

# CSE 601 Data Mining Project

# **Classification**

# **Algorithms**

---

Anjana Guruprasad (anjanagu - 50205233)

Shreya Ravi Hegde (shegde3 - 50208485)

## Introduction

- To implement three classification algorithms namely K nearest neighbours, naive bayes and decision trees.
- To implement Random forests and boosting on the implementation of decision trees.
- To use 10-fold cross validation to evaluate the performance of the classification algorithms in terms of accuracy, precision, recall and F1 - measure.

## Classification

- Classification is a form of supervised learning.
- In classification, the training data is accompanied by labels indicating the class of the observation.
- The goal of classification is to correctly classify new data based on the training set.
- i.e, classification works in two steps :
  - Building the Classifier or Model
  - Using the classifier for classification

## K Nearest Neighbours

- kNN is a non parametric method used for classification and regression.
- In kNN classification, an object is classified based on a majority vote i.e, the most common class label among all of its k (a typically small number)nearest neighbours is assigned to it.
- kNN classifiers are lazy learners whereas decision trees are eager learners.
- The function is approximated locally and all computation is deferred till classification.

## Implementation Details

- We read our data from the file given (project3\_dataset1.txt or project3\_dataset2.txt) and put it into a numpy array.
- In case the files contain categorical attributes, we enumerate over them and replace it with unique integers, where each integer represents a categorical value.
- Next we normalise our data in the numpy array using z-score normalisation. The formula for this is as shown below:

$$Normalized(e_i) = \frac{e_i - \bar{E}}{std(E)}$$

- We mainly normalize the data to make scores on the variables comparable and thus ease estimation.
- We perform 10 fold cross validation on our data set. One round of cross validation involves partitioning the data into subsets, where analysis is performed on one set of subsets (training set) and validate analysis on the rest (testing set). We perform this for multiple rounds with different partitions to reduce variability.
- For each record in the testing set we find k records or data points in the training set that are the closest to it. These points can be considered the record's neighbours.
- The majority class label these neighbours correspond to is then assigned to the test record. This is how prediction is done.
- We use Euclidean distance to measure the distance and identify the neighbours.
- The value of k can be varied till we get maximum accuracy.
- Once the predicted labels are found we compare it with the true labels and calculate various measures.
- Finally once all the rounds of cross validation are complete, we combine the testing results and estimate our final accuracy, recall, precision and f-measure.

The formulae used are :

Accuracy =  $\frac{tp + tn}{tp + tn + fp + fn}$

Precision =  $\frac{tp}{tp + fp}$

Recall =  $\frac{tp}{tp + fn}$

F-measure =  $\frac{2*tp}{2*tp + fn + fp}$

Where tp is true positives, tn is true negatives, fp is false positives and fn is false negatives.

## Parameter values chosen :

**K** : This signifies the number of neighbours we associated with each test data point. Based on the labels of the k closest points, we pick the majority label and assign it to the test point. It chosen in such a way that the accuracy is the highest.

**Distance** : We use Euclidean distance to calculate the distance between two data points.

**Categorical and Continuous** : Categorical values are replaced with a matching unique integer value for each categorical value. All the values are normalised using z score normalization which subtracts each value from the mean of all the attributes divided by the standard deviation of all the attributes. Hence both categorical and continuous values are normalized.

## Results

K Nearest Neighbours					
Dataset Used	K value	Accuracy	Precision	Recall	F - measure
project3_dataset1	3	0.96134	0.9786	0.9157	0.9441
	5	0.9666	0.9782	0.9280	0.9506
	10	0.9683	0.9897	0.9258	0.9550
	15	0.9648	0.9893	0.9188	0.9520
Project3_dataset2	9	0.6904	0.5925	0.3701	0.4392
	12	0.7012	0.6455	0.3001	0.3956
	13	0.7142	0.6610	0.3859	0.4740
	30	0.7165	0.6955	0.3601	0.4638

## Result Analysis:

- Overall we see that K nearest neighbours performs best for dataset1. This is because kNN is a distance based classifier.
- It is easier to compute distance for numerical data than for categorical data. As dataset1 contains only numeric values it performs better than dataset 2 (contains numeric and categorical data) when we use euclidean distance.

- Another reason could be that the dataset 2 is smaller in size. Hence there were not enough training examples to classify properly.
- We can also observe that  $k$  has impact on the overall accuracy. If  $k$  is too small, the classifier could be sensitive to outliers and if it is too large it can misclassify data.

## Pros of K nearest neighbours

- Simple to implement.
- K nearest neighbours can learn non-linear boundaries.
- Robust to noisy training data, more so if you use inverse square of weighted distance as the distance metric.
- Can do well in practice, if there is enough representative data.
- Naturally handles multi-class cases.

