Comparison and Analysis of Supervised Learning Algorithms

Author: Dilara Soylu

Abstract - In this paper, performances of the two UCI datasets, pendigits and tic-tac-toe, are compared and analyzed by applying a subset of supervised learning algorithms: decision trees, neural networks, boosting, support vector machines, and k-nearest neighbors. Percent of accuracies are used as performance metrics. Performance of each algorithm on each dataset is analyzed on its own in dedicated sections. In the final section, observations made in individual sections are combined with analyses examining all the algorithms for both of the datasets.

I. INTRODUCTION

Both of the datasets used in this study are publicly available on UCI machine learning repository. First dataset, tic-tac-toe is a binary classification problem with three possible values for each of its 9 attributes. I picked this dataset mainly because I was interested in AI agents defeating humans in games. It encodes “the complete set of possible board configurations at the end of tic-tac-toe games, where “x” is assumed to have played first” as described by its creators (Alpaydin & Alimoglu, 1998). Each of its attributes represent a box in a classic 3 by 3 tic-tac-toe game. Boxes are either blank or marked by “x” or “o”, so each attribute can take a value from the set {x,o,b}. The classification problem is to find the instances where “x” is the winner (three x’s in a row, column or diagonal). There are total of 958 instances in the dataset. For the purposes of this study, the dataset was randomized and split into an unbalanced (221 negatives vs. 417 positives) training and testing sets with 638 (2/3 of the dataset) and 320 instances, respectively. The proportion of the classes are roughly the same in both training and testing tests.

Second dataset, pendigits, has 10 classes and 16 attributes, each of which can take a natural number between 0 and 100. The digits were encoded by sampling (x,y) coordinates of the points belonging to the digits in a normalized image. The re-sampling algorithm used by the makers of the dataset uses linear interpolation between pairs of points. They report that spatial re-sampling of 8 points give the best trade-off between complexity and accuracy (Aha, 1991). Dataset has 16 attributes since each point has a horizontal and a vertical coordinate. There are total of 10992 instances in the dataset. For the purposes of this study, the dataset was randomized and split into training and testing sets with 7328 (roughly 2/3 of the dataset) and 3664 instances, respectively. Unlike tic-tac-toe dataset, number of instances in each class is balanced.

The two datasets are particularly interesting for couple of reasons. Tic-tac-toe dataset encodes a combinatorial problem. Therefore there are a finite number of possible inputs (all the end game configurations), whereas there is no limit for the number of data points in pendigits dataset (one could add more.) Tic-tac-toe dataset is a binary classification problem with 958 instances, but it is hard to work with since the dataset is unbalanced (more negatives than positives.) Pendigits dataset has much more data points, 10992, but it has 10 classes.

Rest of this paper explores how does each supervised algorithm performs on both of the datasets. The structure is as follows: a section is dedicated to each algorithm for accuracy and

model complexity analyses on the datasets. Later on, insights from all the sections are combined in the final section where algorithms are compared to each other. All the experiments were performed using Weka library. Explanations in this paper focus on categorization tasks. When comparing different figures to each other, note that their scaling might differ.

II. SUPERVISED LEARNING ALGORITHMS

Supervised learning algorithms are machine learning algorithms arriving at a function from labeled data. In this paper, five supervised algorithms are analyzed on the datasets: decision trees, neural networks, boosting, support vector machines and k-nearest neighbors.

A. *DECISION TREES*

Decision trees are supervised learning models mapping observations to outcomes in a tree-like structure. The branches represent the values for variables, decisions that could be made in each transition. The internal nodes in a decision tree represent features of the input vector. The leaves represent the outcome of the tree for a specific variable. There are many types of decision trees, but for the purposes of this work, Weka's J48 classifier implementing C4.5 decision tree model was used.

There are couple of factors determining the accuracy of a decision tree: the algorithm used to build the tree (whether it uses information gain), pruning method used and parameters passed for pruning. C4.5 uses an improved version of ID3 algorithm to build a decision tree, using information gain to pick attributes, which makes it biased towards shorter trees. There are two types of pruning: on-line (pre) pruning and post pruning. On-line pruning operates while the decision tree is being built. In one kind of on-line pruning, when the algorithm adds a new child node, it checks how many examples from the training set the node represents. If the number of examples it represents is smaller than some number M, then the parent node and the child node are combined into one node. Post-pruning collapses the decision tree by deleting parent nodes satisfying certain characteristics after it is built by working from the bottom up. Confidence factor is used to determine a pessimistic upper bound in the error at a node. Hence, smaller confidence factors cause more aggressive pruning.

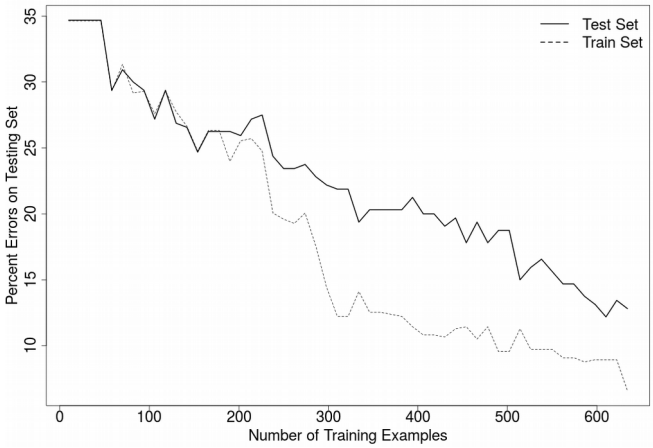

Figure 1: Percent error of decision tree models as a function of the tic-tac-toe training set size.

