CSCI 446 Artificial Intelligence Project 2 Final Report

Roy Smart Nevin Leh Brian Marsh

November 21, 2016

Abstract

Our project attempts to solve the classification problem of machine learning using four different algorithms. Each algorithm is tested on five datasets which train it to obtain enough knowledge to make an educated classification of new data points. After training, the algorithms were tested to classify a new data point, then measured on their precision and convergence for correct answers. The precision scores of the algorithms were comparable across all the datasets, with a clear leader being difficult to distinguish in any of the datasets.

1 Introduction

In a broad sense, machine learning refers to efforts to giving computers the ability to learn without explicit instructions. Machine learning algorithms (MLAs) have been shown to be successful at solving a wide range of problems. One common problem in this realm of machine learning is known as classification. Classification is a problem that can be solved using several algorithms, but the effectiveness of the algorithm depends greatly upon the specific dataset involved.

We implemented four MLAs to attempt to classify new data points into sets based upon previous information gained through training. The algorithms used are k-Nearest Neighbors, Naive Bayes, Tree Augmented Naive Bayes (TAN), and Iterative Dichotomiser 3 (ID3). Additionally, each algorithm is tested with five different datasets: the Wisconsin Breast Cancer Database, the Glass Identification Database, the Iris Plants Database, the Small Soybean Database, and the 1984 United States Congressional Voting Records Database. The effectiveness of each algorithm is measured by the metrics of precision and convergence.

2 Datasets

In this project we are asked to train our MLAs on five different datasets, each with a different set of challenges for our MLAs. For example, the soybean dataset is really small, the cancer database is reasonably large, and the attributes in the glass dataset are not very well correlated (as we will soon see).

A problem with some of the datasets is that the contain ID numbers, numbers that are different for every member in the dataset. These are problematic for our algorithms as the ID number does not provide any useful information, and it hurts the predictive power of our algorithms. Therefore we have edited out all ID numbers from the datasets under the assumption that any researcher attempting to solve real-world machine learning problems would also ignore the ID numbers.

2.1 Discretization

Two datasets, the iris dataset and the glass dataset have continuous values. We are asked to discretize these dataset to more accurately compare the different MLAs. For our project, we adopted *binning* as our discretization scheme. Binning is a simple method for discretization that defines a certain number of adjacent intervals, known as bins, and then replaces points that fall within each bin by a value representative of that bin.

For the data in this report the number of bins was set to 10. In principle, the ideal number of bins in terms of MLA precision could be any integer larger than one, but through tests we found that the ideal number of bins is proportional to the number of classes in the dataset.

2.2 Cross-Validation

Cross-validation is statistical bootstrapping techinque used to estimate the generality of a statistical model using a dataset independent of the dataset used to create the model. In the context of machine learning, this means that we will partition the full datasets into training and validation datasets. For this project we are asked to use the so-called k-fold cross validation technique, where the full dataset is partitioned into training and testing datasets through the use of *folds*. The folds are found by splitting the dataset into k equal sized pieces, and assigning each piece to be a fold. One fold is selected as the validation set, and the rest of the folds are assigned to the training set. This process is then repeated k times, where each fold is taken to be the validation set once.

To further increase the accuracy of our cross-validation process, we will *stratify* the folds described by k-fold cross-validation technique. Stratification in this context means that each fold will have an equal distribution of the classes in each dataset.. This process makes cross-validation more accurate, because it gives each MLA adequate information on each class for every training dataset. Without stratification, some training datasets could give a MLA zero examples of a particular class, hurting the MLA's ability to generalize.

In k-fold cross-validation, k is a free parameter that can be tuned by the user. For this project, we have selected $k = 10$, because it will allow us to perform the convergence tests discussed in Section 7 over a larger range.

2.3 Missing Values

Some of the datasets have missing values. This is problematic since most of the MLAs will yield vastly different answers if some data is missing. However, we note that the appearance of missing values in the voting dataset is merely an illusion, missing values in this dataset can signify a stance on a particular issue and this information should be used by our algorithms.

Since we have determined that the voting set contains no missing values, the only remaining dataset with missing values is the cancer dataset. Since there are so few missing values and the dataset is so large, we have decided to simply delete any data with missing values from the cancer dataset. Therefore, we have found that data imputation is not necessary for this project.

