8) Once the test labels are predicted, it is compared with the actual test labels.
9) The metrics are calculated by calling the function performance_metric(test_actual, test_predicted)
    - The true positives, true negatives, false positives, false negatives are obtained by comparing the actual test label and predicted test label.

- Then the metrics Accuracy, precision, recall, F1-measure are calculated by mentioning the above mentioned formulas.

10) Now again we pick another dataset from the remaining 9 datasets. Each time the train and test data are split in 90 to 10 percent ratio and the above steps are repeated.
11) We continue until all the parts are covered, and the final metrics are calculated which are the mean of the values obtained at each step.

OUTPUTS OBTAINED:

1) File : project3_dataset1.txt

K = 5
Average Accuracy Obtained: 96.83583959899748
Average Precision Obtained: 98.14364876385336
Average Recall Obtained: 93.41135656381923
Average F_measure Obtained: 0.9557889802089171

K = 11
Average Accuracy Obtained: 97.0081453634085
Average Precision Obtained: 98.69565217391303
Average Recall Obtained: 93.63423046411248
Average F_measure Obtained: 0.9603583367950286

K	Avg Accuracy	Avg Precision	Avg Recall	Avg F_measure
10	97.0081453634085	99.52380952380952	92.67968500956702	0.9593289129270117
20	95.95238095238096	98.83333333333333	90.2737209012269	0.9425405321718628
30	95.77694235588974	98.83333333333333	89.90335053085653	0.9405811707785393
50	95.59837092731831	98.80701754385964	89.44880507631107	0.9379098583511293

2) File project3_dataset2.txt

K=5
Average Accuracy Obtained: 65.8140610545791
Average Precision Obtained: 51.22005772005772
Average Recall Obtained: 40.349918356497305
Average F_measure Obtained: 0.439391783785293

K=11
Average Accuracy Obtained: 68.62164662349676
Average Precision Obtained: 59.415181224004755
Average Recall Obtained: 34.689902933323985
Average F_measure Obtained: 0.4229342307732401

K	Avg Accuracy	Avg Precision	Avg Recall	Avg F_measure
10	67.53006475485662	55.42857142857143	27.383391462338828	0.35185811531463707
20	70.790934320074	66.39285714285714	33.09847170373486	0.43076841961794227
30	71.85938945420907	68.94444444444446	35.033083875189135	0.45609338563800805
40	71.21646623496763	66.72222222222223	34.36892347418663	0.4463314808761033

RESULT ANALYSIS:

- Thus we have run KNN for both the data sets and the results are noted as above.
- KNN tends to perform better with the project3_dataset1.txt than project3_dataset2.txt
- It performs well with first dataset as the features are of continuous values as compared to second data set which is a mixture of both continuous and categorical values.
- The K value has to be chosen correctly which is a disadvantage in this algorithm, a too large k or small k could result in mis-classification. Choosing the right k is always a difficult task in Knn.
- Time complexity is quite high as the number of records increases, as it has to find the distance of a test sample with respect to all training samples. Also to find the k values, we need to apply sorting on those values.
- In our case we chose Euclidean distance as the metric, choosing another metric might result in different classification, thus we can see the algorithm is based on the metric we decide to choose.

PROS:

- It is very effective algorithm when the data set is large.
- There is no training phase involved.
- It learns complex models very easily.
- It is robust to noisy training data.
- Used in both classification and regression.

CONS:

- It is computationally expensive as it searches the nearest neighbours for the new point at the prediction stage.
- False intuition is possible and sensitive to outliers which impacts accuracy in prediction.
- Sometimes, not clear which type of distance metric to be used.
- Huge memory requirement is also a big concern

---

2) NAÏVE BAYES

- It is one of the biggest most interesting, mathematically sophisticated areas of machine learning and it is a supervised machine learning algorithm. It is a probabilistic classifier based on Bayes theorem.
- It is a classification algorithm for binary(two class) and multiclass classification problems. The technique is easiest to understand when described using binary or categorical input values.
- The calculation of the probabilities for each hypothesis are simplified to make their calculation tractable. They are assumed to be conditionally independent given the target values.
- It is a very strong assumption that is most unlikely in real data, i.e that the attributes do not interact. Nevertheless, the approach performs surprisingly well on data where this assumption does not hold.

