Analyzing Supervised Learning Algorithms

NBA Draft

Deciding who to pick during the NBA draft (like any sports draft) is big money and can mean the success or failure of a franchise. Coaches need to be able to evaluate the value of draftees. It is also a domain with quite a few statistics, so it should be applicable to machine learning algorithms.

Data on past drafts and individual performances are easily accessible through www.nba.com (see below for specific reference). While the statistics have legal restrictions (so data set is not included with this assignment) they are available for non-comericial purposes. I retrieved stats on draftees for the years of 2005-2008 and the performance of individuals 4 years later 2009-2012. I scrubbed the data using various scripts and hand editing missing, inconsistent formatting, and mismatched names (ex: “Reginald” in draft but “Reggie” in player stats). I then wrote a script, “processPlayers.py”, to load attributes, create classification and output Weka-ready .arff files. Draft years 2005-2007 are used for training and draft year 2008 for test. This should demonstrate how well predictions from past data fit the next year's data.

Here are some potential classifications a coach might want to query:

1. Frequent Player (will the player play frequently, infrequently, or be inactive in 4 years)
2. Will the player be in the top 20 NBA players in any category?
3. Will the player be top in a specified category (rebounding, points scored, assists steals)
4. Any of the above limited to position type (guard, forward, center)

For the purposes of this assignment, I used the “Frequent Player” classification: “frequent” (played in more than 60 of the 82 regular season games), “infrequent” (60 games or less), “inactive” (not on roster).

Initial Analysis

The data is classified as follows:

Data Set	Total Instances	Frequent	Infrequent	Inactive
Training	322 (100%)	33 (10.2%)	39 (12.1%)	250 (77.6%)
Test	102 (100%)	19 (18.6%)	7 (6.9%)	76 (74.5%)

Already we see problems where the test data “frequent” (18.6%) and “infrequent” (6.9%) classifications are inconsistent with the training data (10.2% and 12.1%).

Just to get a feel for the data, I processed the data in Weka with the following algorithms, using all available attributes, classifying by “Frequent Player”:

Algorithm	% Correct in Training Set	% Correct in Test Set
Decision Tree Unpruned (J48)	94.7%	71.5%
DecisionTree Reduced Error (J48)	77.3%	74.5%
Artificial Neural Network – ANN (default)	81.1%	75.5%
kNN (IBk default)	100%	64%
SVM (SMO default)	77.4%	74.5%

These results are pretty bad across the board – which I expected without any prior analysis on the data.

But even more revealing is the “Confusion Matrix” which shows combinations of correct and incorrect classifications:

Artificial Neural Network

		Classification		
		frequent	infrequent	inacative
Actual	frequent	1	3	15
	infrequent	0	5	2
	inactive	4	1	71

SVM

		Classification		
		frequent	infrequent	inacative
Actual	frequent	0	0	19
	infrequent	0	0	7
	inactive	0	0	76

Decision Tree Reduced Error

		Classification		
		frequent	infrequent	inacative
Actual	frequent	0	0	19
	infrequent	0	0	7
	inactive	0	0	76

KNN

		Classification		
		frequent	infrequent	inacative
Actual	frequent	3	3	13
	infrequent	1	4	2
	inactive	10	7	59

