UNIVERSITY AT BUFFALO

CSE601 – Data Mining and Bioinformatics

Project 3 – Classification Report

Aayush Shah (50207564)
Aniruddh Chaturvedi (50206958)
Haril Satra (50208283)
12/2/2017

K – Nearest Neighbors Algorithm

A. Steps in KNN Implementation

1. Predefine K as the desired numbers of nearest neighbours for each test sample.
2. Calculate the mean and standard deviation for each column of the training data.
3. Normalize the training data using the mean and standard deviation (z-score normalization).

$$z = (x\text{-mean})/sd$$

4. Take one test sample at a time and normalize it using the mean and standard deviation in the same way as the train data was normalized.
5. Now for each normalized test sample we need to calculate its distance with all the training samples.
6. From these distances we obtain the labels of the samples with smallest K (nearest neighbors) distances.
7. From the labels obtained in the above step we find which label occurs the most and assign that to that test sample.
8. Repeat from steps 4 to 7 for the other test samples.

B. Pros:

- KNN is not sensitive to outliers.
- Easy and simple to implement.
- Naturally handles multi class data

C. Cons:

- Computation time is more during the prediction stage, because most of the computations are done during testing rather than training.
- High memory requirement, as it stores the training data.
- Classification results are more biased towards a category, if that category occurs much more than other categories.
- The value of the parameter K has to be defined.
- Computation cost is high, as we have to compute distance of each test sample was all training samples.

D. Result Analysis:

- KNN performed the best for dataset 1 as compared to the rest of the algorithms. However, KNN did not perform very well on dataset 2.
- This is expected because KNN works better for large datasets.

Naïve Bayes Algorithm

A. Steps in Naïve Bayes Implementation

1. Split the data into two halves based on the class labels.
2. Calculate the mean and standard deviation for each numerical attribute of both the splits of training data.
3. For each test sample we calculate the 'Class Posterior Probability'.

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

$P(H_i)$ = (no of samples of class H_i)/ (Total length of training data) = Class Prior Probability

P(X|Hi) (for numerical attribute) = Gaussian_Pdf(X,mean,std)

P(X|Hi) (for categorical attribute) = $n_{i,j}$ /ni
Where $n_{i,j}$ is number of training examples in class C_i having value x_j for attribute A_j

P(X) = n_j /n_i
Where n_j is number of training examples having value x_j for attribute A_j and n is the total number of training examples

4. Based on which Class Posterior Probability is greater we label the test sample accordingly. For example if P(0|X) > P(1|X) we label the test sample as 0.

B. Pros

- The output is fairly acceptable, even if the attribute independence assumption doesn't hold true.
- Naive Bayes classifier is fast, because prior probabilities can be calculated and stored ahead of time. The same prior probabilities are reused to calculate posteriors.
- Naïve Bayes classifier not only gives the prediction, but also the degree of certainty.
- Not sensitive to irrelevant features.

C. Cons:

- Naive Bayes assumes that the attributes are independent.
- Precision and Recall will be very low, if Naive Bayes classifier is used on a small dataset.
- If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero-Probability"

D. Result Analysis:

- Naïve Bayes performed the best for dataset 2 as compared to the other algorithm.
- Naïve Bayes and Decision tree performances should be comparable. They are comparable for dataset 1 however Naïve Bayes performs better for dataset 2.

Decision Tree: CART Algorithm

A. Steps in CART Algorithm Implementation

1. We have built the tree in top down manner using a greedy approach for splitting.
2. For the current training data set, implement a binary split of the data as follows:
    a. For each attributes, iterate through each value of the attribute and choose the value which gives the lowest value of the cost function.
    b. The cost function used is Gini Index, which is as follows:

$$\left(1 - sum\left(p\left(\frac{c}{t}\right) * p\left(\frac{c}{n}\right)\right)\right) * (size\_of\_split/total\_number\_of\_records)$$

    Where $p(\frac{c}{n})$ is the relative frequency of the records belonging to class c for all the records at node n.
3. At each binary split, create an object of class Node which contains the index of the split attribute and value of the attribute at which split is done.
4. If the split attribute is numerical:
    a. All the training records with the split attribute value less than or equal to the split value are assigned to the left child node and the rest of records to the right child node of the current node.
5. If the split attribute is categorical:
    a. All the training records with the split attribute value equal to the split value are assigned to the left child node and the rest of records to the right child node of the current node.
6. Build the rest of the tree recursively by repeating the Steps 2 to Step 5 for the left and right child.
7. Stop growing the tree when the number of records in either of left or right child becomes zero. Convert both the children into a terminal node and assign a final class label to both according to the most common class label out of all the records at that node.
8. To classify each test data, recursively navigate through each node of the tree, going to the left or right child according to the split value. If it is not a terminal node, continue traversing ahead. If it's a terminal node, return the class label at that terminal node as our predicted class label and Stop.

