CS 484 Project – Task 2: Naïve Bayes vs Decision Trees vs K-Nearest Neighbor

Pavan Vittala

Pakeezha Arfany

**Important Info for TA**

To run the code, simply do the following:

1. In the Final Project folder I submitted, navigate to the Implementation folder.
2. Run the following commands:
   1. javac \*.java
   2. java NaiveBayesClassifier
3. This will run my code on all 10 datasets. You can then see the output directly from the terminal, if you’d like. Otherwise a text file was submitted with the output.

**Abstract**

We designed our project from the very beginning to test how well three different supervised learning algorithms would function when pitted against ten different datasets. The three algorithms we decided to test were the Naïve Bayes algorithm, the 1-nearest neighbor algorithm, and the J48 decision trees algorithm. We implemented the Naïve Bayes algorithm and found that it was rather good at classification. The results of the Naïve Bayes and J48 decision trees algorithms were consistent with what we learned in class and thus could be deemed reasonable. However, the results of the 1-nearest neighbor algorithm may have been severely inflated due to testing the algorithm on the training set itself. To maintain scientific control, we were forced to test all the algorithms the same way we were testing our implementation of the Naïve Bayes algorithm. We chose to use the training set as the testing set to make our lives easier (in terms of coding). However, we realized, in hindsight, that this method of testing may not have been a good combination with K-nearest neighbor.

**Introduction**

Machine learning is a hot topic in the field of Computer Science today. Its applications are widespread and well-sought after by industry professionals. Machine learning can be cleanly dichotomized into two separate categories: supervised learning and unsupervised learning. This study focused on supervised learning. In terms of supervised learning, there exist several ubiquitous supervised learning algorithms. We have chosen to focus on three of these algorithms. The first is the Naïve Bayes algorithm. This algorithm focuses on using the occurrences of attributes given a class label to determine the class of an unseen instance. The second algorithm is the J48 decision trees, implemented in Weka. This supervised learning algorithm utilizes splitting on attributes that provide the best “information gain”. There are multiple ways to measure information gain (GINI, entropy, etc.). The last algorithm is the k-nearest neighbor algorithm. This algorithm measures the distance between instances to classify instances into a class. Some of the distance measures are Euclidean distance and Cosine-Similarity. More specifically, we are planning on utilizing the 1-nearest neighbor algorithm.

In our project, we chose to compare the results of our algorithm (Naïve Bayes) and the other two (J48 trees and 1-nearest neighbor). In our implementation of Naïve Bayes, the results output displays the correct and incorrect number of instances classified and the confusion matrix for the test. This is like Weka’s output and can be compared easily. In addition to the actual results, we thought it would be interesting to compare the runtime of our algorithm when it’s given datasets of varying sizes. The instances and number of attributes in the dataset are primarily what affect the runtime of our program. To test this, we’ve chosen datasets that have a varying number of these aspects. Just as an example, one of our datasets, the ads datasets, has 1558 attributes and 3279 instances. Another dataset, the poker hand dataset, has 1,025,010 instances with 11 attributes. With this wide variance of dataset size, we hope to better test the capabilities of our algorithm.

**Related Work**

Comparisons of supervised learning algorithms have been done frequently in the past. A notable study was conducted by the Department of Computer Science at Cornell University in 2006. Numerous supervised learning algorithms had been developed by 2006, but not many of them had been properly tested. The last large-scale evaluation of supervised learning had been done in the 1990s by the Statlog Project. To bridge this gap, the experts in Cornell compared the following supervised learning algorithms: SVMs, neural nets, logistic regression, naïve Bayes, memory-based learning, random forests, decision trees, bagged trees, boosted trees, and boosted stumps. The paper mentions that a significant motivation behind the evaluation was to determine the performance of the methods, which is identical to what we’re trying to do.

The metrics they are using to compare the algorithms are slightly different from ours. The experts at Cornell are using three categories of performance evaluation: threshold metrics, ordering/ rank metrics, and probability metrics. The threshold metrics are accuracy, F-score, and lift. Out of these three metrics, we are only using accuracy. Ordering/ rank metrics consist of determining whether the ordering of the cases was preserved. This is done using primarily the ROC curve. Our implementation of Naïve Bayes does not measure the ordering/ rank in any manner. Finally, the probability metrics are described by evaluation metrics like the confusion matrix, a feature which our Naïve Bayes implementation reports to the user.