GENERIC ALGORITHM FOR NAÏVE BAYES:

- It operates on the following intuition: you start off with some initial confidence in the labels 0 and 1, assuming it is a binary classification problem.
- When a new information becomes available, you adjust your confidence levels depending on how likely that information is conditioned on each label.
- When you have gone through all available information, the final confidence levels are the probabilities of labels 0 and 1.

BAYE'S THEOREM:

In a classification problem, our hypothesis (h) may be the class to assign for a new data instance. One of the easiest ways of selecting the most probable hypothesis given the data that we have that we can use as our prior knowledge about the problem. Bayes' theorem provides a way that we can calculate the probability of a hypothesis given our prior knowledge.

Bayes' Theorem is stated as:

$$P(h|d) = (P(d|h) * P(h)) / P(d)$$

Where

- **P(h|d)** is the probability of hypothesis h given the data d. This is called the posterior probability.
- **P(d|h)** is the probability of data d given that the hypothesis h was true.
- **P(h)** is the probability of hypothesis h being true (regardless of the data). This is called the prior probability of h.
- **P(d)** is the probability of the data (regardless of the hypothesis).

- You can see that we are interested in calculating the posterior probability of P(h|d) from the prior probability p(h) with P(D) and P(d|h).After calculating the posterior probability for a number of different hypotheses, you can select the hypothesis with the highest probability.

- This is the maximum probable hypothesis and may formally be called the **maximum a posteriori** (MAP) hypothesis.

ALGORITHM IMPLEMENTATION OF CODE:

1) Initially we obtain the file name and the value of K from the user. The value of K is nothing but number of nearest neighbours to be considered.
2) Once the file is obtained, it is converted into a np array using np.asarray() function.
3) We pre-processed the data i.e. the points are converted into a matrix form using np.matrix() and the ground truth from the input file is stored in a variable separately.
4) While pre processing, the categorical data is checked and respective columns are stored separately.
5) While pre processing, we normalize the data using the Min Max Normalisation technique. It linearly transforms x to y = (x-min)/max-min where min and max are the minimum and maximum values in X, where X is the set of observed values of x.

6) Then we split the data according to a 10 fold cross validation technique, where 90 % is used for training data and 10% is used for test data at each iteration.
7) The next step is to calculate class posterior probability for continuous data and categorical data.
8) For continuous data, we won't be able to count the values, so we used Gaussian distribution to find posterior probability
    - To do that we first obtained training continuous data and testing continuous data separately by deleting the categorical column indexes using the function np.delete.
    - We keep track of records whose training labels are 0 and 1 separately.
    - The mean and standard deviation of train_zero_label records and train_one_label records are found separately.
    - The Gaussian distribution is applied to find the posterior probability of a record being a 0 and 1, both are found by passing the respective values found above.
    - The process is repeated for every test record. At the end of this step, there are two lists cont_zero_result() and cont_one_result() which holds the values of test records being a zero probability and one probability.
9) The next step is to calculate Categorical probability of every test record for the categorical columns obtained above.
    - The class prior probability P(Hi) is obtained from the training data by dividing the number of training samples of class Ci by total number of training records. In this case Ci can be 0 or 1.
    - The Descriptor prior probability P(X) of the data point is by getting the count of records of that data point by total number of training records.
    - The Descriptor Posterior probability i.e P(X|Hi) is calculated by obtaining the count of training records of that data point by total number of records of that respective class. For each training sample,say X, we have X=(X1, X2,…. Xd) , where each feature is independent of the other.Hence $P(X \mid H_i) = P(X_1 \mid H_1) * P(X_2 \mid H_2)$ and so on. This is done for zero and one records separately.
    - The zero probability is handled while finding this by adding one to numerator and adding the count of unique values in that particular column to the denominator.
    - Finally the class posterior probability P(Hi | X) is calculated by multiplying the class prior probability P(Hi) and Descriptor Posterioir probability P(X|Hi) obtained above.
    - Here again we calculate the class posterior probability of the test record of being 0 and 1
    - The process is repeated for all categorical columns.
    - At the end of this step we again have two lists cat_prob_one() and cat_prob_zero() which holds the values of test records being a zero probability and one probability.
10) Now by looping to the number of test records,
    - We multiply the final result obtained from step 8 and step 9, i.e multiply the categorical and continuous probability obtained for zero and one respectively for each test record.
    - If the value of probability of one is higher, the test record is predicted as one else it is predicted as 0.
12) Once the test labels are predicted, it is compared with the actual test labels.