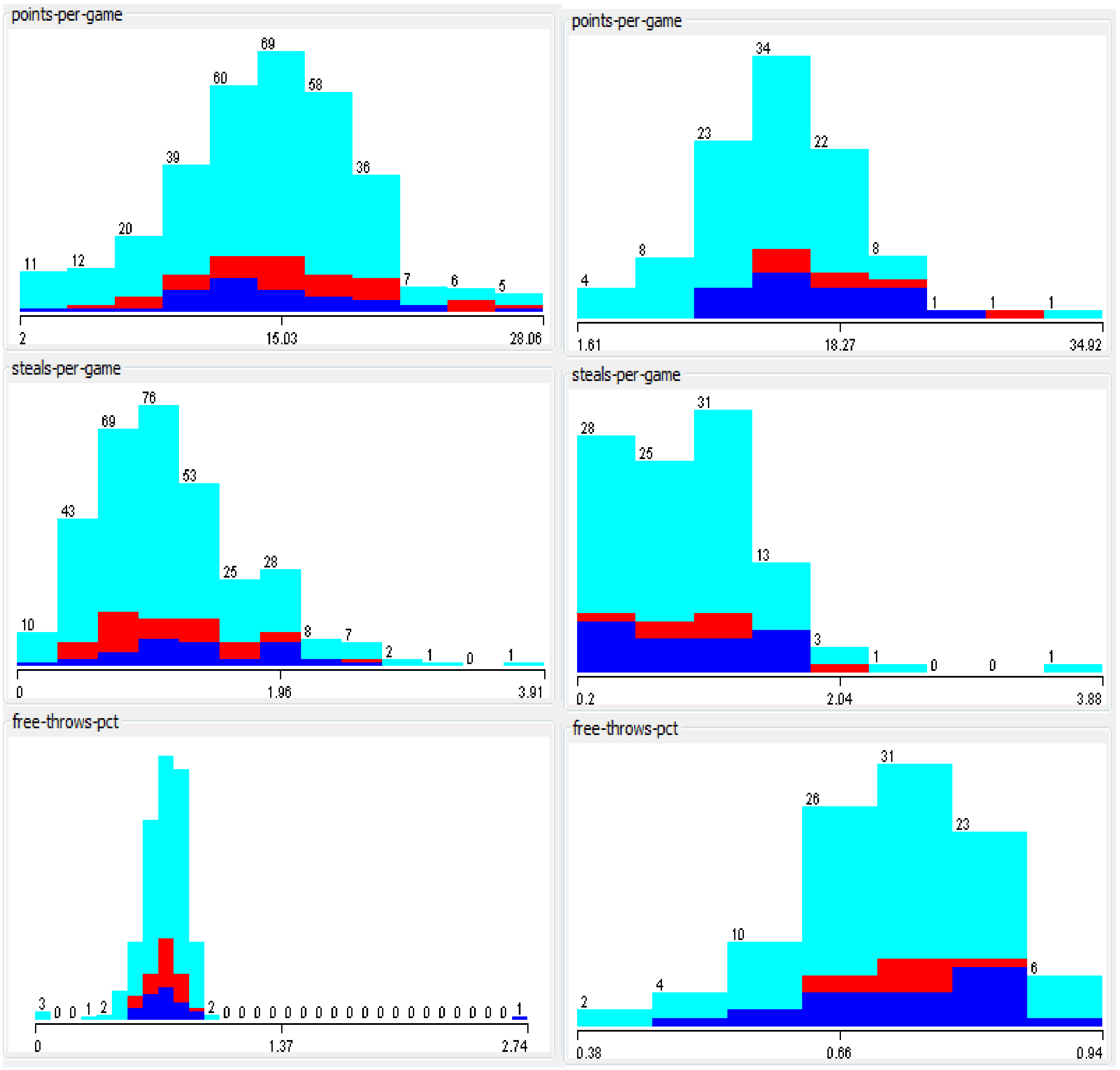
Both SVM and “Decision Tree Reduced Error” simply classify every instance as being “inactive”. The “Decision Tree Reduced Error” prunes the tree to a single node! Both these decision trees get about the same “% Correct” as “Neural Network”. And although kNN and “Nerual Network” classified several as “frequent”, more were wrongly classified as “inactive”. And both algoritms classified more as “inactive” that turned out to be “frequent”. Maybe this just shows that it's really hard to make it in the NBA!

Going back to the data, it becomes clear that we need to do a better job of preparing our data before even investigating various algorithm options. There are 22 attributes for draftees. Due to parse consistency issues, 4 were already determined to be invalid (adding percentages). Ironically, the “Decision Tree Unpruned” made it's second split on “3ptrs-pct-invalid”. But this might be promising, because rather than using “attempts” and “made” we should probably use percentages. I repurposed my script to calculate “per-game” and “per-attempt” percentages as follows:

Original	New
points	points-per-game
rebounds	rebounds-per-game
assists	assists-per-game
blocks	blocks-per-game
steals	steals-per-game
field-goals-attempted	field-goal-pct
field-goals-made	
3ptrs-attempted	3ptr-pct
3ptrs-made	
free-throws-attempted	free-throw-attempts-per-game
free-throws-made	free-throws-pct
league	league
height	height
weight	weight
games-started	games-started-per-game
games-played	
age	age

After changing attributes and running the default “DecisionTree”, ANN, kNN, and SVM algorithms on the new attributes the “% Correct” is the same: in many cases the algorithms classify all instances as inactive.

The problem seems to be that *the attributes are distributed similarly for each classification*. This means that perhaps there is no way to accurately distinguish between classes. Below are the distributions for 3 attributes (which are indicative of other attributes):



Training Data Test Data

Points-per game – blue = frequent, red = infrequent, cyan = inactive

The “free-throw-pct” has one outlier somewhere at 2.74 – which is invalid and must be a sign of incorrect source data or parsing. Indeed, if you look at “Ramon Sessions” stats at http://www.nba.com/draft2007/profiles/RamonSessions.html, he has FTM 118 out of FTA 43 – a super-human accomplishment! Taking him out of the line-up, however, makes no difference as most algorithms still predict all players “inactive”.

Next Steps

After looking at the similar distributions for each classification, it seems unlikely we will find any algorithm that will produce better predictions. But perhaps there are subtle combinations of attributes that will improve predictions.