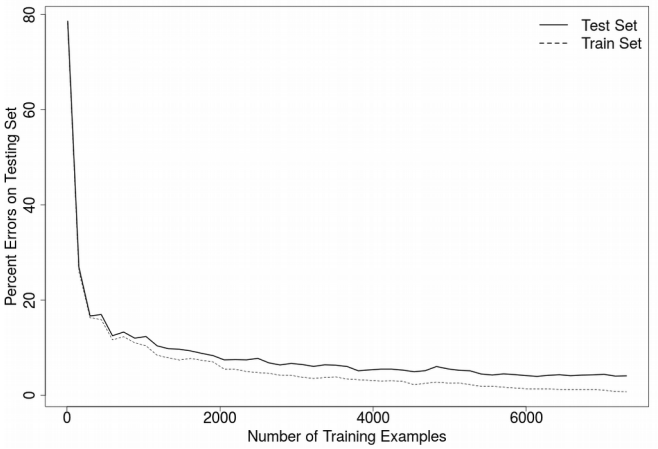

Figure 2: Percent error of decision tree models as a function of the pendigits training set size.

Figures 1 and 2 visualize how does the accuracy of the decision tree models change as the training set size varies. Learning curves show that smaller portion of the examples can decrease the error at a higher rate for pendigits dataset, whereas for tic-tac-toe dataset even if the whole set is used, the error is higher than that of pendigits dataset. It should be noted that in pendigits dataset there are 7328 data points in the training set whereas the training set for tic-tac-toe dataset contains 638 examples. It is clear that for pendigits dataset after 4000 examples, the accuracy of the testing set doesn't vary much, the line becomes flat. The model won't benefit from adding more data points at that point. However, for tic-tac-toe dataset the error keeps decreasing even if all the data points in the training set are used, and even though we cannot add any more data points apart from those in the testing set. This means that tic-tac-toe data points are more different in terms of inputs to outputs matching in comparison to pendigits data points.

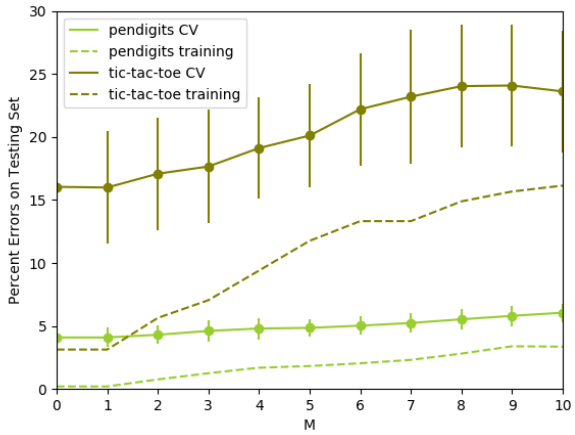

Figure 3 Percent error of decision tree models in respect to M.

Figure 3 illustrates how the percent error varies for both of the datasets as M changes. Increasing M forces the model to put more data points on the same node. Since both the training and validation errors increase as M increases, it can be inferred that increasing M restricts the capabilities of the models; hence, they start to underfit the data.

The increase in the error rate is more aggressive on tic-tac-toe dataset in comparison to that of pen digits dataset. Perhaps, tic-tac-toe dataset contains some hard examples with similar input vectors, but belonging to different classes; therefore the models trained on tic-tac-toe dataset should be capable of splitting to nodes containing only one data point to have high accuracy. For pendigits dataset, there is more overlap between different data points, since the data points labeled with the same digits tend to have similar values for their features. For example, for the digit '4', it is less likely to have a feature (point with x,y-coordinates) corresponding to the middle of the image. On the other hand, the points making up the tic-tac-toe dataset are different end game board configurations. The similarity is more abstract and combinatorial, the data points do not overlap as much in that sense.

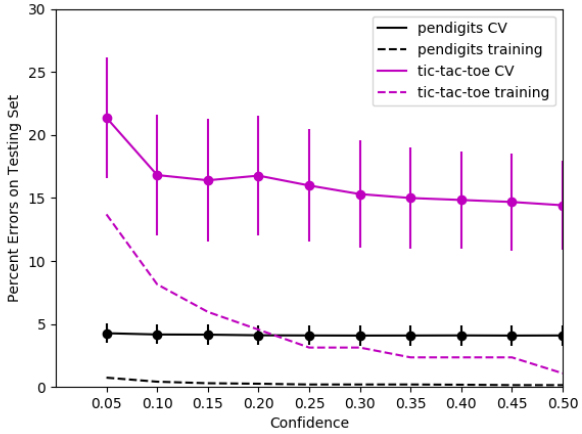

Figure 4: Percent error of decision tree models in respect to the confidence factor.

Figure 4 shows how does the error change with confidence factor. Comparing *Figure 3* to *Figure 4* reveals that for tic-tac-toe dataset post pruning is more effective than pre-pruning. This means that

even though the model put some data points in a node by their selves using information gain as a metric, the data points were not representative of the validation set. In this case, more aggressive post pruning decreased the error because model was over-fitting. For pendigits dataset, the models didn't get any better after post-pruning because the data points were easier to generalize; more similar to each other as mentioned previously. Decision trees are sensitive to small changes in the data. As mentioned previously, this explains why the percent errors of the models trained on tic-tac-toe dataset are higher.

B. *NEURAL NETWORKS*

In this section, multilayer perceptron classifier was used for experiments. Each perceptron in the hidden layers are connected to other perceptrons with certain weights. The input and output layers perceptrons are also connected to the input and output vectors, respectively. Through back propagation, or other chosen methods, neural networks adjust the weights of the links connecting the perceptrons by minimizing the error.

