Abstract:
As the speed of data production increases, automated systems are gaining ever more importance in everyday life. This report looks at four of the most common algorithms: Decision Trees, Random Forests, Neural Networks, and Naïve Bayes. In doing so, it compares their performance on four data sets which represent a variety of conditions and challenges. The algorithms were graded based on accuracy, recall, speed. This report found that one average Random Forests were the most accurate and best resisted skew, however, they were quite slow. Neural Networks were remarkably close in accuracy, but far more sensitive to bias in the datasets. Additionally, Neural networks were significantly faster to predict than RF. Decision Trees and Naïve Bais both performed significantly worse overall for accuracy, however, in certain situations they could perform at a comparable level into the front runners in a fraction of the time.

Introduction:

In 2010, it was estimated that the world produced or replicated 2 zettabytes of data, or 2,000,000,000,000 gigabytes. By 2022, that number has jumped to 97 zettabytes, an increase of almost 50 times. While these figures do not reflect the amount of data saved,[1] They do illustrate an important trend in the digital age: Data is growing far faster than humans' ability to interpret it. Considering this shift, new methods of machine-based data processing and analysis have become increasingly important to everyday life. Termed Machine Learning, these algorithms allow us to exploit the ever-increasing availability of processing power to find previously hidden connections between minute quirks in the data. They allow us to recommend new songs or shows, to predict earthquakes far in advance, and even can assist in life saving diagnoses.

In this paper, we will look at four implementations of popular algorithms, neural networks, decision tress, random forests, and naïve-baye to gauge their applicability in a variety of contexts. They will be assessed against four different datasets representing various goals and challenges. These include labelling a given robot, labeling the digit shown in a picture, predicting a congressperson's party given a voting record, and classifying the health of a patient's thyroid gland.

The ML algorithms were evaluated on accuracy, recall, and speed. After combining the results from multiple runs, it became clear that no single method outperformed all others in every metric. There were a variety of tradeoffs, and some were just not feasible for given problem types. Overall, choosing between them is heavily context based.

Problems and Data sets:

The first data set used in this experiment was *monks1*. This was originally designed in 1991 as a benchmark for international comparison of learning algorithms.[2] The data is entirely categorical with each instance having 6 attributes and the labels being a binary 'yes' or 'no'. There is a set rule for labeling where if (attribute[4]=='red') or (attribute[0]==attribute[1]) then the 'yes', else 'no'. Additionally, there is no noise in this dataset, meaning that ideally from an end-user perspective, we would want all algorithms to be able to reach 100% accuracy.
Following this we have the *mnist_1000*[3] which is an optical recognition dataset. It represents a subset of the *MNIST* database which is itself a subset of the larger *NIST* database and represents one of the major international benchmarks for comparing ML algorithms. *mnist_1000* is a dataset of 10,000 training instances which each instance being vector of dimensions 1x784 which each element ranging from 0-255. The vector represents a flattened grayscale photo of resolution 28x28. These photos were of handwritten digits from 'zero' to and including 'nine.' Overall, this data set presents a multiclassification problem with numeric data. http://yann.lecun.com/exdb/mnist/

The third dataset, entitled *votes*, is aimed at getting learning algorithms to predict a congressperson's party given their voting record. It is based on the 2nd session of the 98th Congress, occurring in 1985. It is like the monks1 dataset in that it is again a binary classification problem, with labels 'democrat' and 'republican', using categorical values for its attributes. However, unlike *monks1*, *votes* has a slight skew to its labels with democrats being more represented than republicans. Additionally, whereas past data sets have been complete, *votes* has a degree of missing data. Of nine different actions that a Congressperson could take in a vote, three were simplified to 'yes,' three were simplified to 'no' and finally three were simplified to '?'. These last three, "voted present, voted present to avoid

[1] Estimated at ~2% of total data produced or copied

[2] https://archive.ics.uci.edu/ml/datasets/MONK%27s+Problems

[3] On the paper it says that the dataset provided is the mnist_100, however, I believe that the mnist_1000 is what was in GitHub and so I'm unsure of which was intended to be used. I apologize if this was not the set you wanted analyzed

conflict of interest, and did not vote or otherwise make a position known," do not cleanly collate to a specific party, at least to the same degree as 'yes' and 'no.'

The final data set used was *hyperthyroid*, another multilabel classification problem only this time with mixed continues and categorical attributes. Each instance includes a variety of medical measurements/observations related to the patient and one of four labels for that person's thyroid.

