CSE 601

Project 3

Classification Algorithms

Submitted by:

Sampreeth Reddy B (sboddire/50298591)

Paritosh Kumar Velalam(pvelalam/50295537)

K-Nearest Neighbors:

Description:

Use “k” nearest neighbors of the data point in question and classify them to the respective class
to which maximum of these neighbors belong to.
1. K-nn is a non-parametric and lazy-learning algorithm that is used to predict classification, based on feature similarity.
2. There is no explicit training phase, since it does not use the training data to do any generalization, like most of the other classification algorithms.
3. For the same reason, it requires most of the training data when it needs to actually predict the class/label during the testing phase.

Algorithm

1. Let k be the number of nearest neighbors and D be the set of training examples.
2. **for** each test example **z=(x′,y′)do**
3. Compute **d(x′,x),** distance between z and every example, (x,y) ε D.
4. Select Dz ⊆ D, set of k closest training examples to z.
5. $y' = argmax\ v \sum_{(xi, yi) \in Dz} I(v = yi)$
6. **end for**

Our Implementation:

We have Implemented Naïve Bayes classifier in Python. Our Program performs 10-fold cross validation by choosing different training data and testing data in every iteration and outputs the average accuracy, precision, recall and F-Measure after the 10 iterations.
1. Read the user input and assign it to K, Where K is the number of nearest neighbors of unclassified label to compare and assign.
2. Read the data from the file and divide it into training (90%) and test (10%) datasets.
3. Take a data set from testing data and calculate the Euclidean distance from all the points in training data to this point.
4. Pick top K closest neighbors and check the labels of neighbors.
5. Assign the label which occurs in maximum number of time in our training dataset.

6. Repeat the same process for all the values in test data.

7. calculate the performance of model by comparing the assigned to original label of test data.

8. The same process is iterated for 10 iterations and average accuracy is calculated as average of accuracies in all iterations.

Results on project3_dataset1.txt:

K Val	Avg Accuracy	Avg Precision	Avg Recall	Avg F-Measure
3	0.927	0.924	0.872	0.896
5	0.927	0.919	0.877	.894
7	0.927	0.919	0.881	0.896
9	0.934	0.939	0.882	0.9077
12	0.931	0.938	0.8706	0.901
15	0.927	0.944	0.85	0.899
20	0.927	0.94	0.85	0.89

Results on project3_dataset2.txt:

K Val	Avg Accuracy	Avg Precision	Avg Recall	Avg F-Measure
3	0.58	0.39	0.33	0.345
5	0.61	0.44	0.28	0.331
7	0.64	0.48	0.32	0.38
9	0.657	0.51	0.33	0.396
12	0.642	0.46	0.27	0.335
15	0.6581	0.53	0.323	0.3826
20	0.673	0.587	0.288	0.3704

Important Insights:

- It is important to choose the value of k carefully, because if k is too small it can make the
- prediction sensitive to noise points.
- If k is too large it can include irrelevant points from other classes that may lead to an erroneous classification eventually.
- Our implementation handles the categorical data by checking whether given feature is float or not
- We have used Euclidean distance to calculate the nearest points
- Algorithm provides very high accuracy on dataset1 when compared to dataset2 because dataset1 contains only continuous data but dataset2 contains both continuous and Categorical data. This implies that K-NN does not perform well on categorical data or the data which contains both categorical and continuous data.

- Results show that performance of algorithm is low when the k value is too low or too high. So k value should never be too high or too low.

Pros

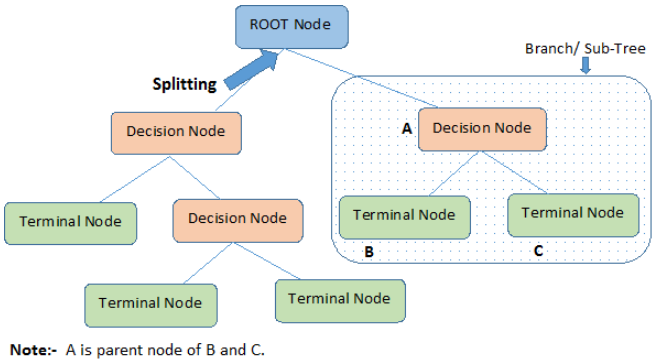
1. Simple algorithm. Easy to implement.
2. The running time is fairly low compared to many other classification algorithms.
3. K-NN can be useful in both the predictive problem cases - Classification and Regression
but majorly used for Classification in the industry.
4. Insensitive to outliers in general.