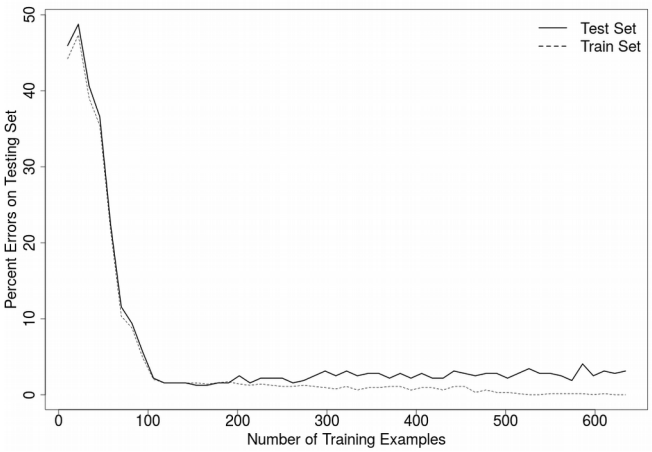

Figure 5: Percent error of NN models as a function of the tic-tac-toe training set size.

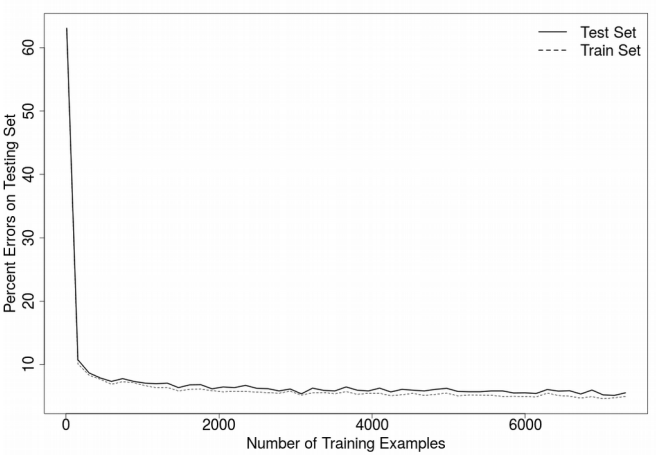

Figure 6: Percent error of NN models as a function of the pendigits training set size.

As it can be observed from *Figures 5* and *6* the percent error for both of the datasets rapidly decrease in the beginning. In comparison to decision trees, neural nets learn tic-tac-toe dataset faster in terms of the number of data points required. The same is true for the pendigits dataset. Neural nets can generalize to more arbitrary functions by seeing a portion of the dataset, which might be the reason why their learning rate is faster on tic-tac-toe dataset in comparison to the learning rate of decision trees.

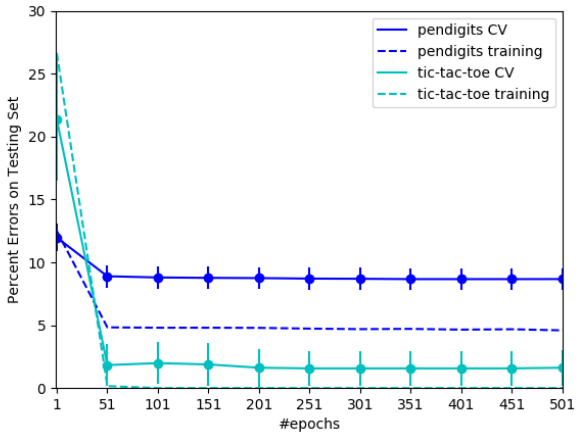

Figure 7: Percent error of NN for tic-tac-toe and pendigits datasets in respect to #epochs.

Figure 7 shows model complexity analyses of neural networks with no hidden layers in respect to

the number of epochs. For both of the datasets, around 50 epochs are needed for the models to learn the hypothesizes. After that the errors stay almost constant. The figure shows that the neural net has initially have a higher error for tic-tac-toe dataset but learns more in each iteration in the beginning in comparison to pendigits dataset. Seeing the full batch over and over again allows the neural net model to learn the overall structure, which might explain the lower error on tic-tac-toe dataset. In comparison to decision trees, neural net models in *Figure 7* have higher errors on pendigits dataset. It can be because pendigits dataset requires a smoother function than the sigmoid function, the underlying activation function used.

The percent error is slightly lower at 401 epochs in comparison to 51 epochs for tic-tac-toe dataset but it doesn't make sense to train the neural net with 401 epochs if it has a similar error rate with 51 epochs. The time it takes to train the model justifies choosing one over the other.

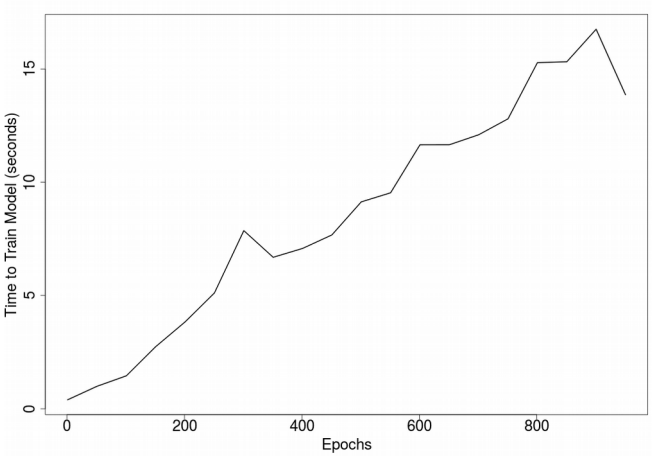

Figure 8: Time to train NNs for tic-tac-toe dataset as a function of #epochs.