Data Set	Dimensions	Dimensions after one hot encoding	Data Type	Label Type	Labels Proportions Number: (Proportion)
monks1	(432,7)	(432,12)	Categorical	Binary	2:(50,50)
mnist_1000	(10000,785)	(10000,785)	Continuous	Multiclassification	10:(10,…,10)
votes	(435,7)	(435,16)	Categorical	Binary	2:(61.4, 38.6)
hyperthyroid	(3772,28)	(3772,32)	Mixed	Multiclassification	4:(92.3,5.14,2.52,.004)

Algorithms:

The first algorithm we tested was a Decision tree with the first widely used implementation of this dated back to 1986. Ross Quinlan developed the Iterative Dichotomiser 3 (ID3) algorithm that is the basis for most modern implementation. The algorithm was based on the idea that using an equation called Shannon's entropy, one could measure the disorder of a data set. Using this, the ID3 algorithm would take in an initial data set of categorical variables and split in into subsets based on their attribute for a given column. So, all data with Attribute[0] =='A' would form one data set, all data with Attribute [0] =='B' would form a second data set, all data with Attribute [0] =='C' would form a third data set and so on and so forth. Then, the computer measures the entropy/disorder of all the newly created data sets and subtracts that from the first to get information gain. It then goes through and tries this with all the other columns. The split that produces the most information gain is the winner of this round and is stored for later. Then the tree algorithm is recursively preformed on each of the new datasets until either all the labels are the same, at which point you have reached the end of the tree, or that there is no more splitting possible in the dataset, and in this case the computer remembers the most common label. Once the tree has reached this stage for all datasets, it is done training. At this point, the algorithm knows what column to split on for round 1, then round 2, and so on and so forth until it reaches a leaf. Once there, it looks to see what label was stored and that is its prediction.

This has been updated in several ways, since inception, however, this paper will focus on Classification and Regression Trees (CART) as they were used in implementation. This algorithm leaves ID3 alone[4] and expands the algorithm to support continuous data by converting values into categories via a chosen threshold. So, (60, 65, 75,80) can be mapped to a 'A' and 'B' category via if(value>Threshold) A, Else, B. Like with categorical data, the algorithm tries a variety of thresholds based on the data and choses the one that maximizes information gain.

The next algorithm we tested is referred to as random forests. As the name suggests, random forests are created by making several decision trees, letting them vote for the label, and then choosing the most common as the answer. This process produces a series of uncorrelated models[5] which are desirable for extracting the most information possible from a data set. One important part of building these models is that due to Shannon's Entropy equation, decision trees are produced in a deterministic way given a specific data set. This means that to produce a random ensemble of learners, each tree must receive a random data set. This can be done in two, non-exclusive ways. The first being to sample with replacement from the original data set, and the second being to restrict the Attributes each tree can see. Together, these two methods can produce a variety of vastly different trees that are each sensitive to different relationships in the data.

The third algorithm used in this experiment is the most well-known: Neural Networks. Dating back to 1944, this class of algorithms was named for its similarities to the neuron-based structure of a human brain. Like the brain, it is based on several nodes or neurons that each take in some numeric input, apply a function (most

[4] It does switch Shannon's Entropy for a new equation called Gini impurity; however, the specific mathematical differences are beyond the scope of this paper. As an interesting aside, this same equation is sometimes used to evaluate the income inequality of a country

[5] Importantly these models are not fully uncorrelated and the degree of uncorrelation is related to how the data is split in the manners discussed further on

commonly logistic or Relu), and output some value that will later be the input to another neuron. This process continues until we reach the terminal or output neuron(s) which also produces a numeric value that can either be used as it is for regression tasks or mapped to a label for classification. This makes neural networks an incredibly powerful tool that can be applied to almost any situation, however, as will been seen in the experiments, there are drawbacks.