Cons

1. K-NN algorithm requires attribute scaling to avoid any overfitting towards one of the attributes.
2. Choosing the value K is difficult.
3. Storing of all the training data can cause a bottleneck leading to this algorithm being computationally expensive and with a higher memory requirement.
4. For really big datasets the running time for predicting is slow with better supervised learning algorithm running much faster in comparison.

Decision Tree:

A decision tree is a graph that uses a branching method to illustrate every possible outcome of a decision. Each internal node represents a 'test' on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label. The paths from root to leaf represent classification rules.



Our Implementation:

1. The entire dataset is split into training and testing sets using 10-fold validation.
2. Training set is used to train the model to construct a decision tree.
3. Records based on all values of a column are taken. (Each column represents an attribute).
4.Gini Index is calculated for the given split based on the below formula

$$GINI(t) = 1 - \sum_{j} [p(j \mid t)]^2$$

5.Information gain is measured for the best split by using below formula

$$GAIN_{split} = Measure(p) - \left(\sum_{i=1}^{k} \frac{n_i}{n} Measure(i) \right)$$

6. The algorithm is performed on the column and value which provides minimum GINI score is taken as it provides the maximum gain.
7. The above steps are recursively called for both left and right child of the node.
8. At any stage, if all the records belong to the same class, then it is made a leaf node with a particular label for that class.

Results:

project3_dataset1.txt:

Avg Accuracy	Avg Precision	Avg Recall	Avg F-Measure
0.915	0.88	0.889	0.884

project3_dataset2.txt:

Avg Accuracy	Avg Precision	Avg Recall	Avg F-Measure
0.603	0.426	0.46	0.43

Pros

1. They are inexpensive to construct and extremely fast at classifying unknown records.
2. Decision trees when at a smaller-depth size are very easy and quick to interpret.
3. The decision tree induction is a non-parametric approach for building classification models meaning not requiring any assumptions regarding the type of probability distributions satisfied by the class and other attributes.
4. They do not get affected by presence of redundant attributes too.
5. Can handle both numerical and categorical data. Can also handle multi-output problems.

6. It implicitly performs feature selection.

7. Nonlinear relationships between parameters do not affect tree performance.

Cons

1. It can result in overfitting

2. This algorithm makes decision trees susceptible to high variance if they are not pruned.

4. Due to top-down approach, the number of records at leaf node may be too small to make

a statistically significant decision about the class representation of the node, also known as data fragmentation. The solution is setting a maximum depth limit.

5. Can be unstable because small variations in the data might result in a completely different tree being generated.

6. The boundaries are rectilinear.

7. This algorithm is repeatedly called for each subset until it cannot be split further.

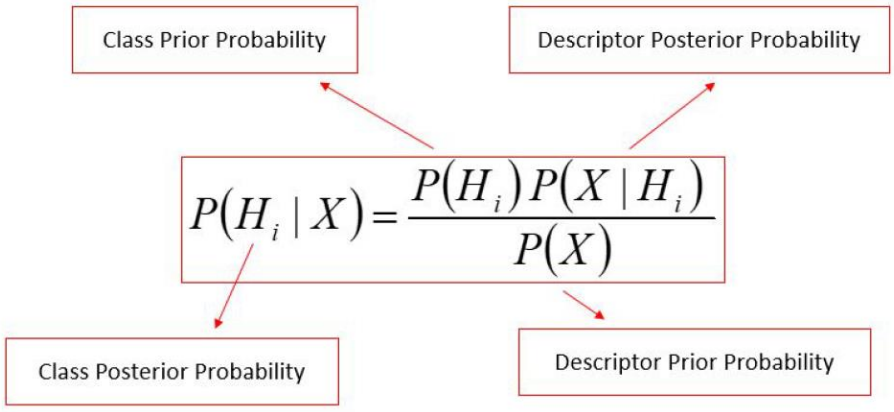
8. Finally, the accuracy from each iteration is taken to find the average accuracy for the whole dataset.