3 k-Nearest Neighbors

3.1 Training

K-Nearest Neighbors attempts to solve the classification problem by using already-known data points with similar characteristics to make an educated guess of the class. The training for k-Nearest Neighbors involves simply reading in all of the dataset and storing these records as points based on their values.

3.1.1 Constructing Probability Table

3.2 Validation

3.2.1 Value Distance Metric

3.2.2 Determining k and p

The effectiveness of the k-Nearest Neighbors algorithm depends heavily on the values used for the number of neighbors (k) and the power used in the distance equation (p). Our test used varying values of k between 1

and 20 and of p between 1 and 9. While the precision of the algorithm with regards to the values of k and p was highly variable for different datasets, overall, higher precision was generally found with higher values of p. The value of p had a lesser effect, with very low values seemingly being favorable.

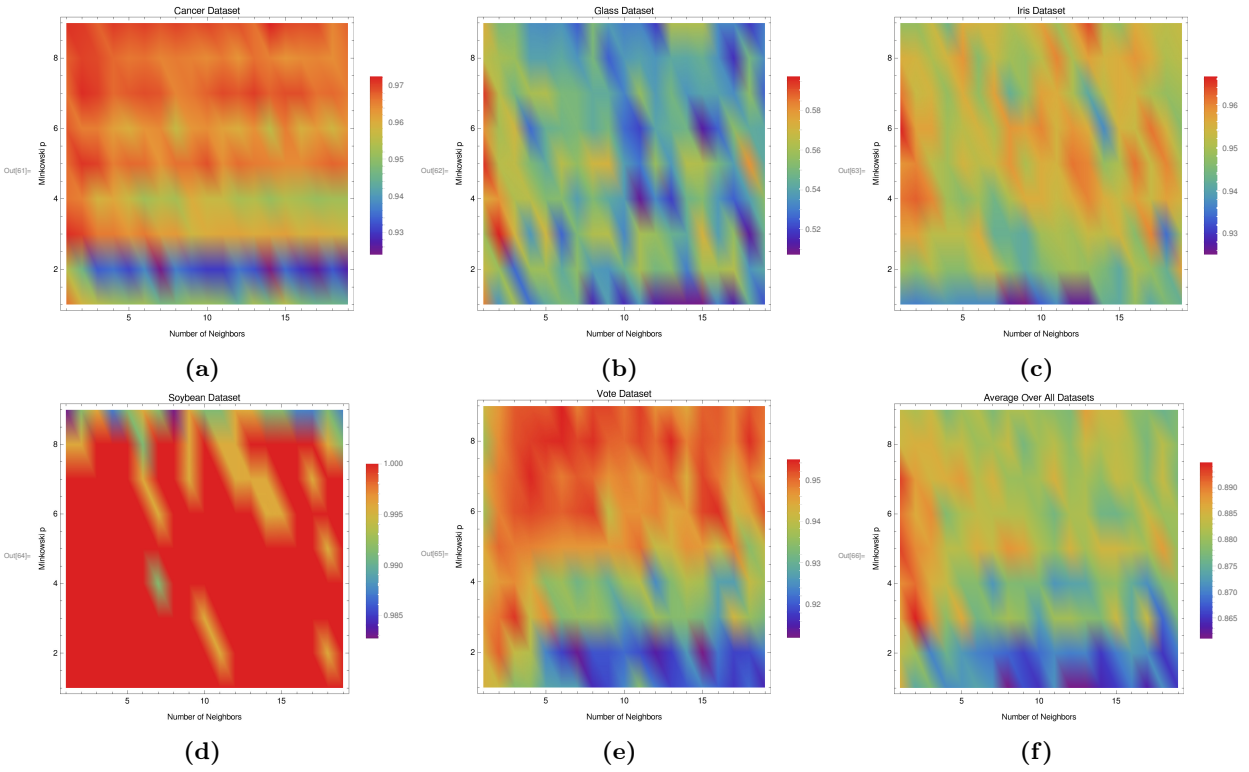

Figure 1: *A series of plots demonstrating the precision (in color) of k nearest-neighbors for different values of k and the p value in the Minkowski distance. We have plotted the response of kNN for all five datasets (Figure 1a - 1e) and the average response over all datasets, Figure 1f.*