13) The metrics are calculated by calling the function performance_metric(test_actual, test_predicted)
    - The true positives, true negatives, false positives, false negatives are obtained by comparing the actual test label and predicted test label.
    - Then the metrics Accuracy, precision, recall, F1-measure are calculated by mentioning the above mentioned formulas.
14) Now again we pick another dataset from the remaining 9 datasets. Each time the train and test data are split in 90 to 10 percent ratio and the above steps 8 to 13 are repeated.
15) We continue until all the parts are covered, and the final metrics are calculated which are the mean of the values obtained at each step.

HANDLING ZERO PROBABILITY:

The zero probability is also handled in the code. That is whenever the value of P(X|C) becomes zero or 1, we add 1 to the numerator where we multiply prior and descriptor posterioir probability and add the length of number of unique features in that particular column to denominator ie descriptor prior probability as given in slides.

Ex. Suppose a dataset with 1000 tuples for a class C, income=low (0), income= medium (990), and income = high (10)

Use **Laplacian correction** (or Laplacian estimator)*Adding 1 to each case*

Prob(income = low|H) = 1/1003

Prob(income = medium|H) = 991/1003

Prob(income = high|H) = 11/1003

HANDLING CONTINOUS FEATURES: We applied Gaussian distribution inorder to calculate the probability of contious features.

OUTPUTS OBTAINED:

1) File : project3_dataset1.txt
   Average Accuracy Obtained: 93.31453634085214
   Average Precision Obtained: 91.31925736801898
   Average Recall Obtained: 90.44426356826324
   Average F_measure Obtained: 0.9078725938618137
2) File: project3_dataset2.txt
   Average Accuracy Obtained: 70.34690101757633
   Average Precision Obtained: 57.09275640080593
   Average Recall Obtained: 61.59396451501714
   Average F_measure Obtained: 0.5867396458138147

RESULT ANALYSIS:

- It performs well for first dataset compared to second data set, as the first file comprises only continuous features and the second file comprises of both continuous and categorical features. Thus it can handle any data whether it is continuous, categorical or nominal.
- It assumes all features are independent and calculates values, which may not happen in real life.

PROS:

- It is easy and fast to predict a class of Test data set.
- Naïve Bayes classifier performs better compare to other models assuming independence.
- It performs well in case of categorical input variables compares to numeric variables.
- It does not require a lot of training data.
- Handles both continuous and discrete data.

CONS:

- It is sometimes called a bad estimator.
- It makes a very strong assumption on conditional independence.
- It requires Laplace correction or adding 1 for simple cases to avoid dividing by zero.

---

3) DECISION TREES:

- It is one of the most popular machine learning algorithm used and it is used for both classification and regression.
- They often mimic the human level thinking so it's so simple to understand the data and make some good interpretations.
- It makes us see the logic for the data to interpret.

TWO TYPES OF DECISION TREES:
1) Classification Decision Tree.
2) Regression Decision Tree.

It is a tree where,

- Node -> Represents a feature or attribute.
- Link -> Represents a decision rule.
- Leaf -> Represents an outcome which can be categorical or continuous value.
- Splitting -> Process of splitting a node into two or more sub nodes.
- Decision Node -> A sub-node splits into further sub nodes, then it is called a decision node.
- Pruning -> Reducing size of decision tree by removing nodes. The process is called pruning.

The idea of decision tree is to build a tree in the above order for the entire data and process a single outcome at every leaf, so that we can minimize the error at every leaf.

The algorithm we have used for the code is Gini Index.
GINI INDEX:

- It states that, if we select two items from a population at random, then they must be of same class and probability for this is 1 if population is pure.
- It works well with categorical variable and it performs binary splits.
- Higher value of Gini leads to higher homogeneity.

- Classification and Regression Trees uses Gini method for binary splits.