## Cons of K nearest neighbours

- Need to determine the value of  $k$ .
- K nearest neighbours predicts just the labels unlike naive bayes and logistic regression which predicts probabilities that are a measure of confidence.
- In distance based learning sometimes it is not clear which type of distance to use and whether to use all/some of the attributes.
- Sometimes if the attributes are not scaled properly, the algorithm may misclassify the data.
- Computation cost is quite high as we have to compute the distance of each sample to the entire training set.

## Decision Tree

- It is a supervised learning algorithm.
- It can be used for regression and classification problems also.
- It creates a training model that can predict the value of target variables or class labels by learning decision rules from prior data (Training Data).
- It gives us a tree where every internal node is a test condition and every leaf node is a label.
- It is used to represent and make decisions.

## Implementation Details

- We read our data from the file given (project3\_dataset1.txt or project3\_dataset2.txt) and put it into a numpy array.
- The last column is obtained separately and we identify the number of classes/labels present in the data.

- We perform 10 fold cross validation on our data set. One round of cross validation involves partitioning the data into subsets, where analysis is performed on one set of subsets (training set) and we validate analysis on the rest (testing set). We perform this for multiple rounds with different partitions to reduce variability.
- Here we need to create a binary tree for the given training data such that at each junction we have a test attribute condition based on which we produce two child nodes from the parent such that the data in the parent node is split between the child nodes based on this condition.
- In this case we are using Gini index as our cost function. The Gini index is used to decide the test attribute condition. We go through all the attributes (columns) in the dataset and for each value in a particular attribute we calculate its Gini index.
- A Gini index indicates how good a split is i.e the split of data between the child nodes. The less mixed the data in each node, the better is our split and lesser is our Gini index.
- So the value of an attribute with the minimum Gini index is chosen as the root test attribute condition of the tree.
- This process is done recursively for each split in data, where the next attribute with the lowest Gini index is picked to split the data.
- Finally we let the tree build until all the data in the node has the identical class label. This is termed a leaf node.
- Once the tree is built with the training data, we pass the test data down the tree to reach a leaf node and the majority class label there will be assigned to this test data. This is one round of cross validation.
- This is further done for 10 folds, by picking different partitions for training and testing data each time.

## Feature handling

**Categorical features** : If the values of the categorical features match with the attribute value with the lowest Gini index picked, place in the left subtree. If they don't match or are not equal then it goes into the right subtree.

**Continuous features** : The records are split based on the value of the attribute picked as a test attribute condition due to its low Gini index. If the feature value is less than the value picked, place it in the left subtree or left child. If the feature value is greater than the value picked, place it in the right child.

**Best feature** : The feature with the minimum Gini Index is the best feature chosen to split the data. Since it has the least Gini Index, it gives purer splits i.e less mixed.

**Post Processing** : We did not use further conditions to stop or prune the tree. We let the tree grow to its full size. Post processing could be done by pruning the tree if a node has less than a certain number of records. We can also add a maximum depth parameter where the tree stops once it reaches that depth level.

## Results

Decision Tree				
Dataset Used	Accuracy	Precision	Recall	F1 - measure
project3_dataset1	0.9243	0.9147	0.8853	0.8979
project3_dataset2	0.6489	0.4962	0.503	0.4915

## Result analysis

- Decision tree gives better accuracy for dataset 1 than dataset 2. A possible reason for this could be that the size of dataset 2 is small and hence the decision tree formed hasn't encountered all possible test cases and is not expressive enough.
- For dataset 1 and dataset 2, kNN and naive bayes perform better than decision tree. This could possibly occur because the data is mostly uncorrelated. Naive bayes performs better when data is uncorrelated whereas decision tree does not.
- It could also be because the tree is overfitting the data as there are no pruning condition such as maximum depth and minimum size.

## Pros of decision tree

- Ability to select the most discriminating features.
- Nonlinear relationships between parameters do not affect the tree's performance.
- The number of hyper parameters to be tuned is almost null.
- Simple to understand, interpret and visualize.
- Can handle both numerical and categorical data. It can also handle multi-output problems.

## Cons of decision tree

- Each split in the tree leads to a reduced dataset under consideration. Hence the model created at the split could potentially introduce bias (Data Fragmentation).
- It does not work very well if there is a lot of uncorrelated data.
- It does not work well if you have smooth boundaries.
- They are highly sensitive in nature - small changes in data can result in a drastically different tree.
- Exponential calculation growth when the problem gets bigger. There is also high probability of overfitting the data.

