STATE UNIVERSITY AT BUFFALO

Project 3 : Classification Algorithms

Yuze Liu 50207903
Luting Chen 50133507
Vicky Zheng 50037709

12/10/2016

1 K-NEAREST NEIGHBORS ALGORITHM

1.1 ALGORITHM DESCRIPTION

K-Nearest-Neighbors algorithm is a method used for classification and regression. In K-NN classification, the output will be the class membership. The objective will be classified by the majority votes of its neighbors, with the objective being assigned to the class most common among its k nearest neighbors.

In this project, we implement the 10-fold cross validation with the K-NN classify algorithm.

First,we split the whole dataset into 10 parts, we pick one part as the testing set and other 10 parts as the training set. For each sample in the testing set, we calculate the Euclidean distance between this sample to all the other samples in the training set. Then we pick the k nearest samples as its neighbors. We will classify this sample by the majority votes of that k neighbors.

In the project, there are only 2 classes. So we just need to compare the two weights.If the neighbor is classified as class 0, then the weight of this sample to be classified in class 0 will be : $weight0 = weight0 + \frac{1}{dis}$, dis is the Euclidean distance between the neighbors and the sample, other wise $weight1 = weight1 + \frac{1}{dis}$, then we compare weight0 and weight1 after go through all the neighbors of this sample, the sample will be classified with the higher weight.

After get the classification result from the test set, then we calculate all the four measurement values and then we pick another dataset from the remaining 9 datasets as the test set and other 9 as training set, repeat this procedure until all the ten datasets have been used as test set once.

In the end, we calculate the average of the four measurement values.

1.2 Pros and Cons

Advantage:
The algorithm is effective when the dataset is very large.
The algorithm is robust to noisy training data.

Disadvantage:
Need to determine the value of parameter K.
There are a lot of choices to choose what kind of distance we will use to determine the neighbors.
Computation cost is quite high because we need to compute the distance of each query instance to all training samples.
If we have nominal data, we need to label the nominal value first and then calculate the distance.

1.3 Result Evaluation

Results for project3 dataset1.txt are:
Accuracy: 0.92615914787
Precision: 0.922329566995
Recall: 0.87079481314
F: 0.89442003084

The above value is the optimal solution I get, k = 3

Results for project3 dataset2.txt are:
Accuracy: 0.60625
Precision: 0.411791272453
Recall: 0.318770209838
F: 0.357137604338

The above value is the optimal solution I get, k = 4

2 Decision Tree

2.1 Algorithm Description

Decision Tree is a supervised learning method for classification. The goal is to create a tree structured model that predicts the class of a test sample by learning decision rules inferred from the data features. The decision rules are represented as internal nodes in the tree. Each internal node denotes a test on an attribute, each branch denotes the outcome of a test, and each leaf node holds a class label. When a new test sample is needed to be classified, it goes down the decision tree from root to a leaf node according to the rules stored in internal nodes and it's class label is determined

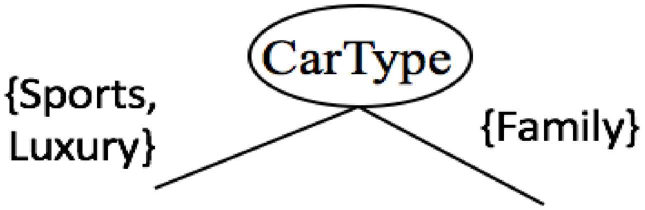


Figure 2.1: 2 way splitting example

by the label stored in the leaf node. We implemented a 2-way-splitting decision tree with prepruning.

2.2 Implementation

Preprocessing In preprocessing, we made all attributes of different types to be processable. For nominal attributes, we numbered them to make implementation easier. For example in dataset2, "Absent" was labeled as 0 and "Present" was labeled as 1. For continuous attributes, we discretized each of them into k bins and the choice of k can be inferred by cross validation. A sample's feature is labeled 0 if it's in the 1st bin, 1 if it's in the 2nd bin, 2 if it's in the 3rd bin and so on so forth, until k-1 if it's in the kth bin. And it becomes an ordinal attribute because samples labeled 0(landed in the 1st bin) are smaller than those labeled 1(landed in the 2nd bin). So the order is: samples labeled 0 < samples labeled 1 < ... < samples labeled k-1. Ordinal attributes can also be labeled as integers accordingly. As a result, after preprocessing, all features will be represented as integers.