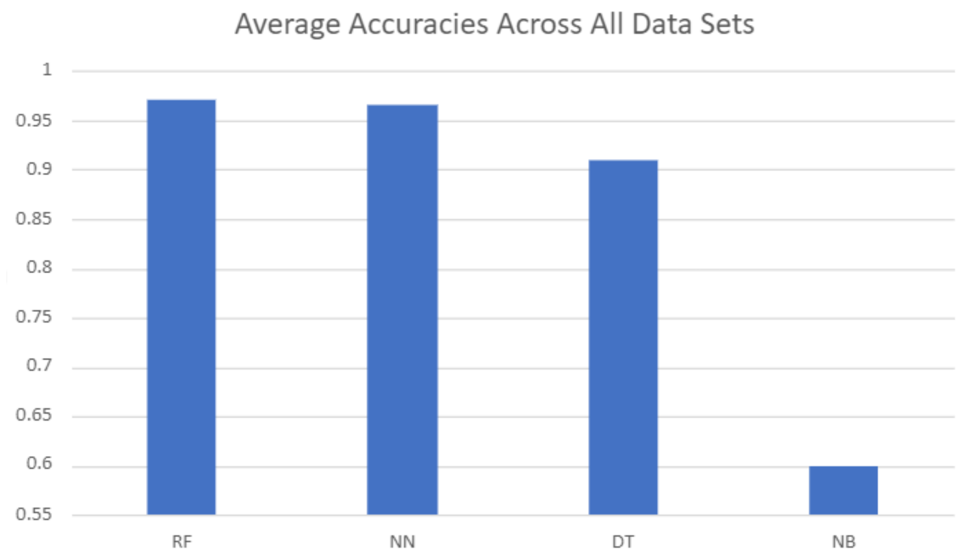
The fourth and final machine learning technique that was examined is called Naïve Bayes. This algorithm is based on a statistical theorem predictably termed Baye's Theorem which describes the probability of an outcome occurring, given the current state of a system. The algorithm is then prefixed with Naïve because it makes two crucial assumptions, 1. that all attributes in each instance are independent from one another, and that all attributes are equally important is predicting a given label.

Experimental Setup:

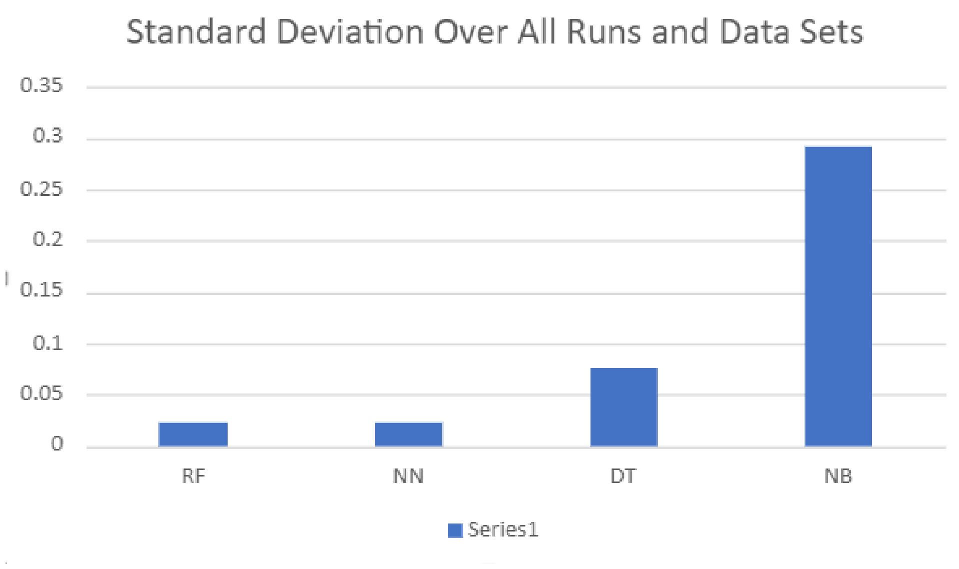
All four algorithms were implemented in Python using the well-known scikit-learn library. Besides seeding random states where applicable, all hyperparameters were left to their default values. This decision is not without substantial implications that will be looked at later in the conclusion.
Each technique was graded on three primary metrics: Accuracy, Recall, and Speed. The first is exactly what it sounds like being calculated via correct labels/total labels for the validation set. Recall looks at how the program performed relative to a specific label, I.e., how often did it correctly identify a compensated hypothyroid from the hypothyroid data set. Finally, speed was measured via the process_time() function from Python's time library. It is broken down into two measurements, time to train and time to predict. Intuitively, this measurement varies wildly based on the context in which the program was run and as such will be used in a comparative manner.
The actual experiment consisted of five runs using seeds 0-4. In each run, the program would read in the first csv file as a pandas' data frame, shuffle that based on the random seed, and then split it along an 80/20% divide to create the training and test sets. These were then provided to each ML algorithm, the accuracy, recall, and speed of which would then be recorded for later analysis. This continued until all four csv files had been tested on.