## Random Forests

- It is an ensemble learning method used for classification, regression and other tasks.
- A random forest classifier consists of multiple trees designed to increase the classification rate.
- To classify a new object from an input vector, put the input vector down each of the trees in the forest.
- Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

## Implementation Details

- We read our data from the file given (project3\_dataset1.txt or project3\_dataset2.txt) and put it into a numpy array.
- The last column is obtained separately and we identify the number of classes/labels present in the data.
- Similar to decision trees, we perform 10 fold cross validation and achieve total accuracy.
- We also create the tree in the same way based on Gini index to ensure the cost remains minimal, ie pick the value of the attribute with the least Gini index.
- The difference here is we don't check all the attributes in the data. We choose m random attributes where  $m < \text{total attributes in data}$ . The attribute with the lowest Gini index from this set of m random attributes is picked first for the root.
- This process is done recursively for each split in data, where the next attribute with the lowest Gini index is picked to split the data.
- At each step of the recursion, m random attributes are selected to build the tree.
- Once the tree reaches its termination and the leaf nodes are formed, we pass the test data to this tree which classifies it and assigns it the predicted label.

## Results

Random Forests				
Dataset Used	Accuracy	Precision	Recall	F1 - measure
project3_dataset1	0.9454	0.9276	0.9219	0.9030
project3_dataset2	0.6623	0.5196	0.4633	0.4530

## Result Analysis

- It performs better than decision tree for both dataset 1 and dataset 2. This could be cause the overfitting problem is eliminated when we use random forests.
- It could also be giving a better accuracy because it can handle high dimensional space well.



## Pros of random forests

- It runs effectively on large databases.
- The overfitting problem can never occur in random forests for a classification problem.
- Because of the way they are constructed, the algorithm can handle high dimensional spaces as well as large number of training examples very well.
- They do not expect linear feature or even features that interact linearly.
- It can be used for feature engineering.
- Does well with uneven data that has missing values.

## Cons of random forests

- It is hard to comprehend the result. It is hard to gain a lot of insights (Hard to visualize).
- It is complex and computationally expensive.
- It assumes that model errors are uncorrelated and uniform .
- Misclassifications are hard to backtrack.
- Difficult to implement

## Boosting

- It is a supervised learning method.
- Boosting is one of several classic methods for creating ensemble models.
- Boosting means that each tree is dependent on prior trees, and learns by fitting the residual of the trees that preceded it.
- Boosting in a decision tree ensemble tends to improve accuracy with some small risk of less coverage.

## Implementation Details

- We read our data from the file given (project3\_dataset1.txt or project3\_dataset2.txt) and put it into a numpy array.
- The last column is obtained separately and we identify the number of classes/labels present in the data.
- Similar to decision trees, we perform 10 fold cross validation and achieve total accuracy and other performance parameters.
- We also create the tree in the same way based on Gini index to ensure the cost remains minimal, ie pick the value of the attribute with the least Gini index.
- Unlike random forest, here we select weighted attributes to generate the tree from the training set.
- Initially we maintain a weight vector in which the initial weights for all the records in the training set is set to 1/total samples i.e uniform for all the records for the first time.
- We build the binary tree based on this training set by randomly picking attributes with replacement but with the influence of the weights, hence it is a weighted random choice of the attributes.
- Based on the predicted labels and the true labels of the training data all the records predicted correctly get a lower weightage and the weak ones or the wrongly predicted records are given more weightage for the next round.

- The weights are updated based on the calculation of error in misclassified data:

$$\varepsilon_i = \frac{\sum_{j=1}^N w_j \delta(C_i(x_j) \neq y_j)}{\sum_{j=1}^N w_j}$$

- The importance of the classifier is given by :

$$\alpha_i = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_i}{\varepsilon_i} \right)$$

- And the weight of each record based on the above parameters is calculated as follows :

$$w_j^{(i+1)} = \frac{w_j^{(i)} \exp(-\alpha_i y_j C_i(x_j))}{Z^{(i)}}$$

- Once the tree is built, we pass our testing set to the tree which then assigns it its predicted label.
- We repeat the above steps for the number of forests we want to create.
- Finally we multiply the predicted label vector with the importance of each classifier. Based on this combined value we decide the final class labels.

$$C^*(x) = \arg \max_y \sum_{i=1}^K \alpha_i \delta(C_i(x) = y)$$

- This entire procedure runs for 10 fold of cross validation.

## Results

Boosting				
Dataset Used	Accuracy	Precision	Recall	F1 - measure
project3_dataset1	0.9209	0.8871	0.9031	0.8937
project3_dataset2	0.6514	0.5078	0.4207	0.4530

## Result analysis

- It performs better than decision tree for dataset 2. This could be due to the fact that while creating new tree the errors produced in the previous tree are taken into account , i.e. weighted sampling is done.
- It produces almost the same results as decision tree for dataset 1. This could be due to the presence of multiple hyperparameters leading to overfitting by the model.