Splitting criterion We used 2-way splitting to split tree nodes when it's needed and we used misclassification error to compute the impurity of parent and children nodes so as to choose the best splits. At tree node i, we tried out all possible 2- way splits: we tried each attribute(f) and attribute value(fv) combination. And we computed their corresponding impurity IMP: IMP = p * misclassificationError(leftChild) + (1-p) *misclassificationError(rightChild). p = $\frac{left\_node\_size}{parent\_node\_size}$. We got the minimum IMP and the corresponding feature f used to split as well as the single feature value fv for one child node. In example Figure 2.1, f is "CarType" and fv is "Family".

Terminating Criterion The terminating criterions we used are listed below:

1. current node has only one sample in the node.
2. all points in the node have same label(impurity is 0).
3. all points in the node's features are all the same.

4. all possible splits's are bad: children's impurity > parent's impurity.

5. the node's impurity is smaller than the threshold impurity (prepruning)

When the threshold impurity is not 0, some nodes, whose impurity is smaller than the threshold impurity, will stop splitting and become leaf nodes even though they don't fulfill the first 4 criterions. Because the pruning doesn't happen after a full tree is grown, this kind of pruning is called prepruning. When the threshold impurity is 0, pruning is disabled.

If a node meets Criterion 4, it means no matter how we split the node, the resulting impurity is always bigger than the original impurity. So, we think it is not worthwhile to split this node. According to our experiments, a lot of leaf nodes with only one sample will be created if we delete Criterion 4.

Except Criterion 1, all the other criterions may create leaf nodes whose class labels are heterogeneous. We used majority vote to assign the label of heterogeneously labeled leaf nodes.

KEY FUNCTION USAGE We designed three key functions to finish the pipeline of decision tree classification. The input parameters are omitted for better readability in the report.

1. hunt(): learn the tree structure

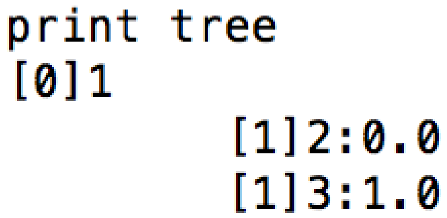
2. treenode_list_label(): majority vote each leaf node and get leaf node class labels

3. assign_labels(): classify new test samples according to the learned decision tree

TREE PRESENTATION Our tree presentation is very humble. The tree grows from left to right: the root of the leftmost node and each colunm is one level of the tree. For each internal node, it follows the format of "[parent_id] my_id". For the root node, it's parent_id is 0. For each leaf node, it follows the format of "[parent_id] my_id:classLabel". Since for Project 3, we only deal with 2 classes, leaf nodes are either "[parent_id] my_id:0" or "[parent_id] my_id:1". We can also print out the samples each node includes as well as internal nodes's splitting conditions. The following photo is an example. In the example Figure 2.2, node 1 is the root and it has 2 children node 2 and node 3; node 1 is not a leaf node, so it doesn't have the part ':0.0' or ':1.0'; node 2 is the leaf node and all data that goes to this node is labeled as 0; node 3 is also a leaf node and all data that goes to this node is labeled as 1.

2.3 PROS AND CONS

Advantage:
Decision tree is easy to interpret, because it can be seen as a set of decision rules.
At each internal node, decision tree will choose the feature that yields the lowest



```
print tree
[0]1
        [1]2:0.0
        [1]3:1.0
```

Figure 2.2: Tree Presentation Example

children's impurity, so it has the ability of selecting the most discriminatory features.
Handling both continuous and discrete data.
Disadvantage:
Need to discrete data for some particular construction algorithm.
When dataset is small, it may yield large errors.
Without pruning, the full tree can be complicated and overfitting.