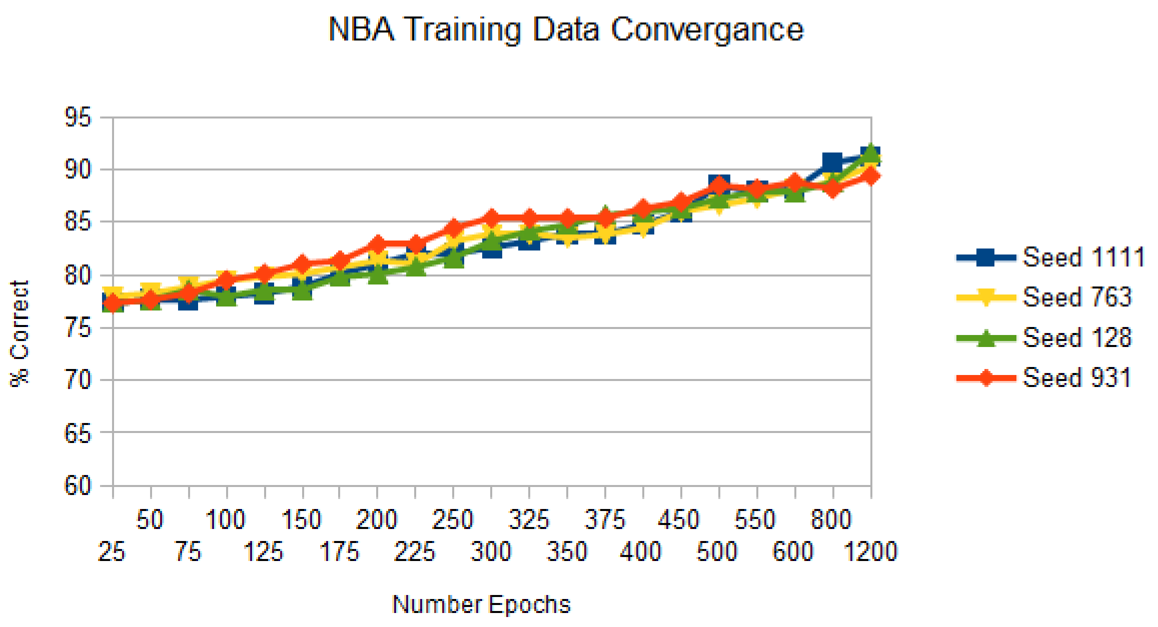
Decision Tree

The Weka "J48 Decision Tree" algorithm is a variant of C4.5, which is an extension of ID3 that works with numeric attributes by choosing the attribute and threshold with the best gain. As used above, J48 allows using reduced-error pruning (removing leaves while prediction accuracy remains the same). It has two additional options: minimum number of instances in leaf nodes (default 2) and laplace smoothing (which pushes probabilities slightly toward uniform). Testing both of these options results in the exact same decision tree as the default unpruned tree.

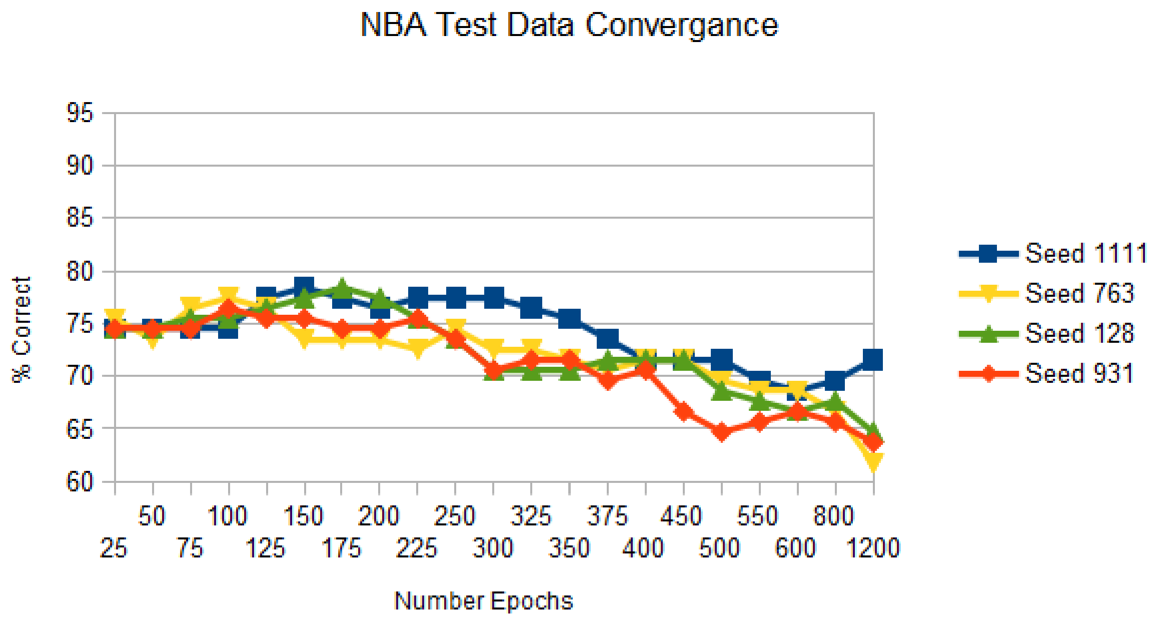
Neural Network

The MultilayerPerceptron is Weka's Neural Network algorithm using a sigmoid activation function. Setting the number of layers and nodes per layer causes the greatest variation in the network itself. The default option automatically chooses one hidden layer with (num-attributes + num-classes)/2 nodes. Changing the model to 2 hidden layers and trying various combinations for number of nodes – layer1 in {15, 20, 25} and layer2 in {8, 10, 12} – returns exact same results as the defualt.

MultilayerPerceptron has various options to guide convergence (learning rate, momentum, number of epochs, % valid training termination, consecutive error termination). These help converge quicker, avoid local minimums and decide when the algorithm has learned enough. To see convergence, I tried varying number of epochs: 25, 50, 75 … 400 and then tried 450, 500, 550, 600, 800 and 1200. I also tried various random seeds which showed some differences, but followed the same convergence path. Below are the result of training and test set convergence.