Results:

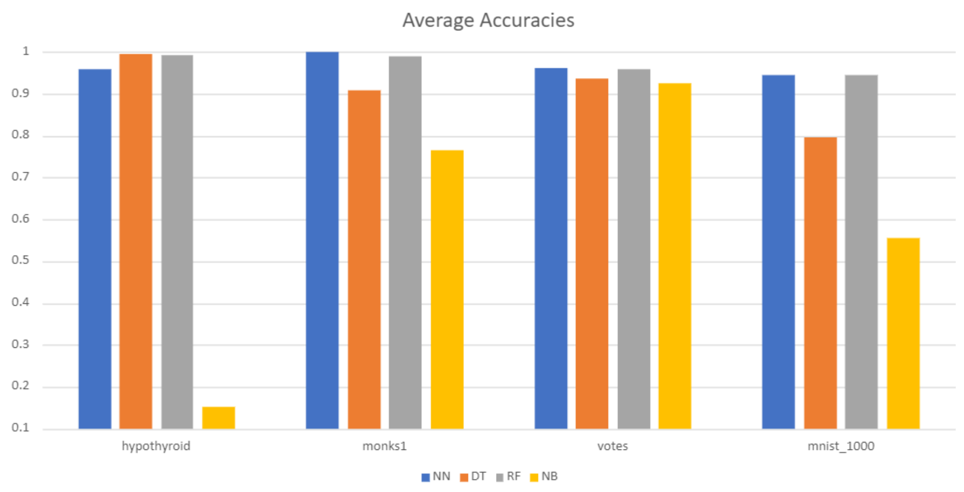
The combined accuracies over all data sets average over five runs for each ml algorithm is recorded on Image 1represents how robust or widely applicable each algorithm is versus a variety of tasks such as binary or multiclassification and with both categorical and continuous data.



The graph shows three primary groupings, firstly Random Forests and Neural Networks which are almost tied at 97.2% and 96.6% respectively. Secondly there is Decision Tress which takes a considerable drop to 90.9%. Finally, there is Naïve Bayes which absolutely lags at 60.0%. However, this broad aggregation covers up lot of minutiae with actual implementation. We can see this by looking at the standard deviations associated with the averages above.



As we can see, whereas the first three algorithms are within 5%-10% in any given run, Naïve Bayes varies by almost 30% per run! So, let us break this down a little more, and look at average performance relative to each data set.



Here we can see that while Naïve Bayes still comes in last place for any given data set, there is a stark difference in performance between its highs (92.6% on votes) and lows (15.2% on hypothyroid). An explanation for this is the types of tasks at hand.

Recall from earlier that Naïve Bayes makes two important assumptions, 1. that the data attributes are independent of one another, and that all attributes are equally important. This context is particularly well suited to Votes as it is a summary of 6 key votes, meaning that all attributes are similar in their significance, and, as there was a broad range of issues, the vote cast for one motion should not be impactful on another vote.[6] This stands in contrast to hypothyroid in which the data represents measurements from the human body it is fundamentally and significantly connected. Additionally, certain attributes are much more significant in that data set such as whether

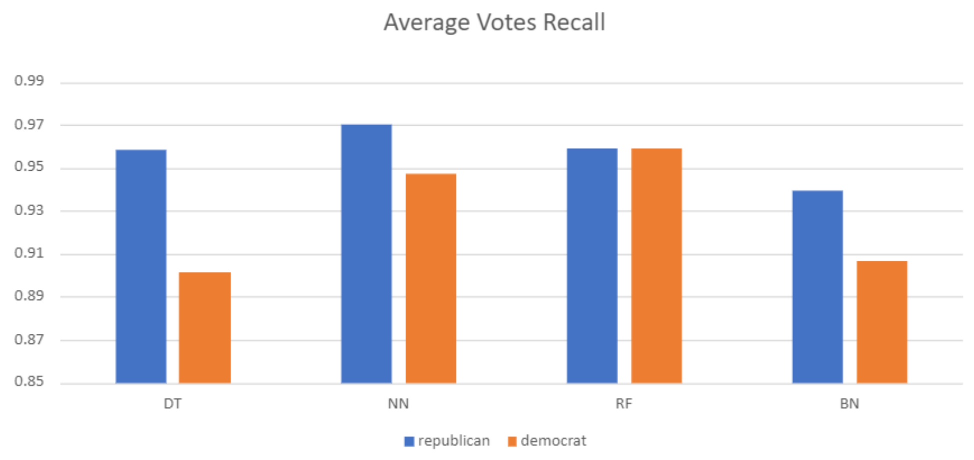
[6] This is not to say that there is no causation between attributes, just that it is not as overt as in many other cases. The lack of consideration for this correlation may be at least in part responsible for why Naïve Bayes still preformed worse than the other algorithms