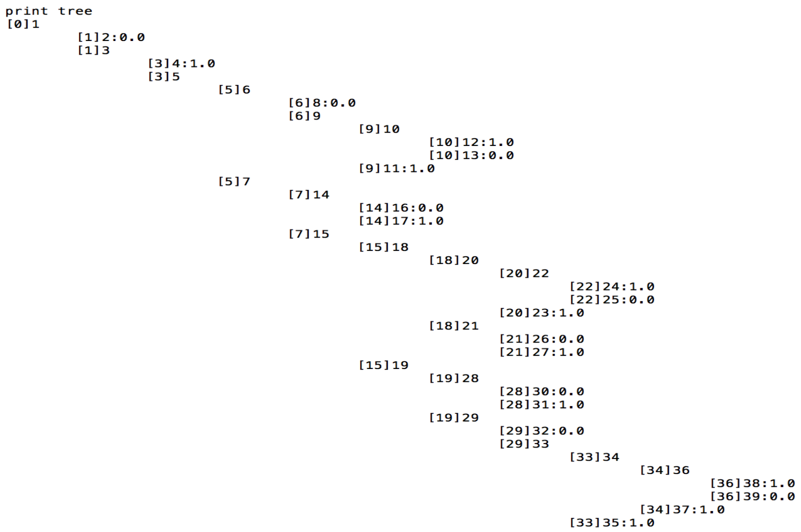
2.4 Result Evaluation

10 fold cross validation average scores, bin size is 5.
Results for project3 dataset1.txt are:
Accuracy: 0.81071429
Precision: 0.8748324
Recall: 0.57385281
F: 0.63581657

Results for project3 dataset2.txt are:
Accuracy: 0.65217391
Precision: 0.39571429
Recall: 0.08718615
F: 0.13374561

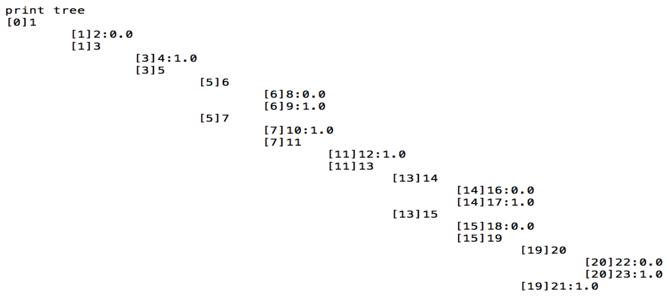
For dataset2, there are some cross validation rounds get 0 for recall and F scores, yielding very small averages.

The following is a small experiment on the effect of prepruning. For dataset2, if the threshold impurity is 0, we will have 49 nodes in the learned decision tree(the result figure is big, so we exclude this figure). While when threshold is 0.3, there are 39 nodes and when threshold is 0.34, there are 23 nodes. As can be seen, more subbranches are cut when threshold impurity gets bigger.



```
print tree
[0]1
        [1]2:0.0
        [1]3
                [3]4:1.0
                [3]5
                        [5]6
                                [6]8:0.0
                                [6]9
                                        [9]10
                                                [10]12:1.0
                                                [10]13:0.0
                                        [9]11:1.0
                        [5]7
                                [7]14
                                        [14]16:0.0
                                        [14]17:1.0
                                [7]15
                                        [15]18
                                                [18]20
                                                        [20]22
                                                                [22]24:1.0
                                                                [22]25:0.0
                                                        [20]23:1.0
                                                [18]21
                                                        [21]26:0.0
                                                        [21]27:1.0
                                        [15]19
                                                [19]28
                                                        [28]30:0.0
                                                        [28]31:1.0
                                                [19]29
                                                        [29]32:0.0
                                                        [29]33
                                                                [33]34
                                                                        [34]36
                                                                                [36]38:1.0
                                                                                [36]39:0.0
                                                                        [34]37:1.0
                                                                [33]35:1.0
```

Figure 2.3: Prepuning with impurity threshold 0.3



```
print tree
[0]1
        [1]2:0.0
        [1]3
                [3]4:1.0
                [3]5
                        [5]6
                                [6]8:0.0
                                [6]9:1.0
                        [5]7
                                [7]10:1.0
                                [7]11
                                        [11]12:1.0
                                        [11]13
                                                [13]14
                                                        [14]16:0.0
                                                        [14]17:1.0
                                                [13]15
                                                        [15]18:0.0
                                                        [15]19
                                                                [19]20
                                                                        [20]22:0.0
                                                                        [20]23:1.0
                                                                [19]21:1.0
```

Figure 2.4: Prepuning with impurity threshold 0.34

3 Decision Tree with Random Forest

3.1 Algorithm Description

Decision tree with random forest is an ensemble learning method for classification. Random forests can be built using bagging with random attribute selection at each splitting node. For bagging, we randomly picked the same number of samples as the training dataset with replacement and this is one round. We repeated the process for n times and each round will generate a decision tree. We will get n decision trees in the end and the class of a new test sample is determined by the equally weighted voting among all learned decision trees.