B. Pros:

- The output of decision tree is easy to interpret and visualize as compared to other algorithms.
- Classification is fast.
- It can be used as a white box while combining it with other decision techniques.

C. Cons:

- For data including categorical variables with different number of levels, information gain in decision trees is biased in favor of those attributes with more levels.
- Decision tree is instable and changes even on slightest change of the data which is undesirable since we want a model to be robust to noise and generalize well.
- Classification on Decision Tree is primarily based on expectations. For some cases it is impossible to plan for all contingencies. (Overfitting)

D. Result Analysis:

- Decision tree performance on both the datasets was neither the worst nor the best.
- It took a lot more time than KNN and Naïve Bayesian but it's metrics were comparable with the rest of them.

Random Forests Algorithm

A. Steps in Random Forests Implementation

1. Predefine the initial parameter T as the number of trees to grow.
2. Predefine the initial parameter m, as the number of attributes chosen randomly to calculate the best split attribute for each node. We have chosen m as the square root of the number of attributes in the data.
3. Let N be the size of original training data. For each tree, perform the following:
    - Generate a training set of size N by randomly choosing training records from the original training data with replacement.
    - For the creation of each node of the decision tree, select m attributes randomly and calculate the best split attribute from those m attributes.
4. To classify each test data, make a prediction on each of the tree formed and assign the test data to the class label which is returned majority of the times among all the trees.

B. Pros

- Fast to train because only a subset of features in used to train the trees.
- Random Forest's are more accurate as compared to Decision Trees.
- It reduces overfitting.
- Runs efficiently on large datasets.

C. Cons:

- Classification is slow, as multiple trees are used for prediction.
- It is difficult to visualize and interpret, as multiple trees are generated randomly.
- For categorical data, random forests are more biased in favor of attributes with more classes.

D. Result Analysis:

- Random forests perform the best (almost at par with KNN) for dataset 1.
- This is expected because they have high number of predictive variables and huge sample size.

Boosting(AdaBoost) Algorithm

A. Steps in AdaBoost Implementation

1. Assign an initial weight of (1/number of records) to all the training records.
2. Predefine the initial parameter T as the number of trees to grow.
3. Repeat the steps 4 to Step 10 until T number of trees are grown:
4. Create a bootstrap sample randomly from the original training data based on the current weights of the training records.
5. Train a Decision Tree model on the sample and apply it on the original training data.
6. Calculate the error for the model as follows:

$$error = \frac{sum\ of\ weights\ of\ missclassified\ records}{sum\ of\ weights\ of\ all\ records}$$

7. If the error is greater 0.5, start again from step 4.
8. Calculate the model's importance, alpha as follows:

$$alpha = 0.5 * \ln\left(\frac{1 - error}{error}\right)$$

9. The weight of each record i, is updated as follows:

$$w_i = w_i * \exp\left(-a * y_i * C(x_i)\right)$$

Where a is the alpha for the current model,
y_i is the class label predicted by the model,
$C(x_i)$ is the true label for that record

10. Normalize the weights so that they lie in range [0,1].
11. To classify each test data, make a prediction on each of the tree formed and the final prediction is based on the weighted average of all the classifiers with weight representing the alpha values of each classifier.

B. Pros:

- AdaBoost is more adaptive in the sense that subsequent classifiers built are modeled in favor of those records misclassified by previous classifiers.
- No additional parameters to tune except T, the number of models to build.
- Handles high dimensional data very well.

C. Cons:

- AdaBoost is sensitive to noisy data and outliers.
- Though adaptive to weak classifiers, if the weak classifiers are too weak and complex, might lead to overfitting.
- Statistics suggest it is particularly susceptible to uniform noise.

D. Result Analysis:

- Boosting performs better than decision trees but not as good as random forests.
- It performs better than the decision trees because it boosts a set of weak learners to strong learners.