The training data clearly continues to converge, however the all measures of test data "% Correct" are lower than at 25 epochs. In fact, it looks like somewhere between 100 and 150 epochs has the best average "% Correct". The actual best average of 4 seeds (76.4% occurs at 125 epochs) – beyond which overtraining occurs. This seems to show that ANN is a very poor predictor for the NBA data.



Boosting

I used Weka's AdaBoostM1 to boost the J48 algorithm. AdaBoost iterates over “time” finding weak-learner hypotheses and then takes a weighted average. I predicted before running that boosting would do no better than the other algorithms. I changed the main parameter (number of iterations) exponentially (2, 4, 8, …, 128). As iteration increases the “% Correct” improves, but it never does better than 74.5% – a poor precision.

kNN

The IBk kNN algorithm has fairly simple options: number of neighbors, weight by distance, weight by inverse distance, number of neighbors to use, distance function, and “hold-out-one”. Useful distance functions are EuclideanDistance, ManhattanDistance (sum of distance in each dimension), and ChebyshevDistance (greatest distance from all dimensions). Running all combinations with k from 2 to 20 (increment by 1) reveals test “% Correct” starting around 50% and improving until k is somewhere between 6 and 9 depending on which options. They all max out at 74.5% correct for the test set and all algorithms classify all instances as “inactive”. Similar results occur when the number of attributes is increased. This too seems like a dead end.

SVM

I used Weka's SMO algorithm as a “Support Vector Machine”. I varied the polynomial kernel with exponent from 1.0 to 3.0 (increment by 0.5) with 8 attributes to see if there was any advantage to the higher order of polynomial. And without surprise the results were exactly as ANN, Boosting and kNN. All players are predicted “inactive” resulting in the “best” “% Correct”: 74.5.

Note: I omitted graphs for the Boosting, kNN and SVM algorithms (which are uniform – and, to be frank, boring) in favor of more detailed graphs on the source data.

Isolet Data Set

Language is one of the great promises of machine learning. It is amazing how accurately Google© voice recognition or Siri© transcribes spoken commands. This second data set I analyzed, "ISOLET (Isolated Letter Speech Recognition)", demonstrates the complexity in interpreting sounds. The data set was generated by 150 subjects speaking each letter in the English alphabet twice for a total of 7797 instances (3 instances were missing). There are 618 attributes (including the letter classifier) which represent various normalized (-1.0 to 1.0) audio features (including spectral coefficients; contour features, sonorant features, pre-sonorant features, and post-sonorant features). I could not find the original paper describing these features and the exact ordering of features where not known to the data set donor. The file, "isolet.info", included with the data distribution, describes papers written that use this set to analyze "Neural Network Backpropagation" (78 hidden units, 26 output units) and "C4.5 Decision Tree".

This data set presents some particular challenges. While the sample size is fairly large, the 618 attributes imply some danger of overfitting. The high number of attributes pose challenges iterating on algorithm options because of the longer running time. I also have only a rudimentary understanding of audio and the meaning of the feature set is not readily available. So it is difficult to limit features based on domain knowledge. But lets see what we can analyze from the data alone.

Decision Tree

Fortunately, training a decision tree (even with so many attributes) runs relatively quick. The J48 classifier took just 47 seconds running against training and test sets.

J48 Options	% Correct in Test Set	Tree Size	Num Unique Attr
Unpruned / Test Set	83.5%	657	211
Subtree Raising / Test Set	83.5%	631	204
Subtree Raising / Cross-validation	82.3%	631	204
Subtree Raising & Reduced Error / Test Set	82.9%	303	113
Reduced Error Pruning Test	82.9%	303	113

I'm surprised that the "% Correct" in the test set is so low (nearly 20% wrong). This is certainly nothing close to the accuracy that Google© voice recognition achieves.The test set accuracy for each tree is virtually the same, but, the size of the tree and number of unique attributes is quite diverse.

