Assignment 1: Supervised learning

Rohan D. Kekatpure

References and links

1. MNIST handwritten digit database.
   http://yann.lecun.com/exdb/mnist/
2. Ionosphere data at UCI.
   https://archive.ics.uci.edu/ml/machine-learning-databases/ionosphere/ionosphere.names
3. B. Kegl and R. Busa-Fekete *Boosting products of base classifiers*
   https://www.lri.fr/ kegl/research/PDFs/KeBu09.pdf

---

1. Description of the datasets

This assignment asks us to apply five supervised learning algorithms on two datasets. The main goal of this exercise to understand the trade-offs in using these algorithms for real-world data mining and predictive modeling tasks. To accomplish this, one needs to identify non-trivial datasets rich in features and size to allow such comparative study. I have chosen the MNIST [1] and the Ionosphere observations [2] datasets for my study.

1.1. What makes these datasets interesting?

The MNIST is a classic image recognition benchmark dataset consisting of 60000 labeled images of handwritten digits in the training set and 10000 in the testing set. Ionosphere observation dataset is 350 observations of 34 attributes with an objective to classify them into 'good' 'bad' observations. I believe the following factors qualify these datasets as 'interesting':

MNIST

1. It has been used to benchmark a number of learning algorithms in image recognition, allowing comparisions to published studies.
2. Each training example is a 28×28 pixels image giving a dimensionality of 784. At this scale, dimensionality is expected to affect model performance, especially for 'sampling' type of algorithms (e.g. nearest-neighbor-type algorithms).
3. Its a multi-class classification problem. While this makes it difficult to benchmark performance of some inherently binary classifiers, it is problem representative of the real world classification tasks we're likely to encounter.
4. By using this dataset I'm taking the risk of not being able to train some algorithms (boosted trees) on the full dataset on my laptop. However, I've always been curious about performance of different learning algorithms on MNIST and would like to see how far I get given my limited computational resources.

Ionosphere

For our studies, we include 150 observations in our training set and use the remaining 201 observations for testing. We don't know the train/test split use by the published studies, which might affect how closely we're able to reproduce these results.

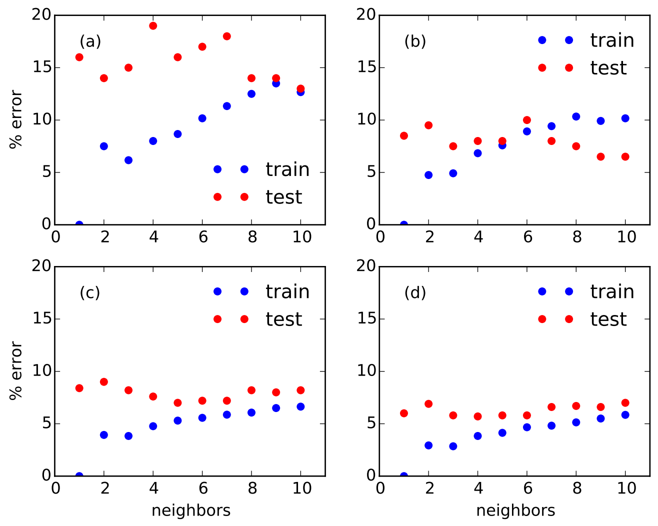

Fig. 1. Error evolution of KNN algorithm with number of nearest neighbors for different sizes of the training set. The value of N_{train} are as follows: (a) 600, (b) 1200, (c) 3000 and (d) 6000. In all subfigures, test error is depicted in red and training error is blue.

1. Ionosphere is a binary classification problem with 34 attributes. Its binary nature allows an easier comparative analysis of various binary classification algorithms.
2. Ionosphere data is relatively smaller in comparison to MNIST. Its smaller size will allow us to efficiently train our models on a laptop. We plan to use this data especially to learn more about boosted tree ensembles.