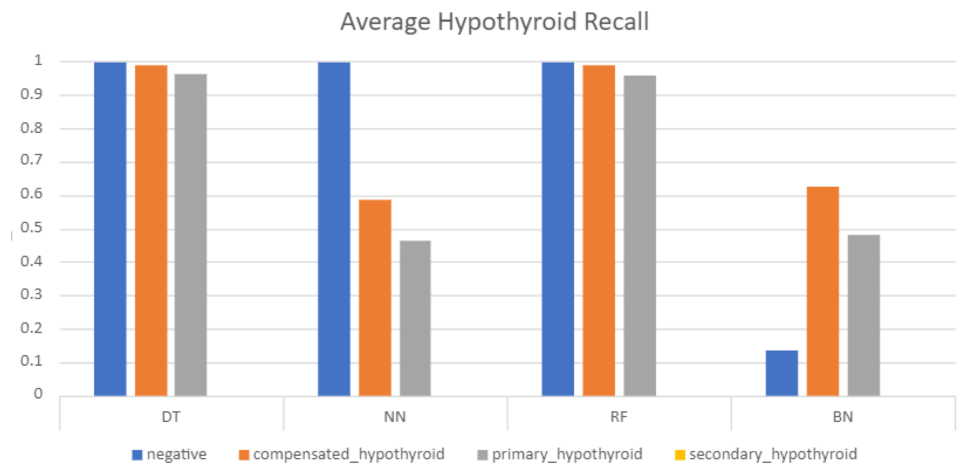
the patient is actively on an antithyroid medication compared referral source. The final nail in the coffin is that Naïve Bayes typically performs best with categorical data rather than continuous.

Overall, while Naïve Bayes is not as robust as the other algorithms, it still can achieve comparable results when used in the correct situations, and importantly, significantly faster. On votes, Neural Networks took on average 1.469 seconds to train and 0 to execute whereas Naïve Bayes took 0 to train and execute. This affect scales too, with mnist_1000, NN took 70.234 seconds to train and 0.203 seconds to execute whereas NB took 0.219 to train and 0.188 to execute. Of course, most of this is due to training, but the point remains that NB can get great accuracy at blazing speeds when used in the right context.

Continuing to our other algorithms, we see predictable results with Decision Trees where they underperform their more complex cousin Random Forests. Here we find a similar, but much less pronounced, situation to NB. While they are less robust than the front runners, in the right context a DT can perform as well or even better than a RF or NN for a fraction of the computational cost.

With this in mind, we can now move onto the flagship algorithms, Random Forests and Neural Networks. As we saw in the earlier graphs, both preformed very similarly to one another in both aggregated accuracies and by data set, therefore, it is important to consider their Recall.





When looking at the two data sets that had biases in their labels, we see that NN was impacted significantly more than RF by that skew.[7] This is of concern for general usage as typically we are interested in most or all the labels equally, specifically with the hypothyroid dataset, an algorithm that misses most ~½ of all positive diagnosis is not much use to doctors, despite its success with negative cases.

[7] There are of course solutions to these problems, but those will be discussed in the conclusion

Finally, a crucial factor is the speed difference between these two algorithms. RF vastly outperforms NN when training with NN averaging 70.234 on mnist_1000 and RF averaging 4.453. However, during execution NN took the lead with an average of __ compared to __ for RF. This means that while the overall time take vastly favors RF, in normal use training an algorithm is needed very rarely compared to execution. So NN could easier be faster than RF in total time in the long run.