3.2 Pros and Cons

Advantage:
tend to be more accurate
no distribution assumptions
robust against outliers
Because random forests consider many fewer attributes for each split, they are efficient on very large databases
Disadvantage:
less efficient sampling compared to boosting

3.3 Result Evaluation

10 fold cross validation average scores, bin size is 5, ensemble number is 5.
Results for project3 dataset1.txt are:
Accuracy: 0.90535714
Precision: 0.9282444
Recall: 0.83087705
F: 0.86405575

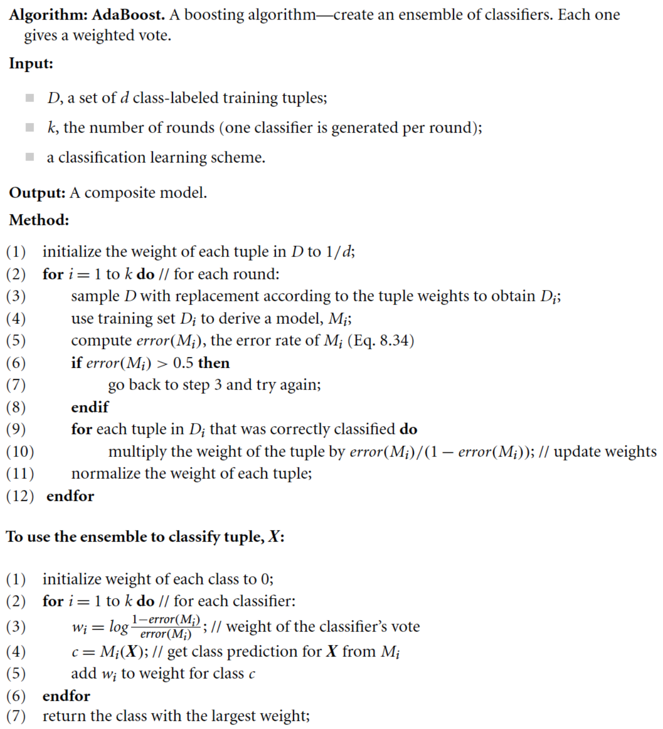
Results for project3 dataset2.txt are:
Accuracy: 0.65652174
Precision: 0.51666667
Recall: 0.10631674
F: 0.16529107

We can see a performance improvement compared to the simple decision tree.

4 Decision Tree with Boosting

4.1 Algorithm Description

Decision tree with random forest is also an ensemble learning method for classification. The main differences between boosting and Random forest is boosting doesn't have random feature selection when building trees and the final voting process for new samples is weighted. The weights for each classifier is affected by its weighted misclassification error. The following pseudocode Figure 4.1 describes how we implemented boosting[1]. Note when computing classifier weight $w_i = log\frac{1-error(M_i)}{error(M_i)}$, in case of "divided by zero" problem, we replaced a very small value like $1e^{-20}$ for 0 if $error(M_i)$= 0.



Algorithm: AdaBoost. A boosting algorithm—create an ensemble of classifiers. Each one gives a weighted vote.

Input:

- D, a set of d class-labeled training tuples;
- k, the number of rounds (one classifier is generated per round);
- a classification learning scheme.

Output: A composite model.

Method:

```
(1)  initialize the weight of each tuple in $D$ to $1/d$;
(2)  for $i = 1$ to $k$ do // for each round:
(3)      sample $D$ with replacement according to the tuple weights to obtain $D_i$;
(4)      use training set $D_i$ to derive a model, $M_i$;
(5)      compute $error(M_i)$, the error rate of $M_i$ (Eq. 8.34)
(6)      if $error(M_i) > 0.5$ then
(7)          go back to step 3 and try again;
(8)      endif
(9)      for each tuple in $D_i$ that was correctly classified do
(10)         multiply the weight of the tuple by $error(M_i)/(1 - error(M_i))$; // update weights
(11)     normalize the weight of each tuple;
(12) endfor
```

To use the ensemble to classify tuple, X:

```
(1)  initialize weight of each class to 0;
(2)  for $i = 1$ to $k$ do // for each classifier:
(3)      $w_i = log\frac{1-error(M_i)}{error(M_i)}$; // weight of the classifier's vote
(4)      $c = M_i(X)$; // get class prediction for $X$ from $M_i$
(5)      add $w_i$ to weight for class $c$
(6)  endfor
(7)  return the class with the largest weight;
```

Figure 4.1: Pseudocode for AdaBoosting

4.2 Pros and Cons

Advantage:
more accurate
very fast
Disadvantage:
tend to overfit not as robust to errors and outliers

4.3 Result Evaluation

10 fold cross validation average scores, bin size is 5, ensemble number is 5.
Results for project3 dataset1.txt are:
Accuracy: 0.92678571
Precision: 0.92975827
Recall: 0.88075427
F: 0.89874908

Results for project3 dataset2.txt are:
Accuracy: 0.68913043

Precision: 0.61375791
Recall: 0.39169192
F: 0.44046561

We can see a performance improvement compared to the simple decision tree.Our boosting performs better than random forest because when learning a new classifier in the ensemble, it tends to choose data samples that are misclassified in previous rounds and it uses weighted voting rather than majority vote in random forest.