4 Naive Bayes

Naive Bayes uses the principle of Bayesian learning to solve the classification problem. This algorithm is said to be naive because it considers the attributes to be conditionally independent when performing classifications. Under this scheme, the probability distribution of each class C is given the set of attributes $x_1, ..., x_n$ is written by [Russel and Norvig, 2010] as

$$\mathbf{P}(C|x_1, ..., x_n) = \alpha \mathbf{P}(C) \prod_i \mathbf{P}(x_i|C) \tag{1}$$

where, using the maximum likelihood hypothesis, $\mathbf{P}(C)$ is the prior probability of the class C in the training dataset, $\mathbf{P}(x_i|C)$ is the likelihood of attribute x_i given C, and α is a normalization constant. To find $\mathbf{P}(C)$ the we simply say it is equal to the proportion of C observed in the training dataset. Similarly, we can calculate the likelihood using the definition of conditional probability

$$P(a|b) = \frac{P(a \wedge b)}{P(b)}$$

where the probabilities are calculated using the number of occurrences in the training dataset.

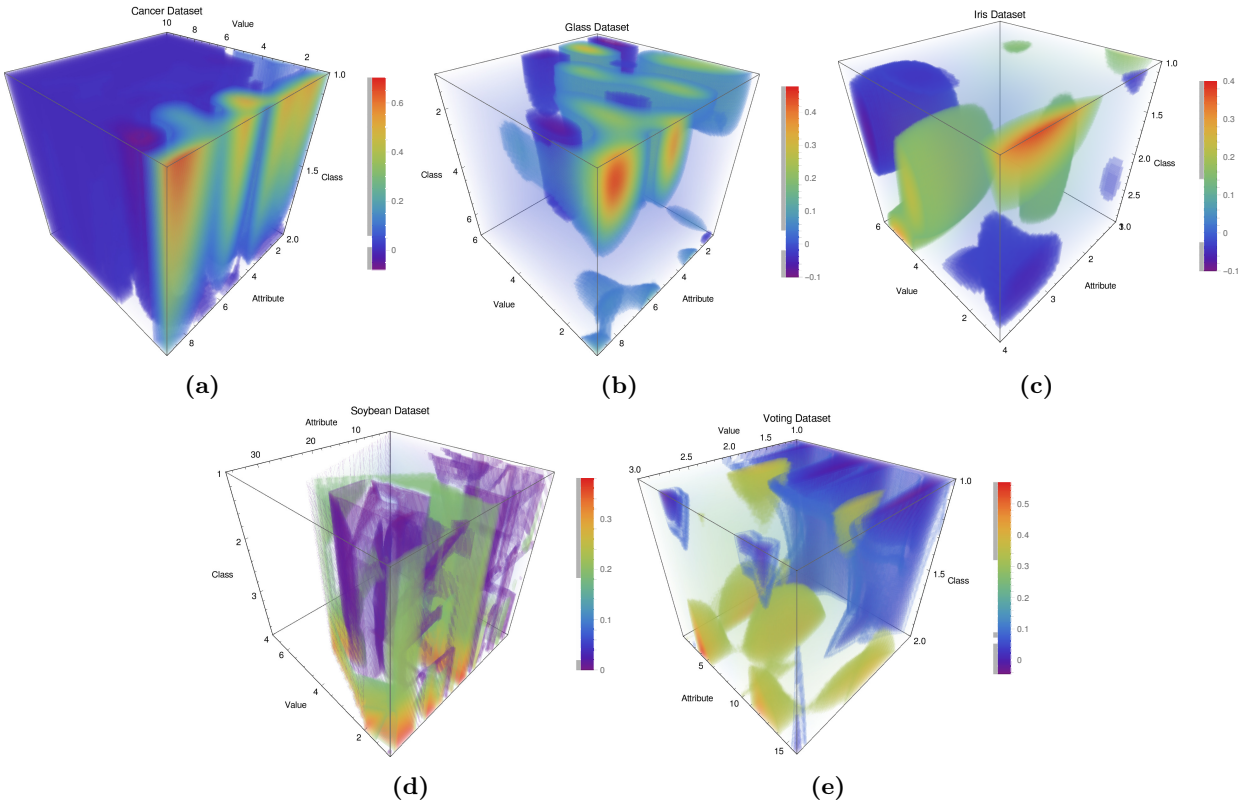

Figure 2: *3D density plots of the naive Bayes probability table for all five datasets. The vertical class axis represents the value of the class, c, the attribute axis denotes the index of an attribute, x, and the value axis represents the vale of that attribute, a. Finally the color is equivalent to the probability of the attibute x having value a, and the class having the value c*