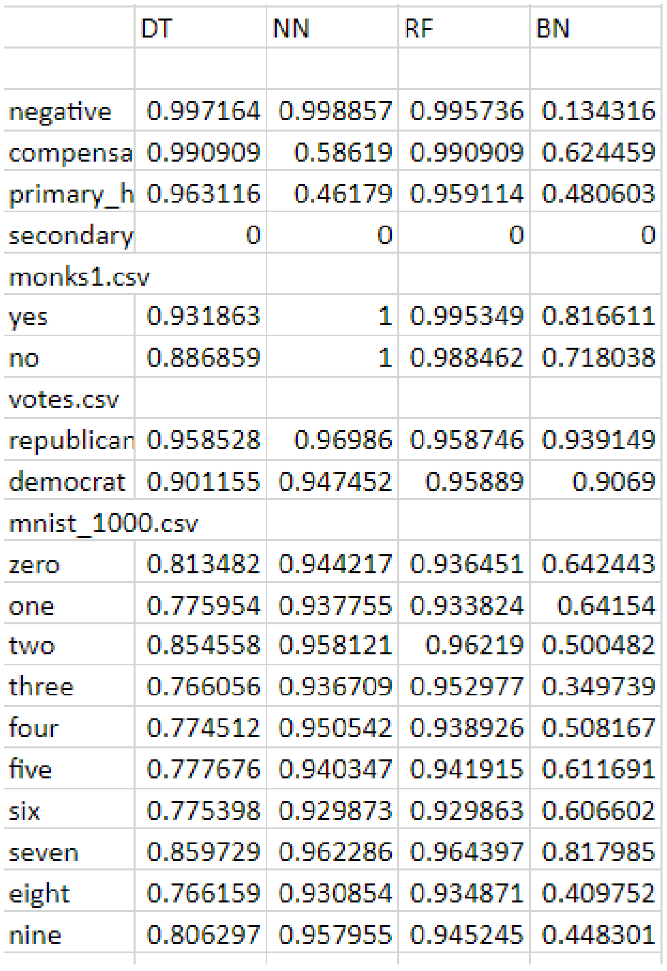
Conclusion:

Overall, of the four machine learning algorithms test, Random Forests performed the best in average accuracy and was the most resistant to data set skew. After that came Neural networks which, while slightly less accurate, have significantly faster execution times and were prone to bias in the labels. After that Decision Trees came in ~5% behind the front runners but given the right circumstances could even outperform them. Significantly, this algorithm was also substantially faster than either of the prior ones. Finally, Naïve Bayes, while extremely fast, struggled to keep up in most contexts.

Importantly, none of these algorithms were optimized for the data sets at hand. Instead, their stock implementations were relied upon due to time constraints. Further research is required to see if these results change significantly when time is spent tuning the algorithms hyper-parameters. Additionally, as training and execution was performed via CPUs, however, increasingly GPUs and TPUs are being used to speed up algorithms via parallelization, which may benefit some algorithms more than others, increasingly altering the results.