Cross Fold Validation (10 fold)

Average Accuracy

Algorithm	Dataset 1	Dataset 2
KNN	0.966071429	0.656521739
Naïve Bayes	0.937445055	0.701358696
Decision Tree	0.930796703	0.597101449
Random Forests	0.952774725	0.672644928
Boosting	0.946428571	0.649456522

Average Precision

Algorithm	Dataset 1	Dataset 2
KNN	0.977719503	0.51312298
Naïve Bayes	0.925020433	0.570786871
Decision Tree	0.892669393	0.421808287
Random Forests	0.96510092	0.549105339
Boosting	0.925	0.498730765

Average Recall

Algorithm	Dataset 1	Dataset 2
KNN	0.926632442	0.390232794
Naïve Bayes	0.906442446	0.620826425
Decision Tree	0.936468901	0.464505816
Random Forests	0.904733045	0.319036614
Boosting	0.926767677	0.438508531

Average F1-Measure

Algorithm	Dataset 1	Dataset 2
KNN	0.950165536	0.417178023
Naïve Bayes	0.91420727	0.585813279
Decision Tree	0.912727087	0.434358761
Random Forests	0.933535861	0.395494587
Boosting	0.923558897	0.447494099

Note: All these values are in a range of 0-1 and have not been multiplied to obtain the percentage.

Choice Description

KNN:

1. **K:** Choosing an optimal value of k is essential as a smaller value of k might make the model sensitive to noise, whereas a larger k value might also include data points from the other classes in the neighborhood. For dataset1 and dataset2, we are getting the best results for k = 5.

2. **Distance Metric**: We have used the Euclidean distance method as the distance metric. Other distance functions like cosine similarity or chi square could also be used here. However, Euclidean distance is the most widely used and hence we have used that.

3. **Continuous Features:** If the feature is continuous, for each test sample, we calculate its Euclidean distance from each of the training data record. It is then assigned a class label which is most common among its K nearest neighbors, which is measured as the minimum Euclidean distance.

4. **Categorical Features:** If the feature is categorical, for each test sample, we calculate its hamming distance from each of the training data record. When the value of the categorical attribute is same for the test and training data record, the distance is zero else one.

Decision Trees:

5. **Categorical Features:** If the feature is categorical, we create a binary split for each node where all the training records with the split attribute value equal to the split value are assigned to the left child node and the rest of records to the right child node of the current node.

6. **Continuous Features:** If the feature is continuous, we create a binary split for each node where all the training records with the split attribute value less than or equal to the split value are assigned to the left child node and the rest of records to the right child node of the current node.

7. **Best Feature:** The cost function used by us is Gini index. The Gini index at each possible split points is calculated and the one with the minimum Gini index is chosen as the best split point.

8. **Post Processing:** Post – pruning can be performed on the tree after growing it to its full size. Starting from the bottom level, nodes are pruned and if the generalization error of the tree improves after the pruning, then the sub-trees at that level are replaced by leaf nodes.

Naïve Bayes:

9. **Continuous Features:** For continuous features, we have used Gaussian distribution. We separated the records belonging to class 0 and class 1. We then calculated the mean and the standard deviation of each continuous attribute for both the classes and created a Gaussian model for both. For each test sample, we calculated and multiplied the probability distribution function of each continuous attribute value, separately for both classes. It is then classified to the class with the maximum probability.

10. **Zero Probability:** If any of the attribute posterior probability becomes zero, then the class posterior probability would become zero. Laplacian correction could be used to handle such cases, where we add 1 to each case with probability zero. However, as the dataset provided to us doesn't have any such case with zero probability of the attributes. Hence we haven't handled the zero probability case separately.

Comparison between KNN, Naïve Bayes and Decision Trees.

1. Accuracy
   - Decision Tree based classification is more accurate as compared to KNN and Naive Bayes.
   - KNN requires larger number of data samples in order to get good accuracy.
   - Naive Bayes is more accurate as compared to KNN.

2. Visualization
   - Decision tree based classification is easy to interpret and visualize for humans as compared to KNN and Naive Bayes.

3. Speed based on size of data
   - KNN is slower for large datasets.
   - Naive Bayes is faster for large datasets.
   - Decision Trees are faster for large datasets.

4. Effectiveness based on size of data
   - KNN is more effective on small datasets.
   - Naive Bayes is more effective on large datasets.
   - Decision Trees are more effective on large datasets.

5. Sensitivity to noisy data
   - KNN is sensitive to noisy data.
   - Naive Bayes is insensitive to noisy data
   - Decision Trees are insensitive to noisy data.

Comparison of Decision Trees, Random Forests and Boosting

1. Random Forest is more accurate as compared to Decision Trees, as it introduces more Diversity and Variance.

2. Random Forest and Ada boosting are similar in average accuracy.

3. Random Forests are less sensitive to outliers and Parameter choices as compared to Ada boosting.

4. Random Forests works better with Deep Trees (Level >= 7).

5. Ada Boosting performs well with Shallow Trees (5 – 15 leaves).

6. Random Forest is more efficient as compared to Decision Trees, as we consider only a subset of features instead of all the features.