PROJECT 3 REPORT

CLASSIFICATION

TEAM 23
PRASHANTH CHANDRASHEKHER(pc74)
VIPIN KUMAR(vkumar25)
ARUN SHARMA(arunshar)

K Nearest Neighbors:

Algorithm Flow:

1) For every test record, measure the distance to every record in the training set.
    a. This is done by first splitting the data into continuous and categorical attributes
    b. The continuous attributes are normalized and we get a Euclidean distance between the normalized values of the continuous attributes
    c. We calculate the Hamming distance for the categorical data types which gives a measure of dissimilarity between the categorical vectors
    d. We add up the distance matrices we get for continuous and categorical datatypes
2) Identify the k nearest neighbors of every test record after sorting the distances
    a. Once we get the distances from previous step, we use argsort to get the indices of the data points to be sorted in increasing order, and by choosing the top k of these helps us to get the k nearest neighbors of the test data
3) Use the class labels of the nearest neighbors to predict the label of the test data
    a. We take the maximum of the count of classes to which the k closest neighbors belong to, and then the class with maximum votes is the predicted class for the particular test data.

Choice Description:

k: if the k value is very small, the classification is affected by noise. If it is very large, the neighborhood may include points from other classes as well. An even value of k is also not usually desired because there is a possibility that the k-nearest neighbors can have exactly k/2 number of examples that are labeled oppositely and either class can be chosen as the maximum of the nearest neighbors.

We varied the k from 3 up to len(dataset)/2 in increments of 2, and plotted the graph of the change in Accuracy, Precision, Recall and the FMeasure. We choose k such that we get the ideal balance of the metric or as required by the application. We ended up with selecting k=9 for dataset #1 and k=39 for dataset #2 to get good accuracy values.

Continuous: the attributes with continuous data are identified and we normalize all these columns in the training data so that each of them contribute equally towards the distance that will be measured. In order to normalize these values, we take the mean and standard deviation of the individual columns and we remove the mean from all the elements of that column and we then divide the column with the standard deviation. Now each attribute is normalized and all of them contribute the same when measuring the distance.

For the new test data, we use the same means and standard deviations we calculated for each of the attribute on the training data and use the same to normalize the continuous attributes of the test data.

Categorical: We identify the columns which have categorical datatypes by checking if the value represents a digit or not. Once an attribute is marked as categorical, the measurement of distance between such attributes is taken care of differently compared to continuous data types. We will have to find a way to translate the distance metric we chose to the actual nearness of neighboring categorical data. Measurement of the distance is given in the next section.

Distance: We measure distance of continuous and nominal attributes differently. For continuous datatypes, after normalizing the attributes, we take the Euclidean distance of the test and the training data. For nominal attributes, we get the dissimilarity (which corresponds to distance between those data) by using hamming distance. Hamming distance of nominal attributes is the sum of the number of dissimilarities we have under each attribute under the data being compared. Because our continuous attributes were normalized earlier, it does make sense to combine the Euclidean distance of the continuous attributes and the Hamming distance of the nominal attributes directly by adding them to each other.

So, the total distance between 2 data points will be the sum of Euclidean distance of the normalized continuous attributes and the Hamming distance of the nominal attributes

Result Analysis:

Performance:

The following are the value which are generated by taking mean of results generated from 10 fold cross validation on the following datasets.

Dataset 1

	Accuracy	Precision	Recall	F-measure
K = 3	0.964912280702	0.972613871636	0.933585643952	0.951724849177
K = 5	0.966666666667	0.982375776398	0.929313575441	0.95359212201
K = 7	0.970175438596	0.986723602484	0.934576733335	0.958570578728
K = 9	0.970175438596	0.986723602484	0.934576733335	0.958570578728

Dataset 2

	Accuracy	Precision	Recall	F-measure
K = 3	0.659574468085	0.510183150183	0.383739808953	0.428537036926
K = 5	0.659574468085	0.508717948718	0.359036605809	0.407433311023
K = 7	0.651063829787	0.492404262404	0.318177573201	0.376487516588
K = 9	0.670212765957	0.576634199134	0.267993492576	0.355430744414

Parameter settings: We changed the parameters of k values and check which one gives the maximum accuracy.