ALGORITHM EXPLANATION:

1) We first Convert all our categorical data into numerical values . We do this by creating a dictionary in python where the keys will be the categorical values and values of the dictionary will be the corresponding numerical values.
2) We split our dataset into 10 chunks to do the 10 cross fold validation.
3) We implement the calculate_gini_index function which performs an exhaustive search over all the features and considers different values of each feature to set as the threshold and calculate the gini index .
4) At each node , we take the split which gives us the lowest gini index and build the tree on top of this node.
5) We use recursion to implement the decision tree. There are two types of nodes in the tree .
 a) Internal Nodes : These nodes in turn contain other nodes . In our implementation the value for these nodes will be in turn a dictionary pointing to it's child nodes
 b) Terminal / Lead Nodes: These nodes contain the output label of this node => there are no child nodes for these nodes.
6) At each Internal Nodes we call the split function to select the attribute which gives the lowest gini index and recursively build the left and right subtrees.
7) To predict the output for a dataset , we first start the traversal from the root node. If the value is less than the value at the root node , we recursively traverse the left subtree else we traverse the right subtree.
8) If we reach a terminal node => we stop the traversal and simply return the output of that node (which will also be the prediction for that dataset)

CHOICE DESCRIPTION:

1) **Categorical Feature**: We Convert all our categorical data into numerical values . We do this by creating a dictionary in python where the keys will be the categorical values and values of the dictionary will be the corresponding numerical values.
If the data is a Categorical variable and is equal to the threshold value then the data is appended to the left list else the data is appended to the right list
2) **Continous Feature:** If the data is a continuous variable and is less than the threshold value then the data is appended to the left list else the data is appended to the right list
3) **Best Feature:**At each node , we choose the feature which gives us the minimum gini index.
4) **Stopping criteria:** If at any node , we get 0 gini index , we consider it to be a terminal node(Leaf Node) and stop the recursive function to build the subtrees.

OUTPUTS OBTAINED:

1) **File : project3_dataset1.txt**
 Accuracy: 92.43421052631578
 Precision: 91.47086922452316
 Recall: 88.53501551356514

F1 Measure: 0.8997900082673099

2) **File : project3_dataset2.txt**

Accuracy: 64.24606845513412
Precision: 48.432040998217474
Recall: 40.14014859409596
F1 Measure: 0.4389796235888718

RESULT ANALYSIS:

- We could see that we are getting a higher accuracy with dataset1 but getting a lower accuracy with dataset2 . A plausible reason for this is because there are lower number of features in dataset2 => we get higher variance when selecting the attributes to split for each node.
- If the number of features are higher => there is low variance when splitting the nodes based on the gini index.

PROS:

- It is simple to understand, interpret and visualize.
- Variable screening or feature selection is possible.
- It requires very little effort for data preparation.
- Handles both categorical and continuous data.
- It is easy and inexpensive to construct.

CONS:

- It can be unstable because of small variations in the data.
- It has to be pruned at times to avoid overfitting.
- Complex and time consuming when we have to construct large decision trees.

---

4) RANDOM FOREST:

- It is an extension of a decision tree.
- Random forest consists of large number of individual decision trees which operates as an ensemble. It is an ensemble machine learning algorithm.
- It classifies more accurately since it uses more than one decision tree.
- Each individual tree in the random forest splits out a class prediction.
- The class with the most votes becomes our model's prediction.
- Therefore, a large number of relatively uncorrelated models operates as a committee will outperform any of the individual constituent models.

KEY CONCEPT:

- Low correlation between the models is the key. For example, in Investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than sum of its parts, uncorrelated modes can produce ensemble predictions more accurately than any of its individual prediction.
- **The reason for this wonderful effect is that trees protect each other from their individual errors.**
- Even if some trees are wrong, many of the trees will be right, so as a group the trees would be able to move in the right direction.

PRE-REQUISITES FOR RANDOM FOREST

- There needs to be some actual signal in our features so that models built using those features do better than random guessing.
- The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

PSEUDOCODE OF RANDOM FOREST:

- Assume a number of cases in the training set is N. Then, sample of these N cases is taken at random but with replacement.
- If there are M input variables or features, a number m<M is specified such that at each node, m variables are selected at random out of the M. The best split on these m is used to split the node. The value of m is held constant while we grow the forest.
- Each Tree is grown to the largest extent possible and there will be no pruning.
- Predict new data by aggregating the predictions of the n trees (i.e. majority votes for classification, average for regression).