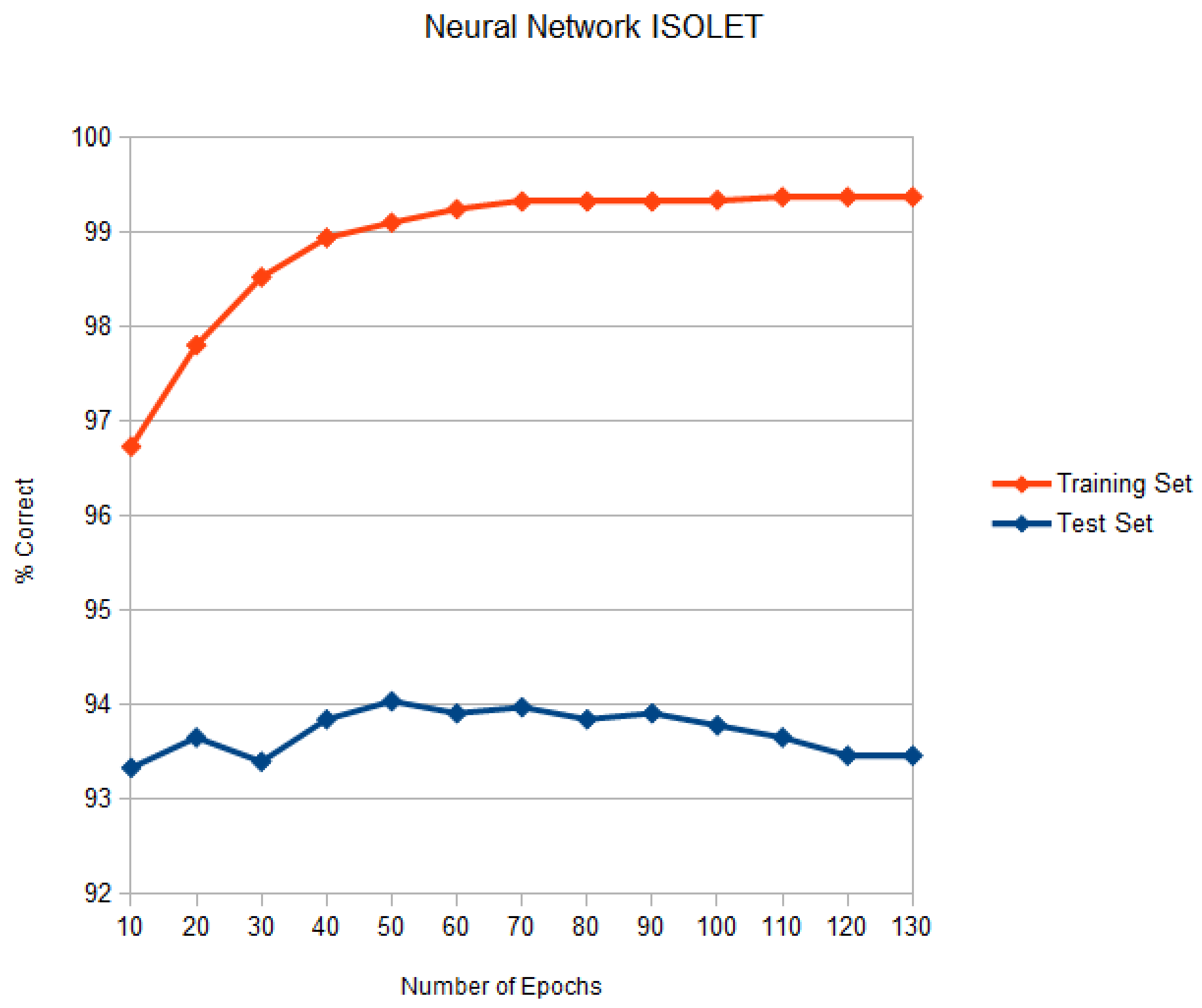
Artificial Neural Network

My first attempt at running the Weka MultilayerPerceptron used the default options and took over 2 hours to train and 102 seconds to apply the ANN to the test data. However, the improvement of "% Correct" to 95.5% is very promising. While this is not practical for prototype iteration, 2 hours is significantly less than the time it takes a human to learn how to recognize the alphabet! And it is beyond human limits to classify 1559 instances of the alphabet in less than 2 minutes (much less get 95% right)! Although slow in computer time, this might be practical for some real-time applications. But the processing power (either on mobile devices or servers) is more expensive than we would like. I am certain we can do better.

To improve performance of the ANN, I simplified the model version by limiting the number of attributes to the top 113 unique attributes (plus the classifier) from the Decision Tree: Reduced Error Pruning.

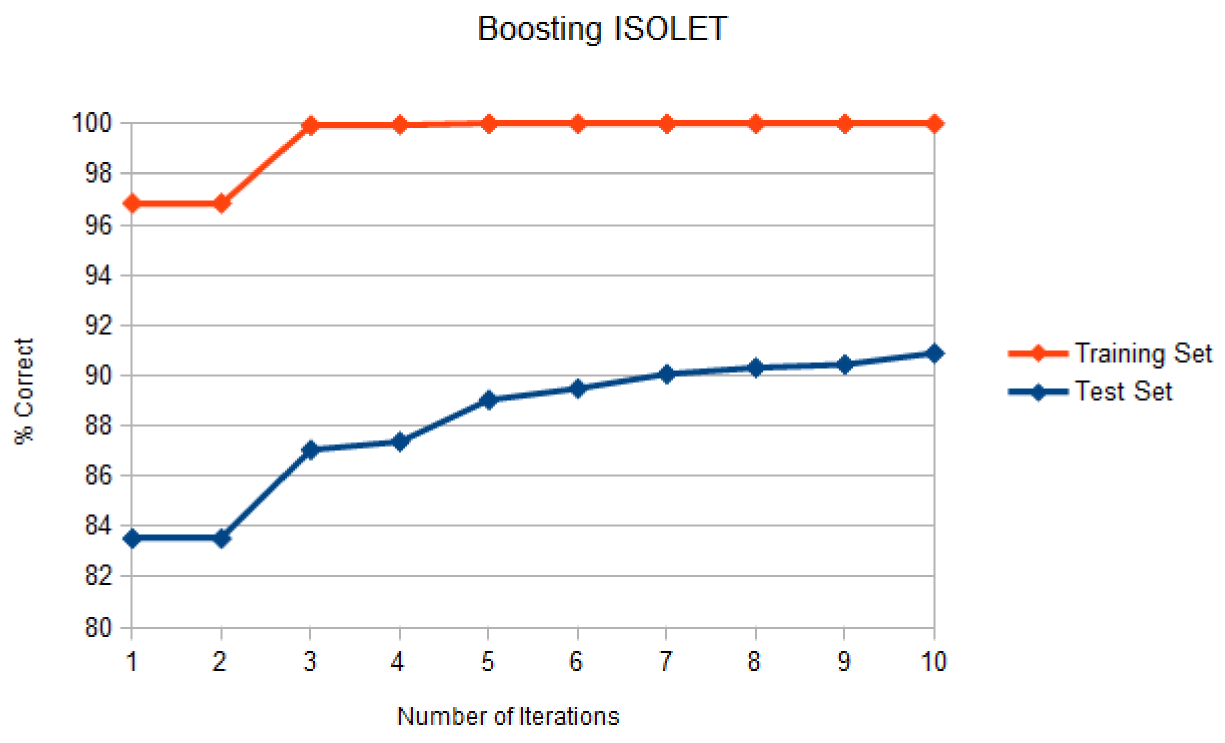
When using the top 113 attributes (with the default 50 epochs) I get a test "% Correct" of 94.1% in a training time of 85.5 seconds and only 1.45 seconds to classify. The graph below demonstrates the "%