4.1 Training

During the training phase for naive Bayes, the algorithm simply counts the how many times a specific attribute takes on a particular value for each class. From these counts we build the probability table visualized in Figure 2. In Figure 2, we have plotted the class vs. attribute index vs. attribute value vs. probability (in color) for each of the five datasets. These plots are interesting because they serve to visualize the relationships between classes and attribute values.

If we contemplate the plots in Figure 2, we can start to visualize how Naive Bayes works. For the cancer dataset in Figure 2a, we can see how benign (class = 1) results are characterized by low values for every attribute. In the glass dataset, Figure 2b, the low classes (building and vehicle windows) are very well characterized by the attributes, while the miscellaneous glass types (containers, tableware, headlamps) were not dependent upon the attributes. In the iris dataset (Figure 2c), we can see beautiful regions that characterize each species. The soybean dataset in Figure 2d demonstrates a probability table with an interesting geometric structure, while in the voting dataset, Figure 2e, it seems almost as if we can see the party line between Democrats and Republicans.

4.2 Validation

To validate our naive Bayes classifier, we simply use the probability table discussed in Section 4.1 and Equation 1 to find the most probable class. To evaluate Equation 1, our implementation looped over the classes to construct the product of the conditional probabilities. $P(C)$ was calculated by taking the total occurrences of each class and dividing by the total size of the dataset. $P(x_i|C)$ was found by taking the

number of times an attribute i with a specified class C had value x_i divided by the total number occurrences of class C;

5 TAN

Tree-Augmented Naive Bayes (TAN) is an extension to the naive Bayes algorithm described in Section 4. Developed by [Friedman et al., 1997], TAN relaxes the assumption that the attributes are conditionally independent and allows each attribute to depend on only one other attribute. It accomplishes this by constructing a fully-connected, undirected graph out of the attributes in the dataset. TAN then assigns a weight to each connection in the graph using the *conditional mutual information function*, given by

$$I_P(\mathbf{X};\mathbf{Y}|\mathbf{Z}) = \sum_{\mathbf{x},\mathbf{y},\mathbf{z}} P(\mathbf{x},\mathbf{y},\mathbf{z}) \log\left(\frac{P(\mathbf{x},\mathbf{y}|\mathbf{z})}{P(\mathbf{x}|\mathbf{z})P(\mathbf{y}|\mathbf{z})}\right).$$

Applying the this weight to the edges allows us to construct a maximum spanning tree of the graph. We can then make the spanning tree directed by choosing a root node and setting the direction of all edges to be outward from it.

Using this representation, we can calculate the analog of Equation 1, the probability distribution, using the expression given by [Zheng and Webb, 2010]

$$\text{class} = \text{argmax}_C \left[P(C)P(x_r|C) \prod_{x_i,x_j} P(x_j|x_i,C) \right] \tag{2}$$

where x_i is the parent of the attribute x_j in the directed maximum spanning tree and x_r is the root node of the spanning tree.

5.1 Training

To accomplish the TAN algorithm, we will need to construct a probability table similar to the one described in Section 4 on naive Bayes. For this algorithm however, the 4D (class, attribute index, attribute value, count) probability table of naive Bayes needs to be extended to a 6D probability table where the dimensions are: class, attribute 1 index, attribute 1 value, attribute 2 index, attribute 2 value, count, to represent probabilites of the form $P(x_i, x_j, C)$, the probability of attributes i and j having the values x_i and x_j and the class value C. Unfortuantely, since the dimensionality is so high, constructing plots to visualize the probability table such as Figure 2 is impossible.

Once we have iterated through the dataset and constructed our 6D probability table, we can construct a fully connected graph between the attributes using the weight provided by Equation 5. With this fully connected graph, we can use Kruskal's algorithm to construct our maximum spanning tree. This algorithm is best described in the words of Kruskal himself,

> Perform the following step as many times as possible: Among the edges of G not yet chosen, choose the shortest edge which does not form any loops with those edges already chosen. Clearly the set of edges eventually chosen must form a spanning tree of G, and in fact it forms a shortest spanning tree [Kruskal, 1956].