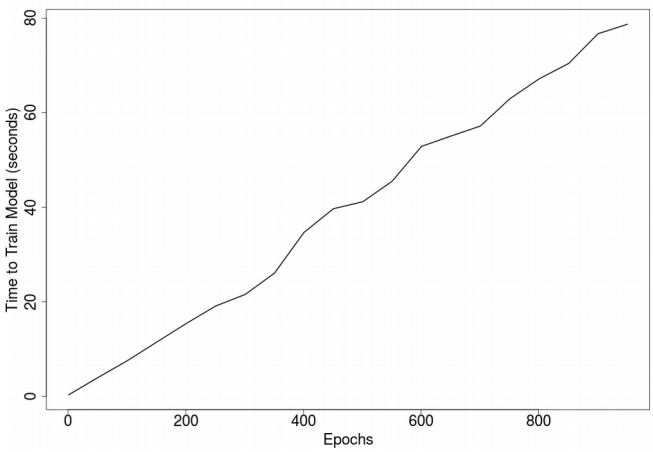

Figure 9: Time to train NNs for pendigits dataset as a function of #epochs.

Given that time to train the neural networks increases with the number of epochs, it is important to iterate just enough to achieve the desired accuracy. Since none of the datasets have too many data points, training times are quite low. It is possible to afford 80 seconds to train a neural network in most of the tasks. However, if time matters more, around 50 epochs would be enough for both of the datasets. It is important to note that this is a common trade-off in machine learning tasks. Increased complexity and accuracy comes with the cost of time, though this doesn't mean complexity always increases the accuracy as it can be observed from the graphs.

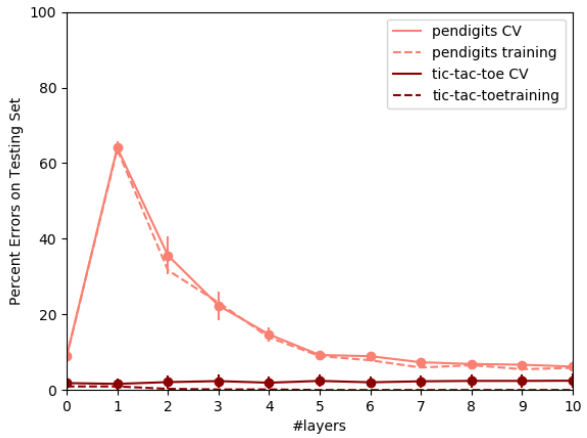

Figure 10: Percent errors of NNs for tic-tac-toe and pendigits datasets in respect to #layers.

Figure 10 shows how does changing the number of hidden layers affect the percent error for both of the datasets. The number of epochs were fixed to 51 for all the experiments. For tic-tac-toe dataset, the error doesn't change as much as it changes for pendigits dataset. Number of epochs have bigger impact on the accuracy of the neural net models for tic-tac-toe dataset then the number of layers. As opposed to tic-tac-toe dataset, for pendigits dataset number of layers has a bigger impact on the accuracy of the neural net in comparison to number of epochs. Increasing the number of layers increases the representational capacity of the network; however, this doesn't explain why the error jumps when the number of layers is increased to 1 from 0. The error rate at 1 hidden layer is around the same even with 10,000 epochs (not shown in the figure). Increasing the number of hidden layers makes it harder to train the network. Specifically adding the first layer doubles the number of weights. Although not a full explanation, combined with that fact that sigmoid is not an ideal activation function for pendigits dataset, this might partially explain the behavior.

C. *BOOSTING*

Boosting is a supervised learning model utilizing local trends to build a better global classifier. It has the "best-of-all" approach for picking the local classifiers. Local classifiers themselves might be weak, only slightly better than a random guess. Boosting aims to combine weak classifiers into a strong one. In this section, Ada-boosting model was used for the experiments.

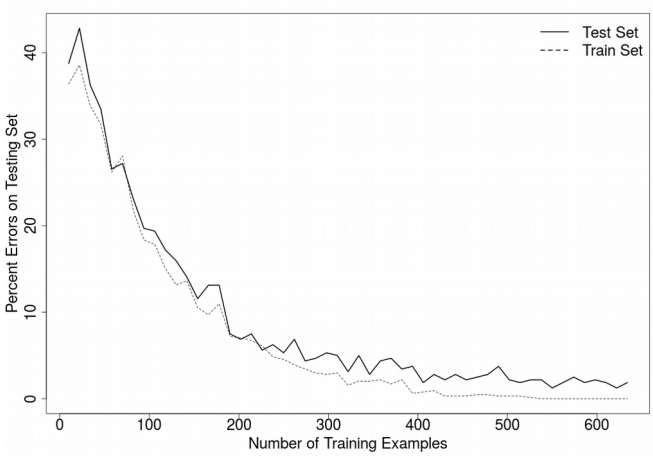

Figure 11: Percent error of Ada-boosting models as a function of the tic-tac-toe training set size.

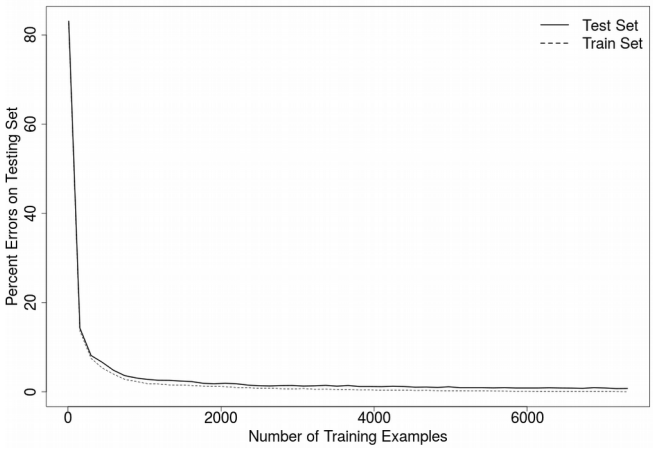

Figure 12: Percent error of Ada-boosting models as a function of the pendigits training set size.