Correct" continues to converge past 130 epochs (although mostly level), but the test correct peaks at 50 epochs. Which implies that beyond 50 epochs we are overfitting.



Boosting Decision Tree

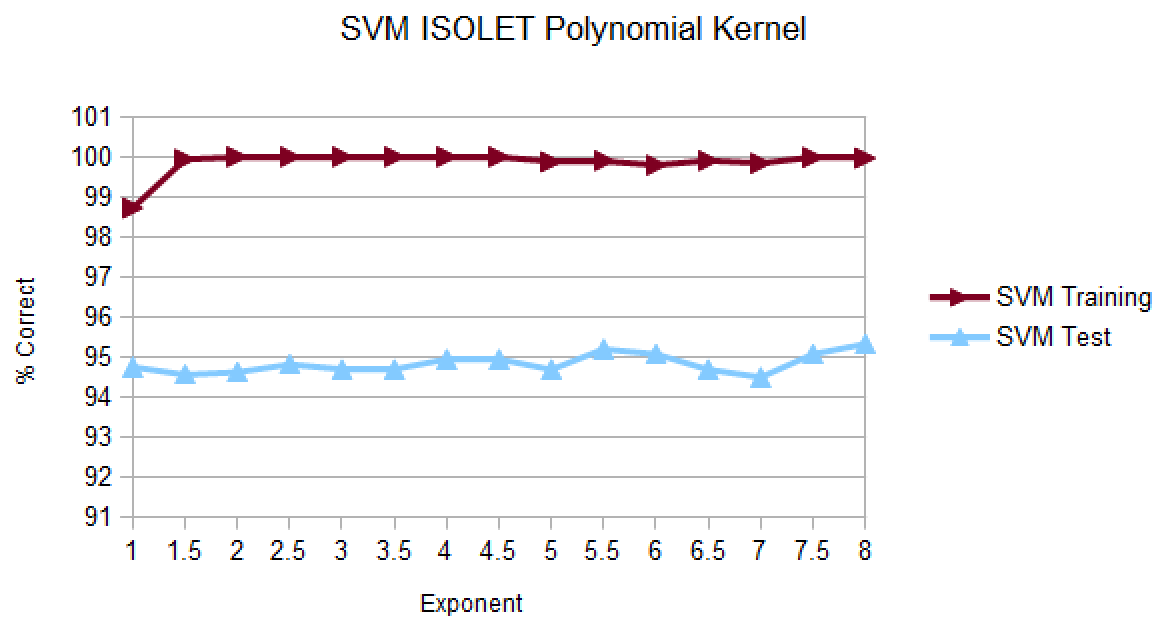
To improve the test set performance of decision trees, I ran the Weka AdaBoostM1 classifier on the J48 decision tree. The results were dramatically better (90.9% of test set correct) than the J48 classifier by itself (82.9%), but not as good as our neural network.



When I varied the number of iterations, the test rate consistently improves. Surprisingly, even after the training set has hit 100% correct, the test set “% Correct” continues to improve while increasing the number of iterations. Time did not permit finding the exact point where overfitting occurs.