**Solution**

For the algorithm we had to code ourselves, we decided to implement the Naïve Bayes classification algorithm. This algorithm relies on the use of Bayesian methods, which state that learning and classification are based on the principles of probability. By using the prior probability of each attribute-class label pair, we can generate a posterior probability distribution over the attribute-class label pairs. This produces a generative model (as opposed to a lazy learner like K-nearest neighbor) that can produce accurate estimates of classification. However, the Naïve Bayes classifier does have some disadvantages. It is “naïve” in the sense that it assumes that the attributes in a dataset are purely independent. This assumption is not always correct because attributes could be dependent. For example, in a census data dataset, we can find attributes like height and weight. Attributes like height and weight are independent, but if you introduce something like BMI (body-mass index), this dataset can’t be adequately modeled with a Naïve Bayes classifier because BMI is dependent on both weight and height. Some advantages of Naïve Bayes are that its fast and not sensitive to irrelevant attributes, like K-nearest neighbor is.

Our implementation has the following features: continuous-attribute Gaussian distribution estimation, conditional probability “zero” consolidation with Laplace smoothing, and efficient space and time complexity.

To determine the probabilities for continuous variables, we had a choice of three options: discretize the range into bins, split the attribute (two-way split), or use a probability density estimation. Discretizing the range into bins is the least attractive option because it violates the independence assumption of the Bayes Theorem that this classifier is built upon. And the second option of splitting on a value seemed arbitrary to us. So, we decided to go with the probability density estimation. For each continuous attribute-class label pair, we calculated the sample mean and sample variance so that we could estimate the probability of any given value of that continuous attribute occurring the dataset. This method isn’t perfect because not all continuous attributes will follow a Gaussian/ Normal distribution, but it seemed like a good tradeoff to us. Implementing this was a simple matter of first calculating the sample mean for each continuous attribute-class label pair. This is done via one pass of the dataset. On the second pass, the sample variance can be calculated. A second pass is required because the mean is required to calculate the variance.

The Laplace smoothing was implemented to prevent probabilities of zero from making the entire Naïve Bayes expression zero (since the formula requires successive multiplication). This was very simple to implement. Once the model was built (the number of occurrences of attributes), we checked to see if any of the occurrence values were 0. If any of these were zero, that probability would evaluate to zero. So, we simply checked if any zeroes existed, and if they did, we used Laplace smoothing for the entire testing phase. Upon the professor’s recommendation, Laplace smoothing was used for all instances.

Our algorithm’s runtime is dependent on three features: the number of attributes, the number of possible values nominal/ ordinal attributes can take on, and the number of class labels. When both building and testing the model, the worst-case running time of our algorithm is . This may look bad, but since most datasets don’t have a large number for *all* four of these, it’s usually not too slow. To understand the space complexity of our algorithm, you must first understand how I organized my model into a data structure. At the top layer, our algorithm uses an ArrayList to store all the attributes in our dataset. Each attribute is stored slightly differently, based on whether it’s a continuous, ordinal, or nominal attribute. Nominal attributes and ordinal attributes have an internal array that stores information about the possible values each nominal/ ordinal value can take on. These possible values are further broken down by another internal array that encodes the count for each class label. As for continuous attributes, they have an internal array for each class label. Each continuous attribute-class label pair then has its own sample mean and sample variance. Based off this information, one can see that the worst-case space complexity is approximately . We originally planned on loading the entire dataset into memory, but we realized this would be incredibly unwise due to the size of some of our larger datasets.

**Experiments - Data**

The following table illustrates our datasets:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Brief Description** | **Number of Records** | **Attributes** | **Class Labels** |
| Connect-4 | Dataset used to predict the outcome of the first player in connect-4. | 67,557 | 42 | 3 |
| Adult | Predict whether income exceeds $50k/year based on census data. | 48,842 | 14 | 2 |
| Cover Type | Predicting forest cover type from cartographic variables. | 581,012 | 54 | 7 |
| Car Evaluation | Determine the viability of a used car based on analysis of car data. | 1728 | 6 | 3 |
| Poker Hand | Predict outcome of poker match based on hand. | 250010 | 11 | 10 |
| Breast Cancer Wisconsin (Diagnostic) | Diagnose whether a person has breast cancer based on characteristics of a potential breast cancer carcinoma cell. | 569 | 32 | 2 |
| Internet Advertisements | Determine if a website has ads or not. | 3279 | 1558 | 2 |
| Contraceptive Method Choice | Effect of contraceptive method choice and other personal characteristics on the outcome of pregnancy. | 1473 | 9 | 3 |
| Abalone | Predict the age of an abalone (a type of sea snail) using physical measurement data. | 4177 | 8 | 28 |
| Bank Marketing | Will a bank client subscribe to a term deposit based on data about the customer? | 45211 | 17 | 2 |

As one can see, we tried to select a set of datasets that were not only interesting, but also large enough to be interesting. We tried to select datasets that had large instance counts, attribute counts, and class label counts. These qualities were existent across all the datasets. Some datasets, like the breast cancer and contraceptive method choice datasets had a low instance count, but made up for it with their larger attribute count.