2. Supervised learning on the MNIST data

2.1. k nearest neighbors (KNN)

Given a test instance, KNN performs a (linear) lookup in the training set and returns the label of the instance closest to the test instance according to some distance metric. For $k > 1$, KNN performs a majority vote between the labels. For the MNIST data, the training examples are 1 dimensional arrays with size 784 and the individual elements having integer values between 0 to 255. The Euclidean distance between two image arrays a and b is defined as:

$$d(a,b) = \left[\sum_{i=0}^{784}(a_i - b_i)^2\right]^{1/2} \tag{1}$$

Figure 1 shows the evolution of the error (defined as misclassification rate) for the training and the test sets as we vary the number of nearest neighbors. Two major conclusions are evident from the figure. First, because of the high dimensionality of the problem, the KNN algorithm is expected to need large amounts of the data to sample any reasonable neighborhood of a given test instance. Consequently the model performs better as the dataset size is increased. The cost, however, is a linearly increasing lookup time. For a training set size of 6000, the lookup for all training examples takes over 30 minutes. Therefore all results are reported only up to a $N_{\text{train}} = 6000$ sample size.

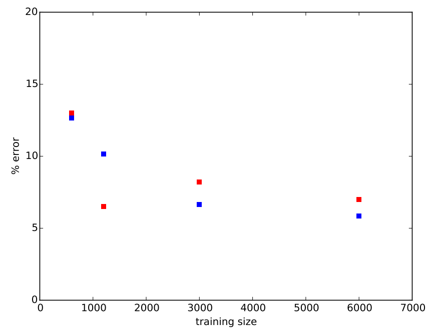

Fig. 2. Misclassification rate of the KNN algorithm with training set size

Second, the training and test errors show opposite variations with increasing k, the number of nearest neighbors. Training error *increases* with k while the test error *decreases*. With a small k, the algorithm is averaging over a small local region around the test instance causing inclusion of more noise. In other words, with small k the model is overfitting. Increasing k and averages over larger sample lead to better generalization and smaller test error (Fig. 2).

Model improvement: Given the simplicity of the KNN algorithm, it is remarkable that it is able to achieve a $\sim 95\%$ accuracy on an image classification task with only 6000 training samples. Because of the high dimensionality of the problem, the error is expected to decrease even more with increase in the size of the training data, albeit at the cost of increased prediction time. The fundamental issue of nearest-neighbor classifiers is the curse of dimensionality. One way to improve performance of the KNN model might be to use dimensionality reduction techniques like principal component analysis (PCA) and perform the lookup in the space of reduced dimensions.

2.2. *Decision trees*

Single decision trees tend to perform well on problems with a small number of discrete attributes. The MNIST dataset, in contrast, has 784 attributes that are practically continuously spaced (0 through 255 with a spacing of 1). As such, we'd be justified in having doubts about performance of single decision trees for this dataset.

We ran decision tree algorithm using the `rpart` package in R. In addition to rapidly computing the trees, `rpart` is also able to perform pruning using the depth of the tree (`maxdepth` parameter) as well as the marginal information gain (via the `complexityParameter`). To determine the optimal tree size, we first studied the error evolution as a function of the tree depth.

Figure 3 shows the variation of the training and test error with increasing tree depth. Although not visible at the scale of the figure, the test error *increases* for `maxdepth` > 15, clearly signifying overfitting. The performance of the model is significantly worse compared to the KNN. The minimum error rate for 6000 samples (corresponding to Fig 1(d)) is $\sim 21\%$ almost four times that of KNN trained on similar volume of data. Even training the model on the full dataset of 60000 samples improves the error rate to $\sim 12\%$, still double that of KNN trained on 1/10th of data.