Pre-processing: We first normalized the dataset by taking mean and standard deviation of the training data. In test data we subtract the test sample with its corresponding attributes.

Avoiding Overfitting: In order to avoid overfitting, we randomly shuffle the data.

Pros:

- Allows us to choose our own distance functions for each attribute independently and hence we can use it well for datasets which have mixed type of attributes.
- It's simple to implement and pretty flexible to modify quickly according to the given datatype
- Without a lot of changes, the algorithm can handle multiple classes as well and not just 2 classes.

Cons:

- Computationally expensive as we need to get distances for every test data with all the train data and then we need to parse through them and then find the minimum ones in order to identify the nearest neighbors
- Because we need to store the various distance matrices all in memory in order to sort and retrieve the nearest neighbors, we need a lot of storage as the size of the data grows.
- We must have proper distance functions define in order to get the correct results since we may be dealing with mixed types of attributes in the datasets.

Cross Validation:

We are performing a 10-fold cross validation in order to get a good estimate of the accuracy measures of the algorithm. Below is the way this is implemented.

We first shuffle the dataset and then identify the size of the test data set required for a 10-fold validation.

We then extract this test set and move the remaining elements into another matrix and call them as train set.

We perform regular knn algorithm on this test and train dataset pairs and compute the performance metrics.

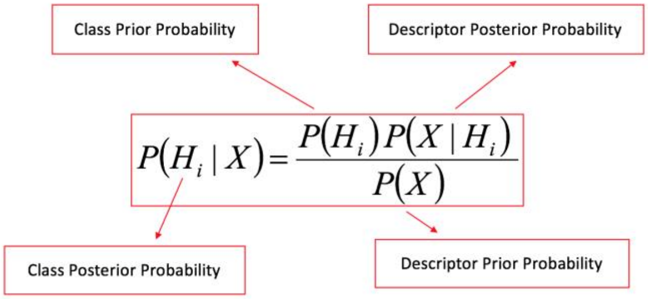
We next go on to the next fold, i.e, the next set of test data set elements are extracted and now the remaining data is the train data set, and knn is performed on this.

The process is repeated 10 times until different segments of the dataset is considered as test dataset and performance metrics are evaluated for each of the folds.

Finally, we calculate the average performance metric across the 10-folds.

Naïve Bayes:

Naïve Bayes Algorithm is based on Bayes theorem involving prior and posterior probabilities
The equation below is taken from the sldes:


Class Prior Probability
Descriptor Posterior Probability
$P(H_i \mid X) = \frac{P(H_i)P(X \mid H_i)}{P(X)}$
Class Posterior Probability
Descriptor Prior Probability

<u>Algorithm Flow:</u>

1) Check whether the data contains how many categorical and continuous features and extract the columns for handling them separately.
2) Calculate number of samples in the respective classes into a hashmaps taking key as classes and values as the sample rows.
3) For continuous data segment calculate column-wise mean and standard deviation in order to get Gaussian distribution. Similar for categorical calculate prior and posterior probability.
4) In testing phase, for each sample calculate conditional probability for each class and multiply continuous and categorical probability for getting total class-wise probability.
5) Check which class has higher probability which in turn will be class of that sample.

Choice Description:

<u>Continuous feature:</u> In order to check the continuous, we first convert each value to float, if it gives the value error then the feature is categorical. In continuous, we first calculate column-wise mean and standard deviation of all the samples and use it for Gaussian probability for test data. In categorical data, we found discrete probability separately using count on posterior and prior probability of a particular test sample.
<u>Zero probability</u>: Here, if the posterior probability is zero then we use Laplacian correction, but we didn’t code in our system.

<u>Result Analysis:</u>
<u>Performance:</u>
The following are the value which are generated by taking mean of results generated from 10-fold cross validation on the following datasets.