ALGORITHM IMPLEMENTATION OF CODE:

1) We first Convert all our categorical data into numerical values . We do this by creating a dictionary in python where the keys will be the categorical values and values of the dictionary will be the corresponding numerical values.
2) We get the number of trees as input from the user.
3) We then create a subsample of our training dataset . We repeat this process for each tree.
4) To create a subsample of the dataset , we loop till the length of the dataset and get a random number each time . we then append the row corresponding to this number . At the end of this process , we would have created a random subsample with length the same as that of the original dataset.
5) we then split our subsampled dataset to build our decision tree.
6) When we are splitting the node , we need to select only a subset of the features instead of all the features. To do this , we first create a random number and then select features based on this.
7) We do the above steps for all the trees.
8) We define a dictionary to store the results of all trees . The key for this dictionary will be the indices of the test data and the value will be a list containing the prediction of each tree => for each key the value will be a list with length = number of trees.
9) We then take the maximum vote for each index in the dataset . This will be the final prediction of our random forest.

NUMBER OF FEATURES: We have used 20 percent of all the features for making a decision to split at each node.

RESULTS OBTAINED:

1) File : project3_dataset1.txt
   Enter the number of tree: 2
   Accuracy: 93.1484962406015
   Precision: 97.3974358974359
   Recall: 82.63198046442002
   F1 Measure: 0.894091997075067

   Enter the number of tree: 3
   Accuracy: 94.37030075187971
   Precision: 93.10369021430402
   Recall: 90.85740115531387
   F1 Measure: 0.9196683133222773

TREES	ACCURACY	PRECISION	RECALL	F1 – MEASURE
4	95.25375939849621	98.97698209718669	88.23355706343908	0.9329700386521678
5	95.77694235588972	96.49766126546312	91.78878509342167	0.9408434078181788
6	94.72117794486215	97.4878364389234	88.04592534734438	0.9252662897293625
7	95.0689223057644	95.23154623154623	91.36892796417588	0.932602591183202

2) File : File : project3_dataset2.txt
   Enter the number of tree: 2
   Accuracy: 62.13691026827013
   Precision: 39.03174603174603
   Recall: 18.38779495358443
   F1 Measure: 0.24998727972955712

   Enter the number of tree: 3
   Accuracy: 61.8963922294172
   Precision: 44.84691452338512
   Recall: 39.392436510857564
   F1 Measure: 0.41943087437893073

Trees	ACCURACY	PRECISION	RECALL	F1 – MEASURE
4	66.6604995374653	52.74084249084249	30.394574723522094	0.3856444176940848
5	65.5781683626272	50.38933340481327	40.7384304292199	0.4505284156923658
6	65.9990749306198	51.663752913752916	31.71214604109341	0.3930076911840398
7	63.6586493987049	47.34708428826076	40.3078405512616	0.4354481456808828

PROS

- Like other algorithms, even this is used for classification and regression.
- It won't over fit the model

- It is used to handle large data set with higher dimensionality.
- It handles the missing values and also maintains accuracy for missing data..

CONS

- It is a good model for classification, but not a better model for regression at times.
- We have very little control over what the model does.
- Since there are too many classes, misclassification is also possible.
- It is a complex model

KAGGLE REPORT:

TASK :

Apply various tricks on top of any classification algorithm discussed in class (including nearest neighbor, decision tree, Naïve Bayes, SVM, logistic regression, bagging, AdaBoost, random forests) and tune parameters using training data.

ANALYSIS:

- We have done 38 submissions in Kaggle.
- Each day we tested an algorithm and found out the F1 – measure obtained for each algorithm.
- We fine-tuned the parameters of each algorithm in order to get the best result.
- In the end we found the algorithm that works the best.

KEY IMPROVEMENTS:

PRE PROCESSING OF DATA:

TRAINING DATA: We normalized the training data in the range of values 0 to 1. To do that, we took the mean of every column and standard deviation of every column and applied the below formula. The values are subtracted with respective mean and standard deviation

$$z = \frac{x - \mu}{\sigma}$$

TESTING DATA: We normalized the test data using the statistics obtained from the training data. The same formula is applied, but the mean and standard deviation of each column is obtained from training data.

SEARCHING FOR HYPER PARAMETERS:

We used GridSearchCV provided by sklearn in order to find the best parameters of each algorithm.