Figures 11 and *12* show percent errors of Ada-boosting models as a function of the training set size. When compared to the same graph in decision

trees section, it can be seen that the performance of the pendigits dataset is quite similar. However, the learning curve for tic-tac-toe dataset is much more steeper, but still not as steep as the neural net learning curve, confirming the earlier observation about the importance of generalization for this dataset.

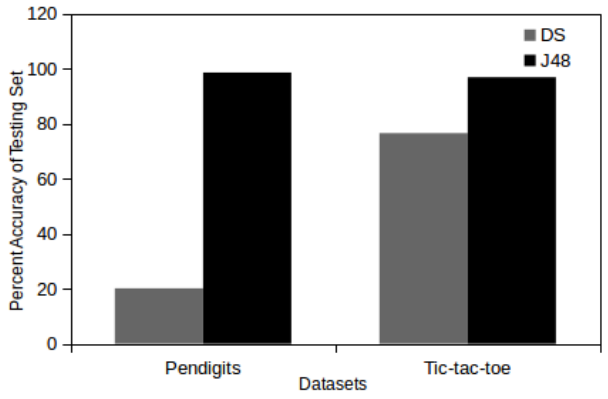

Figure 13: Percent accuracy of adaboosting for pendigits and tic-tac-toe datasets with different weak classifiers.

The choice of the weak classifier is crucial for the performance of the boosting model. Figure 13 shows how different choices of the weak classifiers affect the percent accuracy for both of the datasets. When decision stump (DS) is used as the weak classifier, accuracy of the Ada-boosting model is no better than the accuracy of the decision stump alone. 75% is better than random categorization for tic-tac-toe dataset. Even then, the result of Ada-boosting with decision stump is no different for tic-tac-toe. However, when J48 is used as the weak classifier, pendigits and tic-tac-toe datasets reach accuracies of 98.5% and 96.8%, respectively. For pendigits dataset this is slightly better than using J48 alone but for tic-tac-toe dataset the increase in accuracy is around 10%. This can be explained by the bias of boosting, meaning that tic-tac-toe dataset has local properties that are neglected by a global classifier but Ada-boosting can detect them.

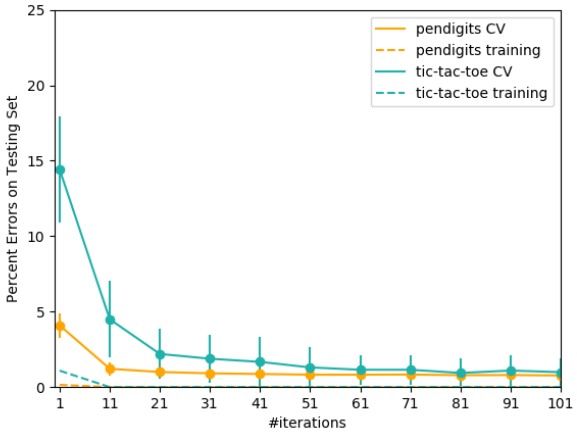

Figure 14: Percent accuracies of adaboosting for pendigits and tic-tac-toe datasets as functions of the number of iterations.

Figure 14 shows how does the percent error changes with the number of iterations for both of the datasets. Cross validated results are shown to make sure that the improvement is not due to over fitting. The weak learner used is a J48 decision tree. Both datasets benefit from increasing the number of iterations. This is interesting because the percent error of using J48 by itself is higher for both of the datasets. Boosting, in fact, can combine learners with poor accuracy and achieve a higher accuracy as a result.

D. SUPPORT VECTOR MACHINES

Support vector machines (SVMs) are learning models aiming to find the hyperplane that separates data points of different classes with a clear gap. When a point is queried for a classification task, SVM model determines its class based on which side of the gap the point falls. Even though SVM models are linear binary classifiers, there are ways to apply them to nonlinear multi-class problems, otherwise it wouldn't be possible to use SVM models with pendigits data. Kernel trick, method

makes it possible to operate in an implicit feature space with a little computational cost, makes it possible for our SVM model to quickly train on pendigits dataset. This method increases generalization error, but since the datasets used in this study have many data points, the method works well.

SVMs aim to find a hyperplane with the largest minimum margin while correctly classifying as many points as possible. There are two main parameters for determining the accuracy of SVMs: *C* and the kernel function. Weka's libsvm classifier was used to measure performance of different kernel functions on both of the datasets, which uses one against all approach for multi class problems.

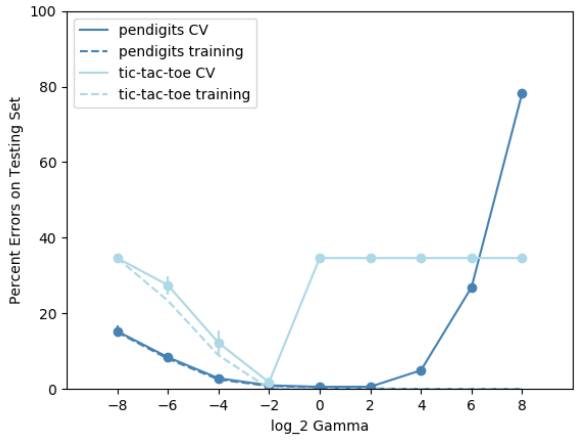

Figure 15: CV Percent Error of SVM with RBF Kernel as a function of Gamma.