Dataset: 1
Accuracy is: 93.5124135124
Precision is: 91.8032698649
Recall is: 90.8213536329
Fmeasure is: 91.2754370051

Dataset: 2
Accuracy is: 69.5231178891
Precision is: 53.5245681284

Recall is: 67.9467633234
Fmeasure is: 59.6336948148

Pros and Cons:

Pros:

- Easy to implement since each label is independent to each other.
- Faster and less complex than other classification algorithms

Cons

- Chances of facing zero posterior probability issues is high in case of categorical data
- Not a good choice if attributes of the data is highly dependent on each other.

Pre-processing:

The data is first separated into continuous and categorical so that both of them can be handled separately and effectively.

Cross Validation:

We are performing a 10-fold cross validation in order to get a good estimate of the accuracy measures of the algorithm. Below is the way this is implemented.

We first shuffle the dataset and then identify the size of the test data set required for a 10-fold validation.

We then extract this test set and move the remaining elements into another matrix and call them as train set.

We perform regular algorithm on this test and train dataset pairs and compute the performance metrics.

We next go on to the next fold, i.e, the next set of test data set elements are extracted and now the remaining data is the train data set, and algo is performed on this.

The process is repeated 10 times until different segments of the dataset is considered as test dataset and performance metrics are evaluated for each of the folds.

Finally, we calculate the average performance metric across the 10-folds.

Decision Tree

Algorithm Flow:

1) Create a decision tree on the basis of training data using best node splits.
2) Check for the best nodes split by pruning left and right according to the given depth of the tree
3) For the best split we used gini index which is used to assigning the classes.
4) For test cases, pass the test sample from the tree and simply prune the tree from it's root.
5) Assign the class and evaluate accuracy and other evaluation measures.

Choice Description:

Continuous features: In order to check the continuous, we first convert each value to float, if it gives the value error then the feature is categorical.
Categorical features: Since it is a decision tree, the categorical data can be used as a value like any other value. For test cases we need to detect the continuous and categorical features in order to prune the trees.
Best Feature: For the best feature, we used gini index which is the evaluation metric used to give the best split in the decision tree. The lower gini index, the better is the split and therefore those features have lowest gini values (i.e. close to 0). Hence they are best features.
Post-processing: Results are evaluated through cross –validation, other than that no post-processing is done.

Results Analysis:
Performance:
Dataset 1:
Mean Precision: 93.510%
Mean Recall: 86.818%
Mean Accuracy: 93.133%
Mean F1: 89.867%

Dataset 2:
Mean Precision: 59.300%
Mean Recall: 42.632%
Mean Accuracy: 69.468%
Mean F1: 48.246%

Parameter setting: For simplicity, we take minimum size of tree be 1 and depth of tree be 3. Other than that, we take cross validation of 10-fold cross-validation.

Pros:

- No need to handle categorical data separately
- Give comparable accuracy with other classification algorithms
- Easily combine with other ensemble techniques.

Cons:

- May become complex when specifying large parameter values on large datasets
- Slower than other algorithms

Avoiding Overfitting: random selections of test samples on every cross-validation step.

Cross Validation:

We are performing a 10-fold cross validation in order to get a good estimate of the accuracy measures of the algorithm. Below is the way this is implemented.

We first shuffle the dataset and then identify the size of the test data set required for a 10-fold validation.

We then extract this test set and move the remaining elements into another matrix and call them as train set.

We perform regular algorithm on this test and train dataset pairs and compute the performance metrics.

We next go on to the next fold, i.e, the next set of test data set elements are extracted and now the remaining data is the train data set, and algo is performed on this.

The process is repeated 10 times until different segments of the dataset is considered as test dataset and performance metrics are evaluated for each of the folds.

Finally, we calculate the average performance metric across the 10-folds.

Random Forrest:

It is based on bagging the samples and make decision trees based on the selected features. The Algorithm flow is given below:

Algorithm Flow:

1) Get the input parameters as number of trees needed and selecting random features from the tree.
2) Create a decision tree on the basis of training data by selecting random rows with replacement (bagging).
3) Check for the best nodes split by pruning left and right according to the given depth of the tree
4) For the best split we used gini index which is used to assigning the classes.
5) For test cases, pass the test sample from the tree and simply prune the tree from it's root.
6) Assign the class and evaluate accuracy and other evaluation measures.
7) Repeat all the steps from 2 if the trees are more than one and aggregate the accuracy.