Naive Bayes Classification:

1. In many applications the relationship between the attribute set and the class variable is

non-deterministic because of noisy data or other certain confounding factors.

2. For such cases we resort to modeling probabilistic relationships between the attribute set and the class variable - based upon Bayes Theorem and Classifier



Algorithm

```
1. Let T = (X1, X2,... , Xd) denote a total order of the
   variables.
2. for j=1 to d do
3.    Let Xt(j) denote the jth highest order variable in T.
4.    Let π(Xt(j)) = {XT(1), XT(2), … , XT(j-1)} denote the
   set of variables preceding XT(j).
5.    Remove the variables from π(Xt(j)) that do not affect Xj
   (using prior knowledge).
6.    Create an arc between XT(j) and the remaining variables
   in π(Xt(j)).
7. end for
```

Our Implementation:

We have Implemented Naïve Bayes classifier in Python. Our Program performs 10-fold cross validation by choosing different training data and testing data in every iteration and outputs the average accuracy, precision, recall and F-Measure after the 10 iterations.

1. Read the input data from the file.
2. Identify all columns with numerical value and categorical values and save the indices in two different np arrays
3. Using the 10-fold cross validation split the data into training and testing data with test data changing in every iteration.
4. Calculate the class prior probabilities of each class in the training data.
5. Calculate the Mean and Standard deviation of numerical attributes in the training data for each class.
6. Calculate the Gaussian probability for numerical attributes of the given test data for each class using probability density function, mean and standard deviation calculate din the previous step.

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$

7.Calculate the probabilities for the nominal data by counting the number of times given feature appeared in given class

8.Mutliply the Gaussian probability for continuous data and nominal Data probability to get the final probability

9.Predict the class value by comparing the final probabilities of the both classes

Results:

Avg Accuracy	Avg Precision	Avg Recall	Avg F-Measure
0.827	0.96	0.69	0.87

Avg Accuracy	Avg Precision	Avg Recall	Avg F-Measure
0.675	0.55	0.53	0.54

Important Insights:

1. By comparing the average results of our model on two datasets we can say that our model performed well on dataset1 when compared with dataset2. The reason what we think for the decrease in accuracy of the model is, dataset1 contains only continues values whereas dataset2 contains both continuous and categorical values.

2. Another reason might be some correlated attributes in dataset2 which could have resulted in decrease in accuracy.

Pros:

1.It is easy to understand.

2.It can also be trained on small dataset.

3.The classifier works for both the binary and multiclass features.

Cons:

1.It has a 'Zero conditional probability Problem', for features having zero frequency the total probability also becomes zero.

2.Another disadvantage is the very strong assumption of independence class features that it makes. It is near to impossible to find such data sets in real life.

Random Forests:

It is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the classes output by individual trees.

Learning algorithm

1. Each tree is constructed using the following algorithm:
2. Let the number of training cases be N, and the number of variables in the classifier be M.
3. We are told the number m of input variables to be used to determine the decision at a node of the tree; m should be much less than M.

4. Choose a training set for this tree by choosing n times with replacement from all N available training cases (i.e. take a bootstrap sample). Use the rest of the cases to estimate the error of the tree, by predicting their classes.
5. For each node of the tree, randomly choose m variables on which to base the decision at that node. Calculate the best split based on these m variables in the training set.
6. Each tree is fully grown and not pruned (as may be done in constructing a normal tree classifier)

Implementation:

The implementation of random forest is similar like decision tree.

It is an extension of bagging and building trees based on multiple samples of your training data, it also constrains the features that can be used to build the trees, forcing trees to be different.