Radial Basis Function (RBF) is one of the kernel functions that can be used with SVMs. RBF is a similarity function with a value decreasing as the distance between two input vectors increase. Gamma is used to determine how far the influence of a single data point would reach. It is related to the variance in normal distribution. If gamma is low, then the input vector has a far reach. If gamma is high, then the result is the opposite.

Referring to *Figure 15*, for pendigits dataset, when gamma exceeds 0.25, only a few data points decide which class the new input belongs to, hence leading to high variance and poor accuracy. This also explains the clear over fitting, when gamma exceeds 0.25, the training errors become 0. When it is less than 0.25, distant data points starts to interfere with the decisions of the closer points more than they need to; hence the poor accuracy. This explains why the accuracy of tic-tac-toe dataset is lower than the pendigits dataset in the graph: the input vectors that are close result in more different classes in comparison to pendigits dataset, in line with our earlier observations. This is more obvious when k is varied for kNNs.

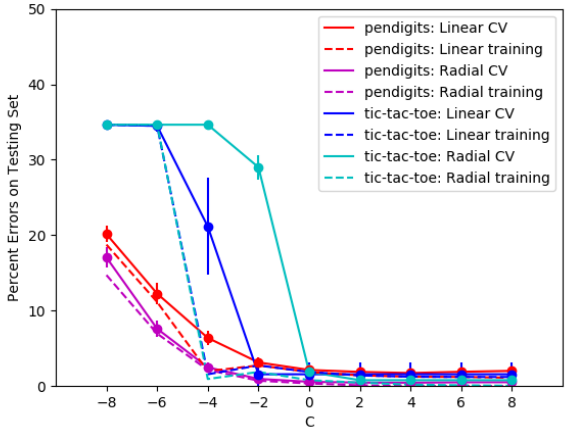

Illustration 16: Percent Error of SVM with RBF and Linear Kernels as a function of C (log_2).

Figure 16 shows how do the accuracies of SVMs with different kernel functions are affected by C. C determines how big the margin is going to be. For smaller Cs, the optimizer would look for big margins. For bigger Cs, the optimizer would search for smaller margins. Smaller margin increases variance but decreases the bias. This can be seen as overfitting in SVMs with radial kernels for tic-tac-toe dataset. The percent error goes down for both of the datasets using both of the kernels. Linear kernels perform well for high dimensional

problems, which suggests that both of the problems may have an high dimensional portion. Linear kernel is able to find a bigger margin for tic-tac-toe dataset in comparison to radial kernel at higher accuracies. For pendigits dataset, it is the reverse, radial kernel is able to find a bigger margin in comparison to linear kernel at higher accuracies. Non-linear kernels perform better when the number of dimensions are smaller. RBF kernel tends to select the smoother solutions, which might explain why it can find bigger margins for pendigits dataset in comparison to the linear kernel. For both of the datasets both kernel functions seem to perform equally well for high values of C, finding a small margin.

E. K-NEAREST NEIGHBORS

k-Nearest Neighbors (kNNs) is an instance based non-parametric lazy learning algorithm obtaining the classification by calculating the distance function of the query point with previously stored k closest instances. kNNs are said to be non parametric because it is not possible to compress the model into a few parameters like weights in neural networks or coefficients in simple linear regression tasks. Aside from the obvious factors, such as the size of the training set, there are two main factors affecting the accuracy of kNNs: choice of *k* and the distance function used.

Analyzing the extremes of *k* makes it easier to understand its meaning. In the case that *k* equals to 1, kNN will simply return the category of the closest instance. In most cases *k=1* causes model to over-fit the training data. Exceptions are the cases in which categories are near-perfectly separated by a large value. When k is too large, it is likely for neighbors to include lots of points from other classes, which may cause under-fitting.

Distance function is typically the majority vote of the k-nearest neighbors for classification tasks. Some distance functions also assign different weights to different attributes. Since all the attributes in both of the datasets have the same domain, there is no need to standardize the output of the distance function in our case. For the purposes of this analysis, different distance functions didn't affect the results of the models much, so no space is devoted to discuss them. They are still displayed in the graphs for future reference.

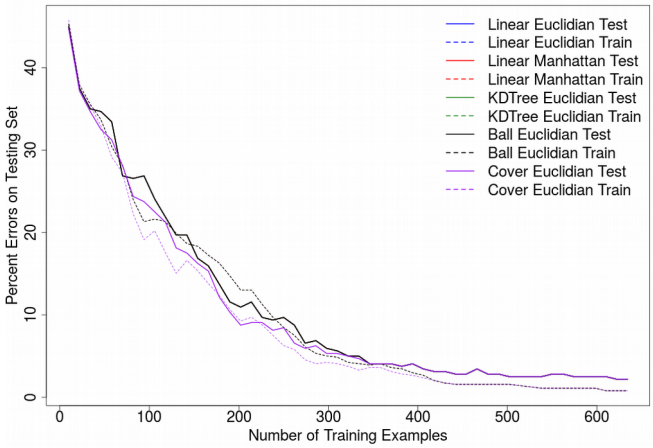

Figure 17 Performance of kNNs algorithm as function of the tic-tac-toe training set size.

Figures 17 and 18 show the performance of kNN as a function of the training set size for both of the datasets. Note that how fast the learning rate of kNN on pendigits dataset is in comparison to the tic-tac-toe dataset. The percent errors for both of the datasets stabilize after around 500 examples. However initial percent error of the pendigits dataset is nearly double the initial percent error of the tic-tac-toe dataset. One of the possible reason for this behavior is the distribution of the collected data. For pendigits dataset, it is likely for data points representing the same digits to have similar values for their attributes. Once a few examples for each digits are collected, classifying the query points becomes an easy task since nearest neighbors are more likely to belong to the correct

class. For the tic-tac-toe dataset, it is hard to find data points with slight differences (i.e. different values in one or two boxes). With just a few examples, kNN might be forced to consider the data-points that are even further from the query point.