10 – FOLD VALIDATION IN LOCAL:

We applied 10 fold validation for training data and found the accuracy and F1 score in local before submitting the files on Kaggle.

MAJORITY VOTING:

It is one of the key concept to improve accuracy. It is very simple, it just combines the predictions obtained from various ML algorithms to make final predictions.

Eg: 3 algorithms predict a record as 1 and 2 algorithms predict a sample as 0. Then it makes the final prediction as 1.

Let me explain in the order of algorithms we tried.

1) **LOGISTIC REGRESSION:**

class `sklearn.linear_model.` **`LogisticRegression`** *(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='warn', max_iter=100, multi_class='warn', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)* [source]

We fine tuned the parameters and obtained the F1 score as follows.

A) 0.7884 B) 0.76271 C) 0.76150 D) 0.76150 E) **0.86029**
We fined tuned the parameters penalty, solver, C in order to test the F1 measure.

Only in one case we obtained the F1 measure as 0.86029. The parameters at that time were solver = liblinear penalty = l1 and c=1.
Overall we found the algorithm is not better comparatively as it is inconsistent. So we rejected choosing it.

2) KNN

class `sklearn.neighbors.` **`KNeighborsClassifier`** *(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs)* ¶ [source]

By applying GridSearchCV we found the best K as 11 for the data set.

A) K = 5 , F1 Score: 0.82593
B) K = 11, F1 Score: 0.82119
C) K = 9 algorithm = ball_tree , F1 Score: 0.83056
D) K = 11, p=1(Manhatan distance) F1 Score: 0.82119
E) K = 7 algorithm = kd_tree, F1 Score: 0.83783

Thus we can see that the algorithm works better than Logistic regression. Still the F1 score ranges between 0.82 and 0.83 all the time. The major disadvantage of this algorithm is choosing the right K and the right metric. Hence we rejected this algorithm as well.

3) **Decision Tree:**

class `sklearn.tree.` **`DecisionTreeClassifier`** *(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)*

By applying GridSearchCV we found the best max_depth as 7 for the data set. We ran most of the parameters in GridSearchCV and found their best values and checked the F1 score for the following cases.

A) Default parameters, F1 Score: 0.78102
B) Max_depth = 7 , F1 Score: 0.78358
C) Max_depth=7 and criterian = “entropy” F1 Score: 0.78545
D) max_depth=7,criterion ='entropy',splitter ='random',class_weight ='balanced',presort =True, random_state=0 F1 score : 0.73517

E) max_depth=7,criterion ='gini',splitter ='best',class_weight ='balanced',presort =True, random_state=0) F1 Score: 0.82121

Thus we can see even after fine tuning the parameters using gridsearch, the accuracy is pretty much less than 80 all the time. It can be unstable with small change in input data. We rejected chosing this algorithm as well.

4) **SVM**

```
class sklearn.svm. SVC (C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True,
probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1,
decision_function_shape='ovr', random_state=None) ¶
```

By applying GridSearchCV we found the best C and gamma value as 0.001 and 0.1 respectively.

A) Default parameters: F 1 Score: 0.80757

B) C = 0.001, gamma = 0.1 F1 Score: 0.

We tried linear kernel as well. Still the F1 Score was in the range of 0.80 again. Even after choosing best parameters, there was not significant improvement in F1 Score. We rejected this algorithm as well.

5) **Naïve Bayes**

```
class sklearn.naive_bayes. GaussianNB (priors=None, var_smoothing=1e-09)
```

When we tried Naïve bayes in K fold cross validation and choosing the best parameters from GridSearchCV in training data in our local we found the accuracies were in range of 0.82 to 0.83. Thus we didn't give a separate submission for it and we rejected this algorithm.

6) **Random Forest**

```
class sklearn.ensemble. RandomForestClassifier (n_estimators='warn', criterion='gini', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto',
max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False,
n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None) ¶ [source]
```

We ran GridSearchCV and found the best estimators as 1000. Here again we applied 10 fold validation on training data and tested in our local. The F1 Score are obtained as follows:

A) Default Parameters: 0.84320

B) n_estimators=1000,bootstrap = True,max_features = 'sqrt' : 0.83828'

C) (n_estimators=1000,bootstrap=False,max_depth=100,max_features='auto', min_samples_split=7): 0.85521

D) n_estimators=800,bootstrap=False,max_depth=70,max_features='auto', min_samples_split=5: 0.85234

Thus we varied the best parameters and found the F1 Score was in range of 0.84 to 0.85 all the time. We were keeping Random forest in consideration, but did not finalize it as we thought of improving the F1 Score even better.