To implement Kruskal's algorithm, we started with our fully connected tree weighted by the conditional mutual information functon. Next, we sorted the weights by descending weight, to ensure we end up with a maximum spanning tree. At this point, the fast way to implement this algorithm would be to use a disjoint set data structure, to detect loops while building the MST. We did not use a disjoint set data structure, instead we detected loops by doing a complete depth first search through the tree while testing if each node had already been visited by the DFS. If a node was visited twice, the graph contained a loop.

Using our loop detection function, we were able to build MSTs describing the "largest" dependence of one attribute on another. With the MST complete, TAN is ready for validation.

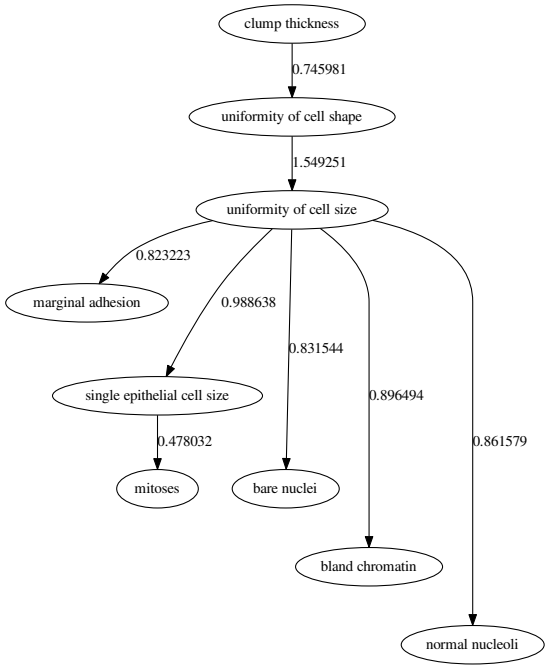

Figure 3: *Example of the maximum spanning tree produced by TAN on the cancer dataset.*

5.2 Validation

The validation phase of TAN is similar to the validation of naive Bayes described in Section 4.2. The only difference is that we use the MST built in preceding section to determine the parent of each node. We then used Equation 2 and our 6D probability table to determine the most probable class.

6 ID3

ID3 is an recursive process that generates a short tree by splitting the tree one attribute at a time. The first attribute to be used is the one with the most information gain. Information gain can described as how much entropy the system lost by splitting on an attribute. The equation for entropy is as follows:

$$H(S) = \sum_{i=1}^{n} \left(-p_i \log_2 p_i\right).$$

where p_i is the proportion of the number of datums in class i to the total number of datums in the set S. To use this equation the entropy of the whole system is determined first. Then the sum of the entropy for each new branch is calculated and subtracted from the total entropy. We try to branch on each attribute that has not been used yet. Since the goal is to maximize this difference so the tree will be small and more general, we choose the attribute that resulted in the greatest reduction of entropy. The reduction of entropy is defined as information gain. The rest of the algorithm is just iterating through the tree as it is made and recursively performing the calculation on an attribute that has not been split yet. The tree is finished when all leaf nodes are a single class or when attributes to split on have run out. In this case each leaf node that does not have a class is assigned the most common class in that leaf.

6.1 Training

We closely followed the decision tree learning psuedocode in Figure 18.5 in [Russel and Norvig, 2010] to construct our tree. The three base cases discussed above were implemented with two methods that calculated plurality value and whether a node only contained one class. The else statement calculated the gain of each value in the current node. It then splits the current data set by variable value and recursively calls itself with the new sets as input.

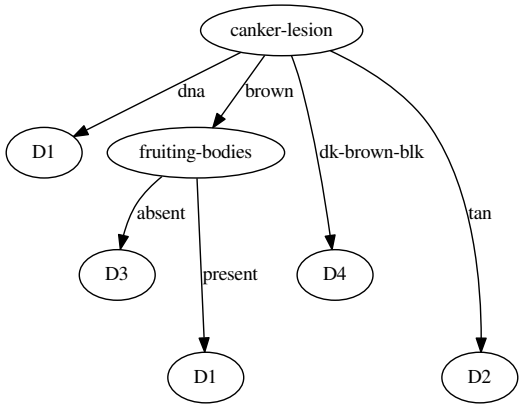

Figure 4: *An example ID3 tree generated for the soybean database.*

We implemented reduced error pruning to help make the tree more general. The approach was to split the training set into a smaller training set and a validation set. The tree was then generated as usual using the new training set. We then removed non-leaf nodes one at a time, checking how well they performed on the validation set. If they performed better, the nodes remained removed. Otherwise they would be restored to their previous state.