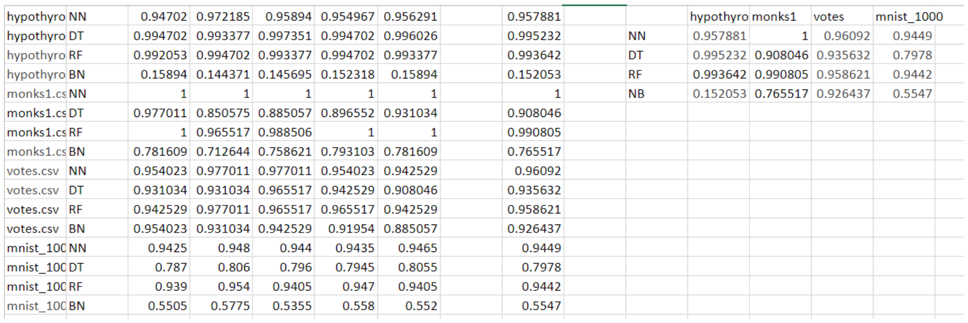
Appendix

Table 1: Average Recall over five runs by label and dataset



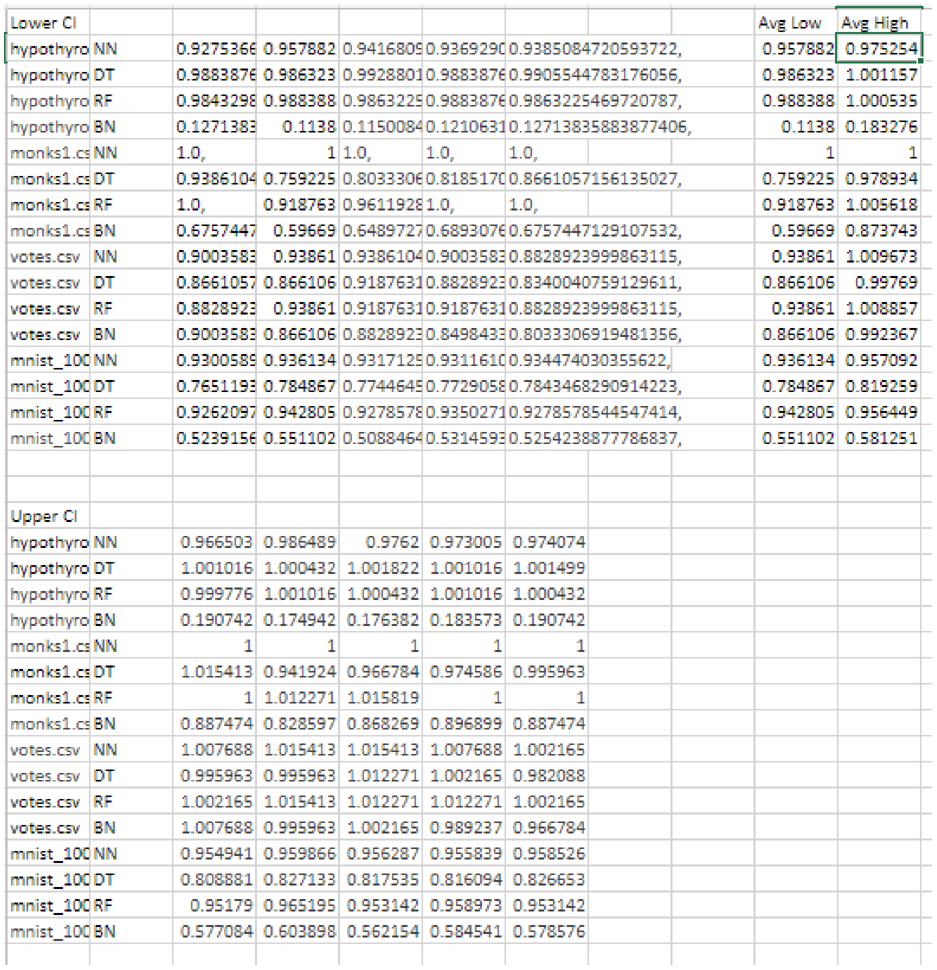
	DT	NN	RF	BN
negative	0.997164	0.998857	0.995736	0.134316
compensa	0.990909	0.58619	0.990909	0.624459
primary_h	0.963116	0.46179	0.959114	0.480603
secondary	0	0	0	0
monks1.csv				
yes	0.931863	1	0.995349	0.816611
no	0.886859	1	0.988462	0.718038
votes.csv				
republican	0.958528	0.96986	0.958746	0.939149
democrat	0.901155	0.947452	0.95889	0.9069
mnist_1000.csv				
zero	0.813482	0.944217	0.936451	0.642443
one	0.775954	0.937755	0.933824	0.64154
two	0.854558	0.958121	0.96219	0.500482
three	0.766056	0.936709	0.952977	0.349739
four	0.774512	0.950542	0.938926	0.508167
five	0.777676	0.940347	0.941915	0.611691
six	0.775398	0.929873	0.929863	0.606602
seven	0.859729	0.962286	0.964397	0.817985
eight	0.766159	0.930854	0.934871	0.409752
nine	0.806297	0.957955	0.945245	0.448301

Table 2: Average Accuracies over five runs



hypothyro	NN	0.94702	0.972185	0.95894	0.954967	0.956291		0.957881			hypothyro	monks1	votes	mnist_1000
hypothyro	DT	0.994702	0.993377	0.997351	0.994702	0.996026		0.995232		NN	0.957881	1	0.96092	0.9449
hypothyro	RF	0.992053	0.994702	0.993377	0.994702	0.993377		0.993642		DT	0.995232	0.908046	0.935632	0.7978
hypothyro	BN	0.15894	0.144371	0.145695	0.152318	0.15894		0.152053		RF	0.993642	0.990805	0.958621	0.9442
monks1.cs	NN	1	1	1	1	1		1		NB	0.152053	0.765517	0.926437	0.5547
monks1.cs	DT	0.977011	0.850575	0.885057	0.896552	0.931034		0.908046						
monks1.cs	RF	1	0.965517	0.988506	1	1		0.990805						
monks1.cs	BN	0.781609	0.712644	0.758621	0.793103	0.781609		0.765517						
votes.csv	NN	0.954023	0.977011	0.977011	0.954023	0.942529		0.96092						
votes.csv	DT	0.931034	0.931034	0.965517	0.942529	0.908046		0.935632						
votes.csv	RF	0.942529	0.977011	0.965517	0.965517	0.942529		0.958621						
votes.csv	BN	0.954023	0.931034	0.942529	0.91954	0.885057		0.926437						
mnist_100	NN	0.9425	0.948	0.944	0.9435	0.9465		0.9449						
mnist_100	DT	0.787	0.806	0.796	0.7945	0.8055		0.7978						
mnist_100	RF	0.939	0.954	0.9405	0.947	0.9405		0.9442						
mnist_100	BN	0.5505	0.5775	0.5355	0.558	0.552		0.5547						