**Experiments - Experimental Setup**

The data was preprocessed in the following way for each algorithm:

Note about “My method” listed below: For all the datasets, I added four lines to the top of the text file before testing it using our Naïve Bayes implementation. I added the names of all the attributes in the first line, their types (ordinal, continuous, nominal, etc.) in the second line, the number of possible values the attribute can take on for the third line, and the actual possible values in the fourth line. This was done to create space as necessary in memory for the data structure that stored the model.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Naïve Bayes (Our Implementation)** | **J48 Trees** | **1-Nearest Neighbor** |
| Connect-4 | My method | None | None |
| Adult | My method | None | None |
| Cover Type | My method | NumericToNominal filter for all attributes but elevation, aspect, and slope | NumericToNominal filter for all attributes but elevation, aspect, and slope |
| Car Evaluation | My method | StringToNominal on two attributes | StringToNominal on two attributes |
| Poker Hand | My method | NumericToNominal for all attributes | NumericToNominal for all attributes |
| Breast Cancer Wisconsin (Diagnostic) | My method and removed ID number as attribute because that has no bearing on whether a patient has breast cancer. | NumericToNominal for all attributes | NumericToNominal for all attributes |
| Internet Advertisements | My method | StringToNominal for first 3 attributes and NumericToNominal for the rest | StringToNominal for first 3 attributes and NumericToNominal for the rest |
| Contraceptive Method Choice | My method | NumericToNominal for all attributes but wife\_age and num\_children\_born | NumericToNominal for all attributes but wife\_age and num\_children\_born |
| Abalone | My method | NumericToNominal for the class label, rings | NumericToNominal for the class label, rings |
| Bank Marketing | My method | None | None |

To maintain consistency between how we tested our implementation of Naïve Bayes and how the Weka portion was tested, we tested directly on the training set across the board.

**Experiments - Experimental Results**

Results Table:

Note: For the sake of brevity, this table only shows percentage correctly classified. Running the code for our implementation will provide you with a confusion matrix as well.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Naïve Bayes (Our Implementation)** | **J48 Trees** | **1-Nearest Neighbor** |
| Connect-4 | 54.32% | 87.53% | 100% |
| Adult | 83.21% | 88.13% | 99.996% |
| Cover Type | 53.83% | --- | --- |
| Car Evaluation | 80.44% | 96.3% | 100% |
| Poker Hand | 14.6% | 71.93% | 100% |
| Breast Cancer Wisconsin (Diagnostic) | 97.8% | 95.85% | 100% |
| Internet Advertisements | 96.91% | 97.59% | 99.91% |
| Contraceptive Method Choice | 49.1% | 71.15% | 95.46% |
| Abalone | 16.1% | 75.72% | 100% |
| Bank Marketing | 79% | 94.12% | 100% |

**Brief Conclusion**

A notable observation in the results is the fact that the 1-nearest neighbor classifier is giving 100% or near-100% results for all the datasets. This is because we unwisely chose to test the trained model on the training set itself. For a classifier like nearest neighbor, which doesn’t explicitly build a model, but uses the data itself to decide, the outcome will be trivial if the training set is used as the test set. We realized this too late in our project, but at least now understand the foolishness behind testing it in this manner. We would have changed just the nearest neighbor test to something else like 10-fold cross-validation, but that would have made the results inconsistent since we used the training set to test all the other algorithms.

Another aspect of our results that I’d like to point out is that we were unable to properly run the cover\_type dataset in Weka. This is likely because this dataset is very large. It has 581,012 instances, 54 attributes, and 7 class labels. These three features being very large results in an incredibly slow build rate. I waited a few hours, but Weka was still unable to build this model.

The results of our algorithm are easy to justify, for the most part. For all but one of the datasets, our algorithm ran well. The one dataset that our algorithm did not work well with was the poker dataset. However, for other datasets, the low correctly classified percentage was to be expected. For example, the abalone dataset yielded 16.1% correctness with our algorithm. This makes absolute sense because many of the attributes in the abalone dataset are heavily related to each other. An abalone’s weight and length will be heavily dependent on the gender of the particular abalone. So, using Naïve Bayes on this type of dataset is foolhardy. Comparatively, abalone yields 75.72% with Weka’s implementation of J48 decision trees. This makes sense because decision trees are a good algorithm to use for any type of dataset. They are highly ubiquitous. In fact, this pattern of “lack of independence yields lower results” occurs quite a few times, and is a good way to generalize the behavior of our algorithm.

Finally, decision trees perform admirably across the board because due to the ubiquity of the decision tree learning algorithm. Decision trees are applicable across a wide variety of datasets.

**Contribution**

Pavan did the entire implementation of Naïve Bayes and the written report. Pakeezha completed the Weka testing and analysis. The split was roughly as follows: Pavan = 70% of the work. Pakeezha = 30% of the work.

**References**

* Link to the Cornell study: <https://www.cs.cornell.edu/~caruana/ctp/ct.papers/caruana.icml06.pdf>