6.2 Validation

The process of classifying a datum according to the tree is a simple process. The tree is traversed according to the values of each variable in the datum. When a leaf node is reached, that leaf's class is returned as reiterated in [Quinlan, 1986]. It is easy to determine leaf nodes because they lack children nodes.

7 Results

To show the relationship between training set size and precision we created the graphs seen in Figure 5.

	Cancer	**Glass**	**Iris**	**Soybean**	**Vote**	Average
kNN	0.953154	0.579221	0.96	1	0.942389	0.8869528
NB	0.975085	0.649783	0.95333	1	0.903436	0.8963268
TAN	0.961893	0.66342	0.96	0.98	0.944767	0.902016
ID3	0.950171	0.662771	0.98	0.975	0.95629	0.9048464
Average	0.960	0.639	0.9633325	0.98875	0.9367205	

Table 1: *1x10 Cross Validation Precision. Green represents the best performing algorithm for each data set. Red represents the worst.*

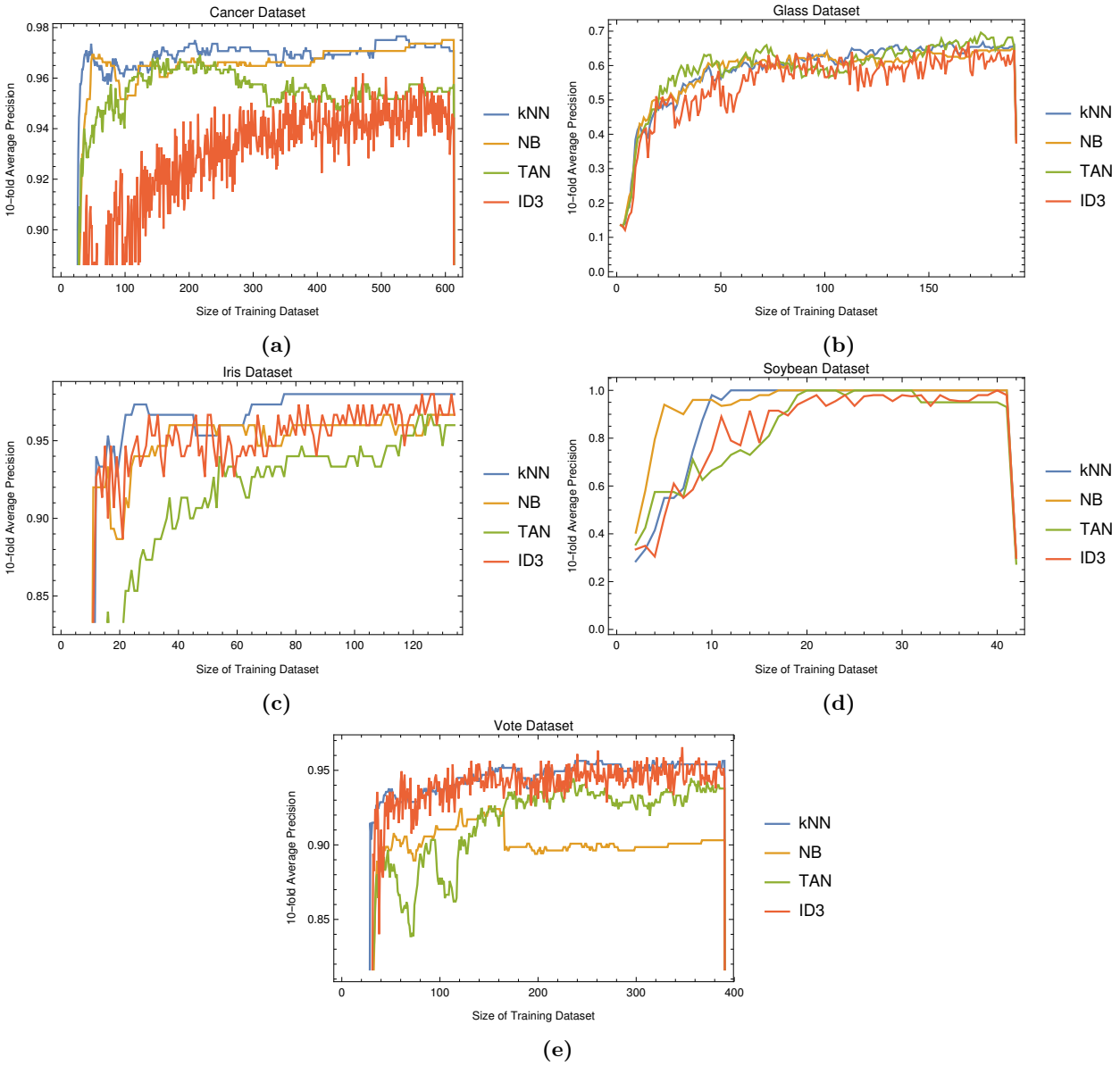

Figure 5: *Plot showing the learning curves for each of the algorithms on the five datasets. Please note the low point on the right hand side of Figures 5a, 5b, 5d,5e is in error and is not part of our results.*

7.1 Cancer Dataset Performance

In the cancer dataset, kNN and naive Bayes obtained very similar results in terms of precision. We can see that kNN had the fastest learning curve, staying dominant until the training set was larger than 400 items where naive Bayes finally caught up. The results from TAN were mildly suprising since the algorithm had a maximum in accuracy at a training set size of 150. ID3 performed poorly on this dataset compared to other algorithms, displaying a slow training time and precision lower than that of TAN.

7.2 Glass Dataset Performance

The glass dataset was easily the most difficult of the five datasets for all algorithms. However, TAN did really well on this dataset, displaying the fastest learning curve and nearly tied for the best in final precision with a

value of 66%. kNN and naive Bayes did similarly, to the other two algorithms (which was still not great).

7.3 Iris Dataset Performance

The iris dataset was a fairly easy dataset for most of our MLAs to solve. kNN did the best, consistenty getting perfect classifications with training sets larger than 80 members. Naive Bayes and ID3 had approximately the same performance on this dataset, displaying a similar learning curve to kNN, but failing to keep up compared to kNN as the training sets increased in size. TAN did exceptionally poorly on this dataset, in having a much slower learning curve than the other three algorithms.