5 Naive Bayes

5.1 Algorithm Description

Naive bayes is a classification algorithm that is based on the Bayes Theorem. Naive bayes aims to classify the probability of a sample X being in a class H_i using Bayes Theorem. The Bayes theorem is:

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

The things that we are classifying often have multiple attributes which we will refer to as A where $A = (A_1, A_2, ..., A_d)$. Let X be something we are trying to classify and $X = (x_1, x_2, ..., x_d)$. $P(X)$ is the prior probability of X where $P(x_j) = \frac{n_j}{n}$ where n_j is the number of of training samples in our training set that have the value x_j for attribute A_j.

$P(H_i)$ is simply the class prior probability.

$P(X|H_i)$,the descriptor posterior probability, can be calculated by:

$$P(X|H_i) = \prod_{i=1}^{d} P(x_j, H_i)$$

Calculating the descriptor posterior probability reveals on of the weaknesses of Naive Bayes. If a single value $P(x_j, H_i)$ is 0 then the entire product of $P(X|H_i) = \prod_{i=1}^{d} P(x_j, H_i)$ will evaulate to 0 which will cause $P(H_i|X) = \frac{P(H_i)P(X|H_i)}{P(X)}$ to also evaluate to 0. This can be corrected by using a Laplacian correction where if there is an $n_j = 0$, then we will simply add 1 to every n_j and increase the total number of samples to $n + k$ where k is the number of possible values the attribute A_j can take on.
You will also notice that $P(H_i|X) = \frac{P(H_i)P(X|H_i)}{P(X)}$ does not work for continuous values because we cannot count continuous values to get posterior probabilities. I chose to address this by assuming a Gaussian distribution to use:

$$P(H_i|x_j) = \frac{1}{\sqrt{2\pi\sigma^2_{H_i,x_j}}} e^{-\frac{1}{2}(\frac{H_i - \mu_{H_i,x_j}}{\sigma_{H_i,x_j}}^2)}$$

I chose to use this method because it seemed to be one of the most popular methods out there for dealing with continuous values when implementing Naive Bayes.
After we calculate $P(H_i|X) = \frac{P(H_i)P(X|H_i)}{P(X)}$ for all H_i, we assign X to the class H_i where $P(H_i|X)$ is the maximum probability.

5.2 Pros and Cons

Some of the pros of using Naive bayes is that its simple to implement and it is efficent. One of the cons, however, is that it assumes attribute independence, which of course is not true for all datasets. Another con is that the descriptor posterior probability may evaluate to 0 - we have to prevent this by using a Laplacian correction. This can happen often in small datasets so Naive bayes performs best on large datasets.
Something else that may be considered a con is that there are multiple ways to deal with continuous attributes. I chose to handle this by using Gaussian Naive Bayes. This may not perform well for other datasets that have different distributions.

5.3 Result Evaluation

I implemented k-cross fold by randomly shuffling my dataset, and partitioning it into k test sets and k training sets. I took the average of the k accuracy, precision, recall and F-values.

Results for project3 dataset1.txt are:
Accuracy: 0.712582417582
Precision: 0.589052375343
Recall: 0.748140191881
F: 0.657284240816

Results for project3 dataset2.txt are:
Accuracy: 0.671195652174
Precision: 0.51574025974
Recall: 0.761451858741
F: 0.612021438585

6 Other clarification

When calculating scores like recall, precision and F score, we may come across "divided by zero" problem. For example, to compute precision = $\frac{truePositive}{truePositive+falsePositive}$, if the denominator is 0, it means both truePositive and falsePositive are 0, and truePositive is the numerator then we can just assign precision as 0. We solved "divided by zero" problem similarly for recall and F score.

References

[1] Han, Jiawei, Jian Pei, and Micheline Kamber. Data mining: concepts and techniques. Elsevier, 2011.