This, in turn, can give a lift in performance.

1. The entire dataset is split into training and testing sets using 10-fold validation.

2. Training set is used to train the model to construct a decision tree.

3. Choose n – number of trees in each node.

4. We use **bagging** where we choose a training set N times with replacement from the training set.

5. For each node, we randomly select m features (m < M, where M is total number of features) and calculate the best split.

6. The we take the majority voting among all the trees to decide a label for a record.

7. Finally, the accuracy from each iteration is taken to find the average accuracy for the whole dataset.

Results:

project3_dataset1.txt:

Num-of-Trees	Avg Accuracy	Avg Precision	Avg Recall	Avg F-Measure
5	0.922	0.90	0.87	0.88
10	0.90	0.93	0.81	0.86
15	0.91	0.95	0.80	0.87
20	0.96	0.94	0.80	086

project3_dataset2.txt:

Num-of-Trees	Avg Accuracy	Avg Precision	Avg Recall	Avg F-Measure
5	0.68	0.52	0.36	0.42
10	0.67	0.56	0.33	0.39
15	0.65	0.48	0.23	0.302
20	0.67	0.61	0.21	0.29

Pros

1.The predictive performance can compete with the best supervised learning algorithms

2.They provide a reliable feature importance estimate

3.They offer efficient estimates of the test error without incurring the cost of repeated model training associated with cross-validation

4. Can effectively estimate missing data and maintains accuracy when a large proportion of the data are missing.

5. All the above benefits can also be achieved for unlabeled data i.e. unsupervised clustering is possible along with outlier

Cons:

1.Random forest models are not all that interpretable; they are like black boxes.

2.For very large data sets, the size of the trees can take up a lot of memory.

3.It can tend to overfit, so you should tune the hyper parameters.

Boosting for Kaggle:

KNN:

We varied k value from 1 to 20 and observed the accuracy on train dataset and found that we are getting high value at k =11 we used this k value for prediction on testdata

parameters Set:

k = 11

Accuracy obtained:

0.82943

Random Forest:

changed the parameters num_trees and num_features and calculated accuracies for different values. And noted the num_trees and num_features for which we got max accuracy.

We ran the testdata with the given parameters and predicted the output values.

Parameters Set:

numofTrees = 74

numoffeatures =10

Accuracy obtained:

0.83032

AdaBoost Classifier:

AdaBoost is similar to Random Forest in that they both tally up the predictions made by each decision trees within the forest to decide on the final classification

Implementation:

1. Initialize the sample weights
2. Build a decision tree with each feature, classify the data and evaluate the result
3. Calculate the significance of the tree in the final classification
4. Update the sample weights so that the next decision tree will take the errors made by the preceding decision tree into account
5. Repeat steps 2 through 4 until the number of iterations equals the number specified by the hyper parameter (i.e. number of estimators)
6. Use the forest of decision trees to make predictions on data outside of the training set

Used SKlearn Package for the training and testing on given kaggle dataset

Parameters Set:

No of estimators = 200

Max_depth = 1

Accuracy obtained:

0.84397

Pros

1. Currently one of the most accurate learning algorithms available and the classifier given a highly accurate results and estimations on what variables are important to classification.
2. Running efficiency on larger datasets is also commendable, since it can handle thousands of input variables without variable deletion.
3. Can effectively estimate missing data and maintains accuracy when a large proportion of the data are missing.
4. It can compute balancing error for unbalanced data sets that can further help in finding relation between the variables and classification with prototype computation.

Cons

1.Sensitive to noisy data and outliers.

2. A very large number of trees may actually make the algorithm slow for real-time prediction and can also cause memory limitations depending on the computing setup being used.

10 Fold Cross Validation:

For all of our algorithms we are using 10-fold cross validation, on our training datasets to split it into training and validation sets. We are essentially randomizing and splitting our data into 10 parts, then putting aside one part for validation and using the rest for training on each of our 10 iterations. This allows us to get average metric values on the performance of our models. This is better, so that we get a more reliable average accuracy, precision, recall and F1 measure from our models.