7.4 Soybean Dataset Performance

As previously mentioned, the soybean dataset was the smallest out of the five datasets. Therefore the validation dataset was very small for 10-fold cross-validation, which may have made our algorithms appear more precise than reality. Nonetheless kNN and naive Bayes did the best on this dataset, having the fastest learning curves and perfect accuracy on the full training dataset. ID3 and TAN appear to have slower learning curves, but the accuracy of these algorithms was still very high on the full dataset.

7.5 Voting Dataset Performance

The voting dataset was the best performance for the ID3 algorithm, although kNN still displayed very similar performance. Most surprising about this dataset is the performance of naive Bayes, which started off with a similar learning curve to ID3 and kNN, but settled at a local maximum at a training dataset size of approximately 130 members before taking a sharp dive in precision at about 160 members in the training set. TAN showed an almost opposite precision response, with poor accuracy between 50 and 120 training dataset members and better with increasing training set size.

8 Conclusion

In this paper we have explored the implementation and performance of four machine learning algorithms: k-Nearest Neighbors, Naive Bayes, TAN, and ID3. We successfully gauged each algorithms precision on five datasets using stratified cross-validation. By analyzing how each algorithm performed on each dataset, we were able to tentatively make some conclusions as to what type of data set each algorithm excelled at. Overall, the results were extremely close and there was no single algorithm that outperformed the rest.

References

[Friedman et al., 1997] Friedman, N., Geiger, D., and Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning*, 29(2):131–163.

[Kruskal, 1956] Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50.

[Quinlan, 1986] Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

[Russel and Norvig, 2010] Russel, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Pearson Education, Upper Saddle River, New Jersey 07458, 3rd edition.

[Zheng and Webb, 2010] Zheng, F. and Webb, G. I. (2010). *Tree Augmented Naive Bayes*, pages 990–991. Springer US, Boston, MA.