7) **ENSEMBLE MACHINE LEARNING:**

Now we started using majority voting concept by combining several algorithms for predictions. We obtained the best parameters of each algorithm and ran it at the same time

and predicted the output. Some of the combinations we tried are listed below.

A) Random Forest, Logistic Regression and KNN: 0.8534
B) RF, KNN and Naïve Bayes: 0.8574
C) RF, DT, BAGGING: 0.84949
D) RF DT LOGREG KNN BAGGING: 0.85135

Now we found Random forest works well, so we thought of extending to include AdaBoost algorithm which provided our best results.

FINAL CHOSEN ALGORITHM : ADABOOST

8) **ADABOOST**

```
class sklearn.ensemble. AdaBoostRegressor (base_estimator=None, n_estimators=50, learning_rate=1.0,
loss='linear', random_state=None)
```

- We applied RandomSearchCV to reduce the scope for hyper parameters used in Ada Boost regression. GridSearchCV does an exhaustive computation over all combinations for Ada BooSt, so we firdst used RandomSearchCV to get the range of parameters whcich gives the highest accuracy and then we used GridSearchCV to get the best out of it.
- Ada Boost was consistently working well for the best parameters in the 10 fold validation we applied in the local.

We found the AdaBoost Regression model returns the predicted regression values for the data set. These values are in the range of 0 to 1. So we had 0.5 as a threshold. The values greater than 0.5 we classified them as 1 and rest as 0

The F1 Score obtained as follows:

HYPER PARAMETERS	F1 SCORE
AdaBoostRegressor(n_estimators =500,loss='linear',learning_rate=0.4)	0.86219
AdaBoostRegressor(n_estimators =500,loss='linear',learning_rate=0.4)	0.83448
AdaBoostRegressor(n_estimators =1000,loss='exponential',learning_rate=1):	0.87364
AdaBoostRegressor(n_estimators =1300,loss='exponential',learning_rate=1)	**0.87455**

This last submission provided our best result and the values are always consistent in the range of 0.86 to 0.87.

BEST HYPER PARAMETERS:

PARAMETERS	VALUE
n_estimators	**1300**
loss	**exponential**
Learning_rate	**1**

We found out the predictions provided by it in csv file vary a little bit when we run again and hence f1 score changes a bit each time for same parameters.

To reduce the variation in the predictions, we thought of applying the ensemble machine learning we learnt above.

We ran the algorithm three times for the parameters AdaBoostRegressor(n_estimators =1300,loss='exponential',learning_rate=1) twice and AdaBoostRegressor(n_estimators =1300,loss='square',learning_rate=1) once
From those three we took majority voting out of it, so that the predictions can be consistent. The values came in range of 0.87 above.

KAGGLE CONCLUSION

- **We Chose AdaBoost as our primary algorithm**
- The main reason is that the predictions were highly consistent even when we tested for different combinations of hyper parameters obtained from **GridSearchCV.**
- The efforts towards improvement in this code are:
  1) Normalisation of data

  2) Searched hyper parameters using GridSearchCV

  3) Applying 10 fold cross validation on local to first check the accuracy on training data.

  4) **Ensemble Machine learning on the AdaBoost itself to get the best results.**

CONCLUSION:

- We implemented KNN, Naïve Bayes, Decision Tree, Random Forest on our own.
- We put our efforts in Kaggle competition to improve the code.

REFERENCES:

1) https://machinelearningmastery.com/naive-bayes-for-machine-learning/
2) https://medium.com/datadriveninvestor/k-nearest-neighbors-knn-7b4bd0128da7
3) https://medium.com/deep-math-machine-learning-ai/chapter-4-decision-trees-algorithms-b93975f7a1f1
4) https://medium.com/@rishabhjain_22692/decision-trees-it-begins-here-93ff54ef134
5) https://towardsdatascience.com/understanding-random-forest-58381e0602d2
6) https://towardsdatascience.com/accuracy-recall-precision-f-score-specificity-which-to-optimize-on-867d3f11124