Next, we fixed `maxdepth` $= 15$ and studied the model performance (as measured by misclassification error) with number of nodes. For $N = 6000$ training samples, there is a clear

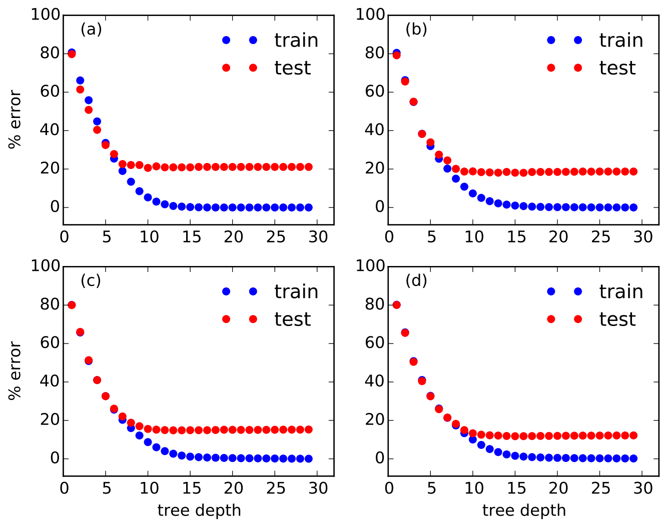

Fig. 3. Misclassification rate of the decision tree algorithm with depth of the tree for varying dataset sizes. The size of the training data is (a) 6000, (b) 12000, (c) 30000 and (d) 60000 training samples.

evidence of overfitting with increasing number of nodes (model complexity). For $N = 60000$ training examples, the over fitting is less evident, but is numerically present.

Figures 3 and 4 suggest that optimum pruning point for a *single* decision tree algorithm would be around `maxdepth` $= 15$ and `complexityParameter` $= 10^{-4}$ resulting in trees with about 1000 nodes.

Model improvement: In the present experiments, a single decision tree trained on the entire dataset is able to achieve only about 12% misclassification rate on the test set. This is consistent with the general observation that decision trees are prone to overfitting especially for datasets having a large number of closely spaced features such as in the MNIST dataset. The overfitting also suggests that ensembles of trees, either in the form of random forests or boosting, may perform better.

2.3. Boosted tree ensembles

Boosted decision trees have performed well on the MNIST dataset [3]. However the iterations needed to obtain these performances are prohibitively high for our computing resources. For example, the 1.26% test error rate requires $T = 10^5$ iterations on the full MNIST dataset [3]. In contrast, our training times were ∼5 minutes for 100 iterations on a 10% sample of the data (6000 training examples). As a result, we seek to evaluate boosted tree ensembles differently.

We fix the sizes of the training and testing sets to 6000 and 1000 examples respectively. We then train a single large tree with `maxrepth` = 30 and 400 nodes and a boosted ensemble with 200 base learners. Our boosting is performed using the AdaBoost.M1 algorithm in R's `adabag` package. Finally we contrast the performance of these three learners.

The base learner for this section is a decision tree of `maxdepth` of 5 with 50 nodes. The training and test errors of the base learner are 31.45% and 32.4% respectively. While the base learner is clearly underfitting, it satisfies the criterion for being a weak learner.

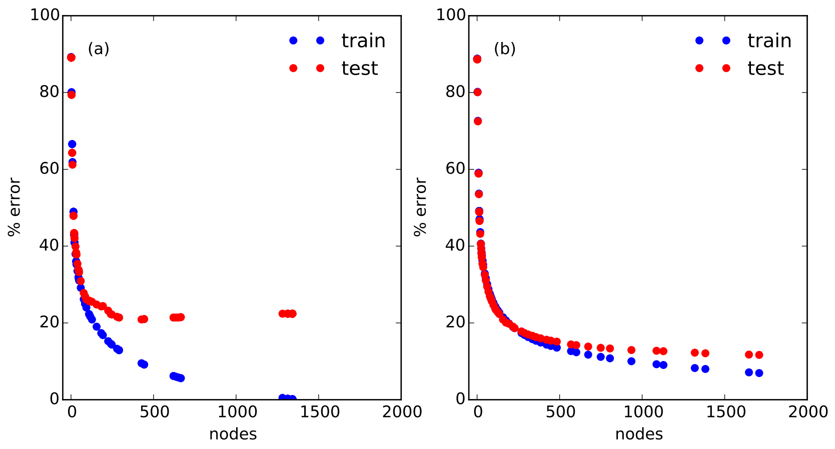

Fig. 4. Misclassification rate of the decision tree algorithm increasing number of nodes. The size of the training data is (a) 6000, (b) 60000 training samples.