Results Analysis:

Performance:

Dataset 1:
For 1 tree:
Mean Precision: 85.601%
Mean Recall: 87.228%
Mean Accuracy: 88.734%
Mean F1: 85.685%

For 3 trees:
Mean Precision: 89.945%
Mean Recall: 82.713%
Mean Accuracy: 89.972%
Mean F1: 85.983%

For 10 trees:
Mean Precision: 94.364%
Mean Recall: 83.032%

Mean Accuracy: 91.911%
Mean F1: 88.186%

Dataset2:
For 1 tree:
Mean Precision: 48.335%
Mean Recall: 49.611%
Mean Accuracy: 63.848%
Mean F1: 46.852%

For 3 trees:
Mean Precision: 59.704%
Mean Recall: 41.074%
Mean Accuracy: 69.704%
Mean F1: 46.927%

For 5 trees:
Mean Precision: 56.810%
Mean Recall: 26.989%
Mean Accuracy: 65.365%
Mean F1: 35.057%

Parameter Settings: Change the number of random features to pick from the dataset. Also changing maximum depth and min values. For simplicity we check the max depth as 7 and min value as 1.

Pros:

- Much faster than Decision Tree
- Gives better accuracy than Decision Tree

Cons

- May have some inaccuracies in the results
- Results may also vary for every iteration

Avoid Overfitting: No need to take of overfitting since when it is selecting random rows

Cross Validation:

We are performing a 10-fold cross validation in order to get a good estimate of the accuracy measures of the algorithm. Below is the way this is implemented.

We first shuffle the dataset and then identify the size of the test data set required for a 10-fold validation.

We then extract this test set and move the remaining elements into another matrix and call them as train set.

We perform regular knn algorithm on this test and train dataset pairs and compute the performance metrics.

We next go on to the next fold, i.e, the next set of test data set elements are extracted and now the remaining data is the train data set, and knn is performed on this.

The process is repeated 10 times until different segments of the dataset is considered as test dataset and performance metrics are evaluated for each of the folds.

Finally, we calculate the average performance metric across the 10-folds.

Boosting:

It is used to give more weights on weak samples i.e. which are misclassified. The boosting technique used is adaBoost which is described below.

Algorithm flow:

1) Take the training set and weight them equally i.e. 1/number of training data
2) Randomly choose weights (bagging) and check the misclassified samples and weight them more than the correctly classified samples.
3) Calculate and save the staging values alpha and trees for further prediction of test labels.
4) Update the weights by using the given alpha and repeat form step 2 for given number of iterations.
5) In testing phase, for every test samples, get the prediction label using saved trees and corresponding to the trees assign alpha for that label by adding it.
6) Check the maximum value of alpha aggregated in the label which will be the class of the test sample.

Results Analysis
Performance:

Mean accuracy is: 92.2912087912
Mean precision is: 94.29592499
Mean recall is: 84.0770895771
Mean fmeasure is: 88.7778781577

Preprocessing: In order to check the continuous, we first convert each value to float, if it gives the value error then the feature is categorical Only data is first converted into list for ease of getting cross validation.

Pros:

- Faster than Decision tree with reasonable accuracy
- Reduce the model bias hence achieve greater accuracy.

Cons

- May leads to overfitting if more boosting is applied
- May take more time than usual because to select appropriate weights.

Avoid Overfitting: Since boosting reduce bias, it is done for limited number of iteration i.e. only 5 iterations.

Cross Validation:

We are performing a 10-fold cross validation in order to get a good estimate of the accuracy measures of the algorithm. Below is the way this is implemented.

We first shuffle the dataset and then identify the size of the test data set required for a 10-fold validation.

We then extract this test set and move the remaining elements into another matrix and call them as train set.

We perform regular algorithm on this test and train dataset pairs and compute the performance metrics.

We next go on to the next fold, i.e, the next set of test data set elements are extracted and now the remaining data is the train data set, and algo is performed on this.

The process is repeated 10 times until different segments of the dataset is considered as test dataset and performance metrics are evaluated for each of the folds.

Finally, we calculate the average performance metric across the 10-folds.