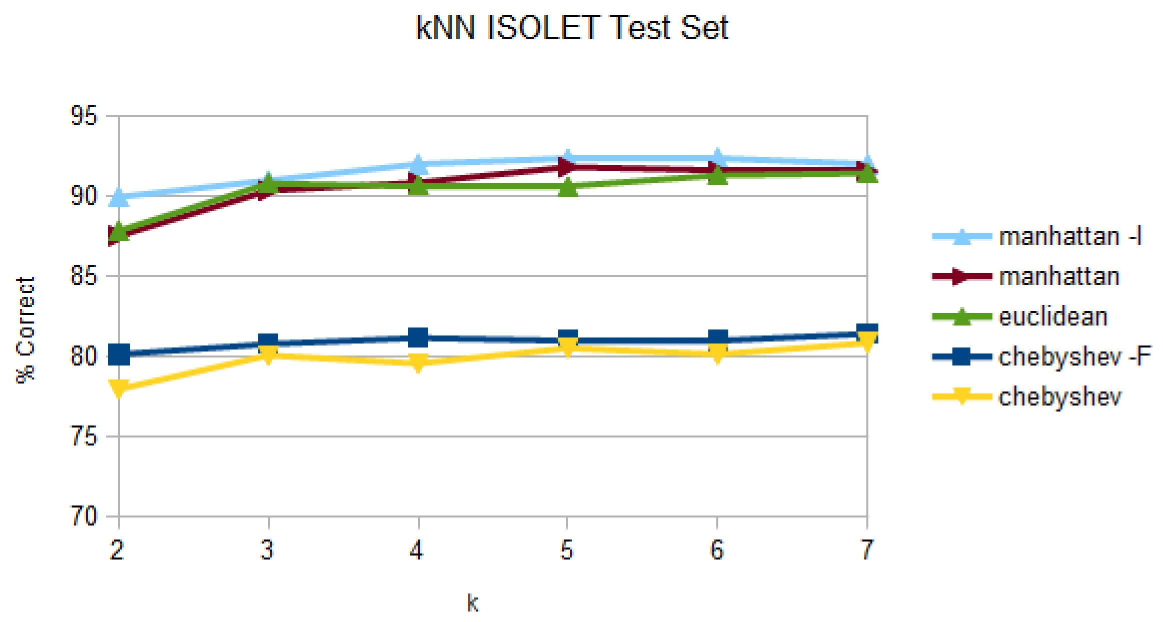
SVM

I ran Weka's SMO with the polynomial kernel, varying the exponent from 1.0 to 8.0 and found that the exponent has little effect on “% Correct”. The graph shows that linear polynomial has only slightly lower “% Correct” than any of the other values. However (see “Timing Performance” section further below), the time for applying the test set varies drastically.



kNN

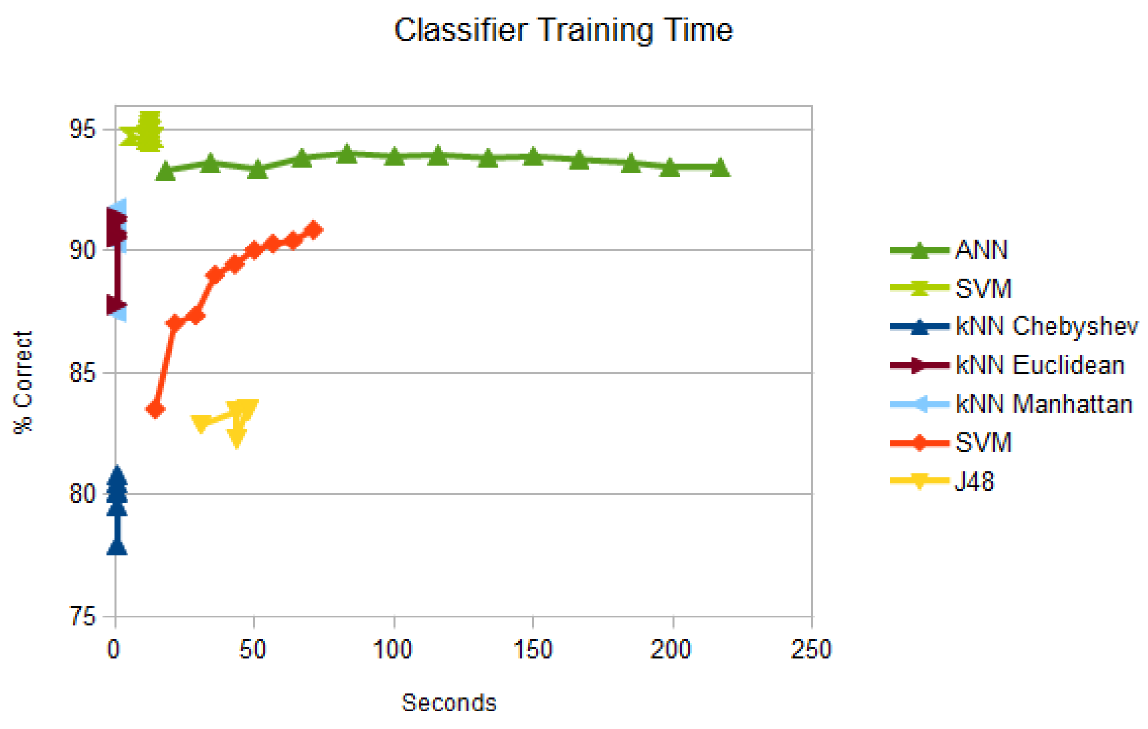
I ran the IBk kNN algorithm against the ISOLET data and experimented with the same options as I used in the NBA data. Many option variations were uneventful, but I included interesting options in the following graph of different distance functions for k from 2 to 7.