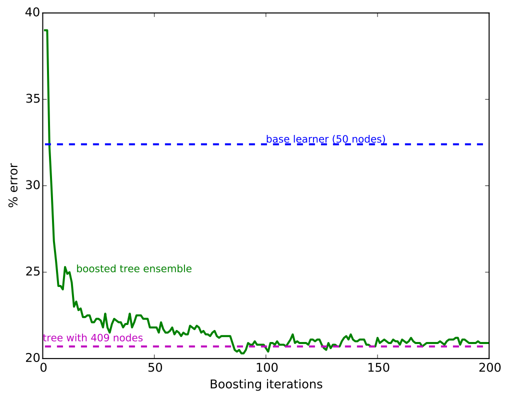

Fig. 5. Misclassification rate of the a single base learner with 50 nodes (blue), a boosted ensemble of base learners (green), and a single learner with 409 nodes. (magenta)

As evident from Fig 5, boosting significantly improves performance of a single base learner. To get a comparable performance from a single tree, one needs a tree with over 400 nodes. Such large trees are susceptible to overfitting over unseen test data (although they don't in this particular instance).

Model improvement: Trees ensembles are preferable over single large trees because of better generalization. However, as the MNIST dataset illustrates, boosted trees are costly to train for high dimensional data. To improve our model here, clearly we need to increase the training data size and the number of boosting iterations. Both of these improvements are outside the range of our present computational resources for the present dataset. One option for model improvements for the present dataset might be to use random forest ensembles instead of boosted trees since the former tend to be more computationally feasible.

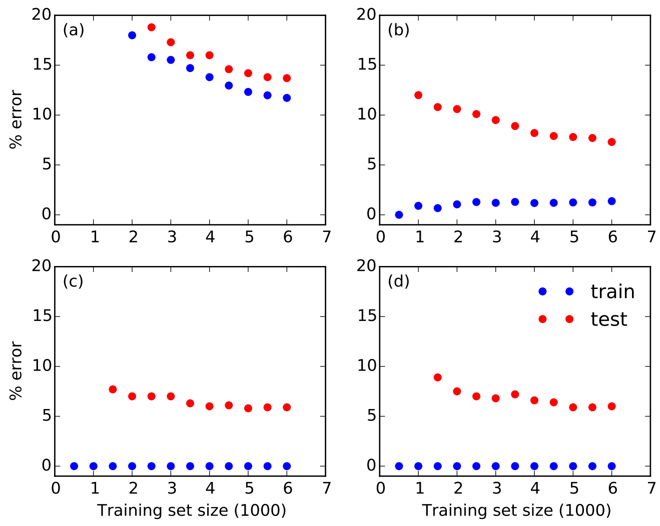

Fig. 6. Misclassification rate of the SVM algorithm with RBF kernel as a function of training size for different values of the cost parameter C. (a) $C = 0.1$, (b) $C = 1$ (c) $C = 10$ (d) $C = 10^5$.

2.4. *Support vector machines (SVM)*

Support vector classifiers seek to find the optimum separating hyperplane while still minimizing the misclassified points. Mathematically, a (linear) SVM problem is expressed as:

$$\arg\min_{w} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i$$
$$\text{subject to } \xi_i > 0 \text{ and } y_i(x_i^T w) > 1 - \xi_i \tag{2}$$

where C is the 'cost' parameter that determines the relative importance of the mis-classified points in computing the optimum hyperplane. To see this, note that making $C \to \infty$ will de-emphasize the minimization of 'weights' and fit a complicated boundary to the training data. In contrast, making $C \to 0$ will tend to de-emphasize minimization of mis-classification cost. In other words, large values of C tend to overfitting and small values of C tend to underfit. The optimum value of C is found from cross validation.