Table 3: Average CI over five runs



Lower CI							Avg Low	Avg High
hypothyro	NN	0.9275366	0.957882	0.9416809	0.9369290	0.9385084720593722,	0.957882	0.975254
hypothyro	DT	0.9883876	0.986323	0.9928801	0.9883876	0.9905544783176056,	0.986323	1.001157
hypothyro	RF	0.9843298	0.988388	0.9863225	0.9883876	0.9863225469720787,	0.988388	1.000535
hypothyro	BN	0.1271383	0.1138	0.1150084	0.1210631	0.12713835883877406,	0.1138	0.183276
monks1.cs	NN	1.0,	1	1.0,	1.0,	1.0,	1	1
monks1.cs	DT	0.9386104	0.759225	0.8033306	0.8185170	0.8661057156135027,	0.759225	0.978934
monks1.cs	RF	1.0,	0.918763	0.9611928	1.0,	1.0,	0.918763	1.005618
monks1.cs	BN	0.6757447	0.59669	0.6489727	0.6893076	0.6757447129107532,	0.59669	0.873743
votes.csv	NN	0.9003583	0.93861	0.9386104	0.9003583	0.8828923999863115,	0.93861	1.009673
votes.csv	DT	0.8661057	0.866106	0.9187631	0.8828923	0.8340040759129611,	0.866106	0.99769
votes.csv	RF	0.8828923	0.93861	0.9187631	0.9187631	0.8828923999863115,	0.93861	1.008857
votes.csv	BN	0.9003583	0.866106	0.8828923	0.8498433	0.8033306919481356,	0.866106	0.992367
mnist_100	NN	0.9300589	0.936134	0.9317125	0.9311610	0.934474030355622,	0.936134	0.957092
mnist_100	DT	0.7651193	0.784867	0.7744645	0.7729058	0.7843468290914223,	0.784867	0.819259
mnist_100	RF	0.9262097	0.942805	0.9278578	0.9350271	0.9278578544547414,	0.942805	0.956449
mnist_100	BN	0.5239156	0.551102	0.5088464	0.5314593	0.5254238877786837,	0.551102	0.581251
Upper CI								
hypothyro	NN	0.966503	0.986489	0.9762	0.973005	0.974074		
hypothyro	DT	1.001016	1.000432	1.001822	1.001016	1.001499		
hypothyro	RF	0.999776	1.001016	1.000432	1.001016	1.000432		
hypothyro	BN	0.190742	0.174942	0.176382	0.183573	0.190742		
monks1.cs	NN	1	1	1	1	1		
monks1.cs	DT	1.015413	0.941924	0.966784	0.974586	0.995963		
monks1.cs	RF	1	1.012271	1.015819	1	1		
monks1.cs	BN	0.887474	0.828597	0.868269	0.896899	0.887474		
votes.csv	NN	1.007688	1.015413	1.015413	1.007688	1.002165		
votes.csv	DT	0.995963	0.995963	1.012271	1.002165	0.982088		
votes.csv	RF	1.002165	1.015413	1.012271	1.012271	1.002165		
votes.csv	BN	1.007688	0.995963	1.002165	0.989237	0.966784		
mnist_100	NN	0.954941	0.959866	0.956287	0.955839	0.958526		
mnist_100	DT	0.808881	0.827133	0.817535	0.816094	0.826653		
mnist_100	RF	0.95179	0.965195	0.953142	0.958973	0.953142		
mnist_100	BN	0.577084	0.603898	0.562154	0.584541	0.578576		