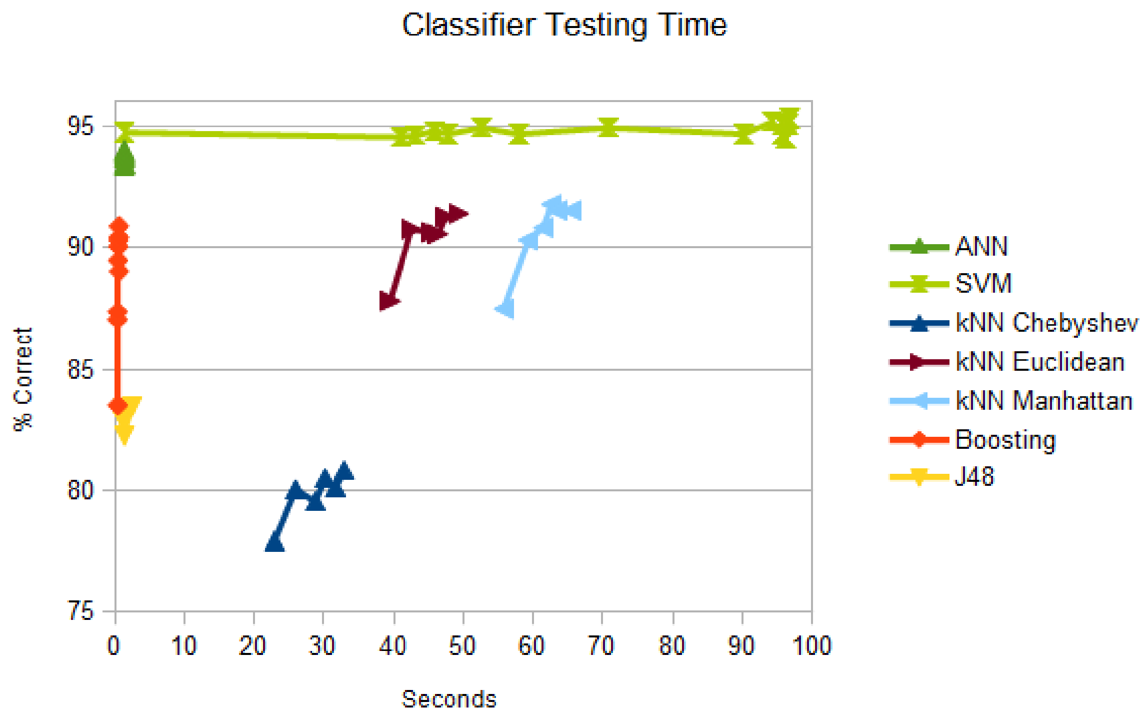


The "-F" option for "Chebyshev" weights neighbors by distance, and the "-I" option for "Manhattan" weights neighbors by inverse distance. I am surprised by how well manhattan distance predicts the letter. In all cases it seems that k is best at 5 or 6 (ranging from 80% to 92%).

Timing Performance

I compared Training and Testing wall clock time for the ISOLET data set on a Windows 7, Intel Q6600, 2.40GHz quad core processor (although only one CPU was used in each example). The larger size of the ISOLET data set make it better for benchmarking than the NBA data set. Below are the relative times compared to “% Correct”.





SVM is clearly the winner here with the best “% Correct” and very good training times. While the test time can get expensive very quickly, the first data point (upper left corner of “Testing Time”) is still better “% Correct” than any other algorithm. This corresponds to a kernel polynomial with exponent of one. Our linear example does the best all around!

Conclusions

Various data sources respond entirely differently to supervised learning algorithms:

- The NBA data is nearly useless in predicting future success. Perhaps this is the exact lesson for NBA coaches – the stats don't seem to make all that much difference in predicting how active a player will be in 4 years. So maybe they should rely on other criteria. Perhaps a different classifier would be useful. Or maybe there are some attributes that can be found with better prediction qualities.
- The ISOLET set, however, is well behaved even when running the supervised classifiers blindly. Because the data is so large, using the decision tree to limiting the number of attributes results in dramatically faster running without sacrificing much quality.

SVM (with polynomial kernel, exponent 1) combines speed and accuracy outperforming all other tested algorithms.

Future Investigations

The following methods would be helpful in understanding the data better.

NBA Data:

- Larger data sets (more years)
- Predict who will be in the draft from the larger pool of athletes
- Investigate other attributes (rank instead of %)

ISOLET Data:

- Try various numbers of hidden layers and number of nodes for “Neural Networks”
- Understand better what the attributes represent
- Find which attributes contribute most to the 1% difference between using all attributes (95.5%) and only the top 113 (94.1%) for “Neural Networks”

References

ISOLET Data Set. Retrieved *Jan 30, 2014,* https://archive.ics.uci.edu/ml/datasets/ISOLET.*Creators:* Ron Cole and Mark Fanty*, Donor:* Tom Dietterich

NBA Draft 2005-08, performances 2009-12. Retriveved *Jan 24, 2014,* http://www.nba.com/draft$i/profiles/byName.html, http://www.nba.com/draft$i/profiles/$j http://stats.nba.com/leaguePlayerGeneral.html?ls=iref%3Anba%3Agnav\&pageNo=$k\&rowsPerPage=100\&Season=20$l
$i = year (ex: 2005), $j = player (from byName.html), $k = page (1-5), $l = year (ex; 12-13)

Weka 3.6. Retrieved, *Jan 30, 2014,* http://softlayer-dal.dl.sourceforge.net/project/weka/weka-3-6-windows/3.6.10/weka-3-6-10.exe

Yoav Freund, Robert E. Schapire: Experiments with a new boosting algorithm. In: Thirteenth International Conference on Machine Learning, San Francisco, 148-156, 1996.

C4.5 algorithm, Retrieved *Feb 13, 2014*, http://en.wikipedia.org/wiki/C4.5_algorithm

Activation function in MLP, Retrieved *Feb 13, 2014,* http://weka.8497.n7.nabble.com/Activation-function-used-in-MLP-td14678.html