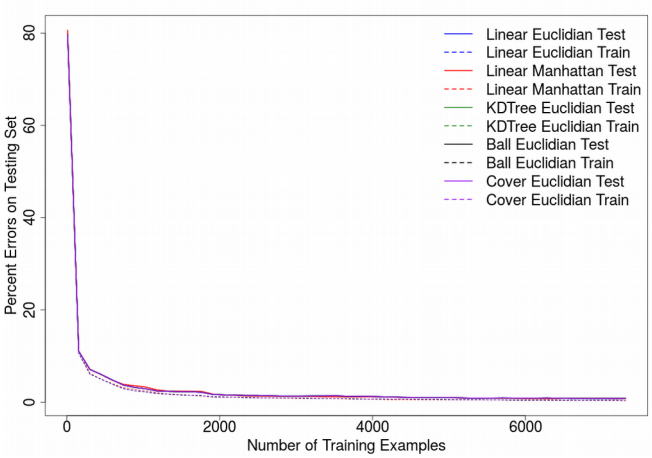

Figure 18: Performance of kNNs as a function of the pendigits training set size.

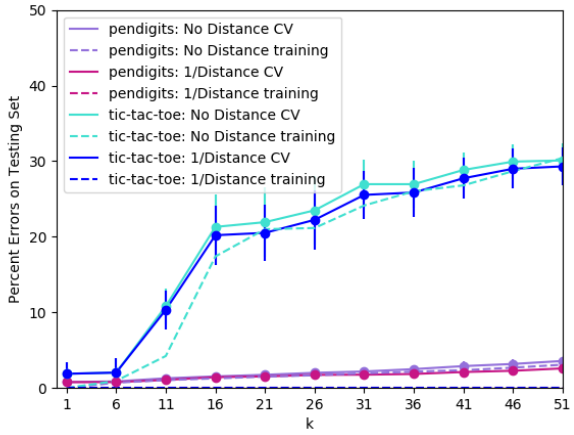

Figure 19: CV Percent Error of kNNs with different distance metrics as a function of k.

As mentioned previously, as *k* keeps growing, the model becomes worse at the task since the model takes the points that are not closely related to the query point into consideration. Since both of the datasets perform well with relatively small values of *k*, it can be said that training and testing sets are not very different (extracted from the same domain). This is no surprising for tic-tac-toe dataset since there are only finite amount of end game configurations. For pendigits dataset however, this is partly caused by the fact that the data-points collected from the same writers exist in both training and testing datasets. In fact, if dataset was trained and tested on the scripts of different writers, the optimal *k* might had been even bigger.

The distance function used doesn't change the accuracy much for both of the datasets. No distance weighting is slightly better than the inverse of the distance, which suggests that the variance of the model increases as the number of contributing neighbors increases. kNN performs pretty well on both of the algorithms with a peak around 97.5% accuracy for tic-tac-toe dataset and 98.5% accuracy for pendigits dataset. As mentioned previously, classes in pendigits datasets are probably clustered more closely leading to a better performance for kNNs. Even though 1% difference doesn't seem much, relative error of tic-tac-toe dataset is slightly less than double the error of pendigits dataset. There are many studies devoted to get the last 5% right (tuning the model for near perfect performances). There are tasks in which that difference might be crucial like cancer screens.

III. Comparison of SL Algorithms

Up until this point, various aspects of specific supervised learning algorithms were discussed. However, comparison between different supervised learning algorithms were limited to a few lines. This section will briefly compare all the supervised learning algorithms discussed so far for both of the datasets.

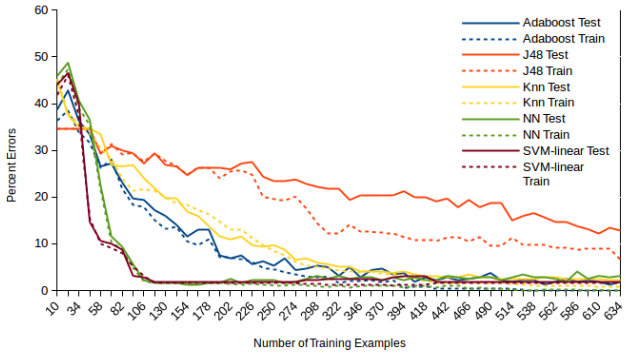

Figure 20: Percent errors of all the models in respect to the tic-tac-toe training set size.

Figure 20 shows the percent errors for all the algorithms discussed for tic-tac-toe dataset. It can be seen from the graph that J48 decision tree has a higher error rate than the other algorithms. Linear support machines have the steepest learning curve. The percent errors of boosting, k-nearest neighbors, neural networks, and support vector machines become pretty close as the size of the training set grows. Neural networks and support vector machines are the most successful algorithms for the tic-tac-toe dataset since they can achieve a competitive accuracy with fewer examples. Both of the algorithms train pretty fast for this dataset, neural networks requiring slightly more time. Given that tic-tac-toe dataset encodes a binary classification problem, it makes sense for support vector machines to perform well on it, since it also seems to be linearly separable. The winner for tic-tac-toe dataset is support vector machines for these reasons: trains quicker than neural networks, easy to store the model (just a linear hyperplane separating coordinate system), and achieves the least error rate of 1.875% (almost, NN model actually achieves an error rate of 1.25%, but it is not our favorite due to the listed reasons.)