## Pros of boosting

- Accuracy is improved.
- Don't need to apply feature scaling for the algorithm to do well.
- It gives better results because each new tree tries to remove the errors created by the previously trained tree.
- Training can be parallelized.
- Works efficiently on large databases when run in parallel.

## Cons of boosting

- Time and computationally expensive. Complexity of the classification increases.
- Harder to tune.
- Results are hard to interpret and it is not robust when the data is noisy.
- Because of the presence of multiple hyperparameters, it can easily overfit.
- Hard to implement in real time platform.

## Naive Bayes

- It is a statistical classifier.
- Unlike decision trees that predict class label, Naive bayes tries to predict the class membership
- It is based on the Bayes theorem (posterior probability).
- It functions on the assumption that the attributes in the data set are independent to each other.
- It is computed using this formula:

$$P(H_i | X) = \frac{P(H_i) P(X | H_i)}{P(X)}$$

Where  $P(X)$  - descriptor prior probability

$P(H_i)$  - class prior probability

$P(X/H_i)$  - Descriptor posterior probability

$P(H_i/X)$  - Class posterior probability

## Implementation Details

- We read our data from the file given (project3\_dataset1.txt or project3\_dataset2.txt) and put it into a numpy array.
- The last column is obtained separately and we identify the number of classes/labels present in the data.
- We perform 10 fold cross validation on our data set. One round of cross validation involves partitioning the data into subsets, where analysis is performed on one set of subsets (training set) and we validate analysis on the rest (testing set). We perform this for multiple rounds with different partitions to reduce variability.
- We first calculate the class prior probability  $P(H_i)$  for both classes 0 and 1. We compute it by dividing the number of samples belonging to a particular class by the total number of samples.
- In the dataset we encounter two types of attributes - continuous and categorical. We handle both the types in the same way.
- For the descriptor posterior probability  $P(X/H_i)$ , We compute the frequency of an attribute value of the sample in the total dataset and divide it by the number of samples in the dataset.
- For the descriptor prior probability  $P(X)$ , we compute the frequency of an attribute value of a sample in a particular class and divide it by the number of samples in that class.
- The probability for each class is then computed by using the formula,

$$P(H_i | X) = \frac{P(H_i) P(X | H_i)}{P(X)}$$

- The class having the higher probability is chosen as the label for that sample.
- Once the predicted labels are found we compare it with the true labels and calculate various measures.
- Finally once all the rounds of cross validation are complete, we combine the testing results and estimate our final accuracy, recall, precision and f-measure.

## Feature handling & Zero probability case:

- We have considered each numeric data to be an entry of its own. So, we have computed it the same way that we would compute a categorical attribute. We count the number of occurrences of different values for that particular attribute.
- The descriptor probability goes to 0 if any of the probabilities is 0, this problem is called the zero probability problem.
- It can be rectified using Laplacian correction/ estimator. In laplacian correction, We add 1 to each case so that the descriptor probability doesn't go to 0.

## Results

Naive Bayes				
Dataset Used	Accuracy	Precision	Recall	F1 - measure
project3_dataset1	93.4899749373%	91.7870936253%	90.4442635683%	91.0031838126%
project3_dataset2	70.3469010176%	57.0927564008%	61.593964515%	58.6739645814%

## Result Analysis:

- We can see that after kNN and naive bayes perform the best for dataset2 overall.
- This could be cause naive bayes predicts probabilities that are a measure of confidence. Hence the classification rate is better than those of the decision trees.
- For dataset 1, kNN gives better accuracy than naive bayes. Naive bayes works on the assumption that the attributes are independent to each other. A possible reason for the lower accuracy could be that the attributes in dataset 1 are not independent. It could also be due to the continuous nature of the attributes.


## Pros of naive bayes

- It is simple - easy to understand and implement.
- Small memory footprint and doesn't require complex optimization.
- A Naive Bayes classifier will converge quicker than discriminative models like logistic regression, so you need less training data.
- Can be applied to large datasets.
- Comparable in performance to decision trees.

## Cons of naive bayes

- Works on attribute independence - But this is seldom satisfied in real life, as attributes are often correlated.
- Descriptor posterior probability could go to 0 - this needs to be handled after analysing the dataset.
- It is hard to compute probabilities if the features are continuous in nature - might have to do binning or use a gaussian distribution.

## Overall result analysis

- 
- For dataset 1, kNN performs the best. This could be cause all the attributes are continuous in nature and kNN handles it really well.
  - For dataset 2, Naive bayes performs the best. This could be cause it has categorical data which is handled well by the naive bayes classifier.

## .References

- Lecture Slides
- Wikipedia
- Stack Overflow

—