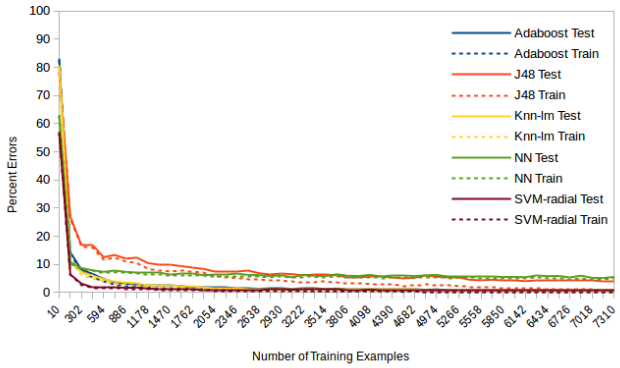

Figure 21: Percent errors of all the models in respect to the tic-tac-toe training set size.

Figure 21 shows the percent errors for all the algorithms discussed for pendigits dataset. It can be seen from the graph that, just like the tic-tac-toe dataset, J48 decision tree has a higher error rate than the other algorithms. Neural networks are the second worse, they learn quicker than the decision trees but have nearly the same error as the decision trees. Boosting and k-nearest neighbors perform equally good. Support vector machines with radial kernel function have the steepest learning curve. For pendigits dataset, time required to train/test neural networks is the highest. However, boosting, k-nearest neighbors and support vector machines perform equally in terms of the time consumption. K-nearest neighbors algorithm is pretty memory intensive, given that pendigits dataset has around 10,000 data points. Least error of 0.74% is shared among boosting and support vector machines. I particularly find boosting methods interesting because I think there is a lot to be discovered to make boosting even better. However, for pendigits dataset, support vector machines with radial kernel functions perform better: steeper learning curve (less examples required for training) and takes a little shorter time to train/run.

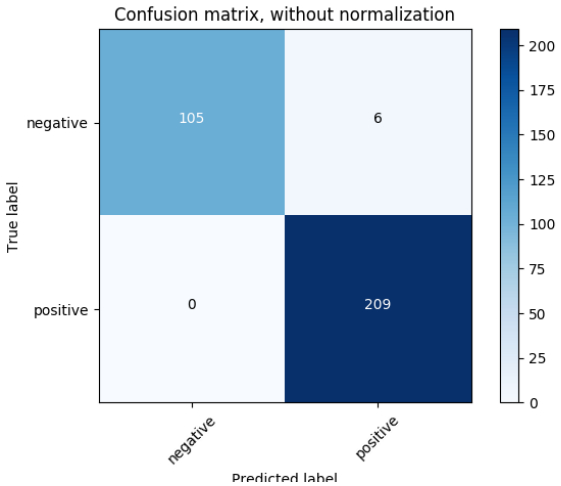

Figure 22: Confusion Matrix without normalization, SVM with Linear Kernel, Tic-Tac-Toe Dataset

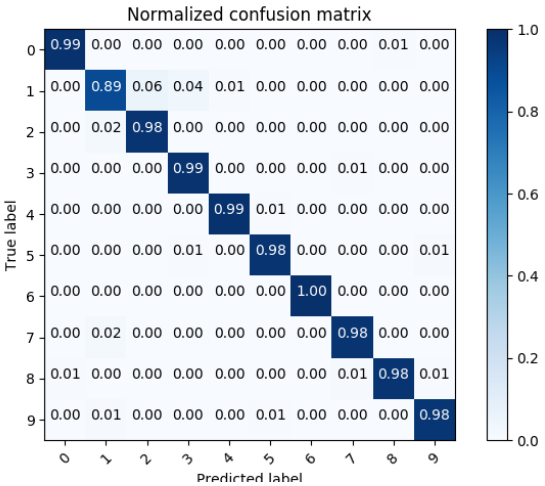

Figure 23: Normalized Confusion Matrix, SVM with RBF Kernel, Pendigits Dataset

Figure 22 shows the confusion matrix for the model with highest accuracy on tic-tac-toe dataset. It can be seen that the diagonal is maximized so are negative precision and positive recall. There are a few negative instances that the model incorrectly classifies as positive. This is expected since the dataset is unbalanced; there are more positives than there are negatives.

Figure 23 shows the confusion matrix for the model with highest accuracy on pendigits dataset. It can be seen that the diagonal is maximized. The model has the least accuracy for 1s, which are confused with 2s and 3s. To further increase the accuracy, one should focus on getting the 1s right. Algorithms focusing on the wrong predictions can be useful in this case.

Decision trees performed the worst in both of the datasets. That might be because of the fact that decision trees are not suitable for data points that are not smooth (specifically for the tic-tac-toe dataset.) Neural networks work well but they are time intensive. K-nearest neighbors algorithm demand a lot of storage space and hard to move around. Boosting doesn't do bad for neither of the dataset, perhaps other forms of boosting might yield even better results. Support vector machines are nice because they can make "full" use of the features of the dataset. If the dataset is nonlinear and high dimensional, kernel trick helps finding the best way to train the model.

REFERENCES

[1] *Alpaydin, E., & Alimoglu, F. (1998). UCI Machine Learning Repository: Pen-Based Recognition of Handwritten Digits Data Set. Retrieved February 05, 2017, from http://archive.ics.uci.edu/ml/datasets/Pen-Based Recognition of Handwritten Digits*

[2] *Aha, D. D. (1991). UCI Machine Learning Repository: Tic-Tac-Toe Endgame Data Set. Retrieved February 05, 2017, from https://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe Endgame*

[3] *Caruana, Rich. Niculescu-Mizil, Alexandru. (2006) An Empirical Comparison of Supervised Learning Algorithms, ICML. Ithaca, NY.*

[4] *"Intro to Machine Learning Course | Udacity." Course | Udacity. Udacity, n.d. Web. 05 Feb. 2017.*