Below we study the performance of SVM models trained on MNIST data using radial basis function (RBF) and polynomial kernels.

2.4.1. SVM with radial basis functions (RBF) kernel

While packages in R allow us to perform automatic cross-validation, it is instructive to visualize evolution of training and test errors with training data size for various values of C. Figure 6 plots this error evolution for $C = 0.1, 1.0, 10.0$ and 10^5.

The training and the testing errors show a clear trend until $C \leq 10$. The model is less sensitive to C for $C \gtrsim 10$. An inspection of Eq. (2) reveals that this behavior is expected at some threshold value of C. After this threshold, the 'weights' term is fully de-emphasized giving an essentially unregularized model. Yet there is no evidence of overfitting. This is possibly because there is still enough training data as apparent from the decrease in the test error with increasing data.

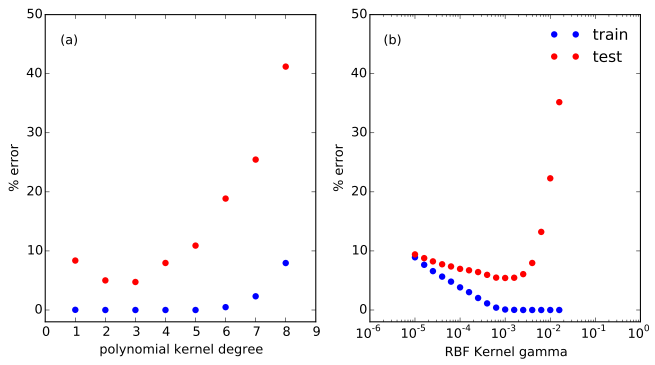

Fig. 7. Misclassification rate of the SVM algorithm with (a) polynomial kernel as a function of its degree and (b) RBF kernel as a function of parameter γ. For both cases, the model is trained on 6000 samples and the 'cost' parameter is fixed at $C = 10$.

Table 1. **SVM best results:** Training and test errors of the optimized SVM models with RBF and polynomial kernels on the full MNIST dataset (training set = 60,000, test set = 10,000 instances).

Kernel	training error	test error	training time
RBF, $C = 10, \gamma = 0.001$	0.19%	2.20%	30 minutes
Polynomial, $C = 10$, degree = 3	0.77%	2.64%	30 minutes

2.4.2. Comparisons of SVM with polynomial and RBF kernels

For this study we fix the 'cost' parameter to $C = 10$. The complexity of the polynomial kernel is tuned by the degree of the polynomial. We study the training and the test error as a function of the kernel degree. For the RBF kernel, the complexity is controlled by the γ parameter in the kernel ($K(x,y) \equiv e^{-\gamma\|x-y\|^2}$). The model is trained on 6000 samples and tested on all 10000 samples.

For the polynomial kernel, the test error is minimum (= 4.74%) for a degree 3 polynomial kernel. The test error increases rapidly for higher degree polynomials due to overfitting. For the RBF kernel, there is a clear evidence of a transition from underfit to optimal fit to overfit as the width parameter γ is varied. The optimum point for RBF kernel gives a minimum error of 5.42 % and was located using cross validation. γ was then varied about this point to study the effect of this variation on training and testing errors.

As a final test, we trained the optimized SVMs with RBF ($\gamma = 0.001$) and polynomial (degree = 3) kernels on the full MNIST dataset and obtained the performance reported in Table 1.

Model improvement: The RBF and Polynomial SVMs are able to achieve respectively 2.2% and 2.64% test error rates on the full MNIST dataset. The state of the art reported by the LeCun et al for SVMs is 1.4% [1]. The reason for this difference is as follows. Our model hyperparameters (C, γ and degree) were obtained from cross-validation on a 10% sample of the full data. Therefore, one obvious way to improve the model performance is to determine the hypermarameters by cross validation on the *full* MNIST dataset. Additionally it might be worthwhile to explore effect of data normalization: centering + unit variance.

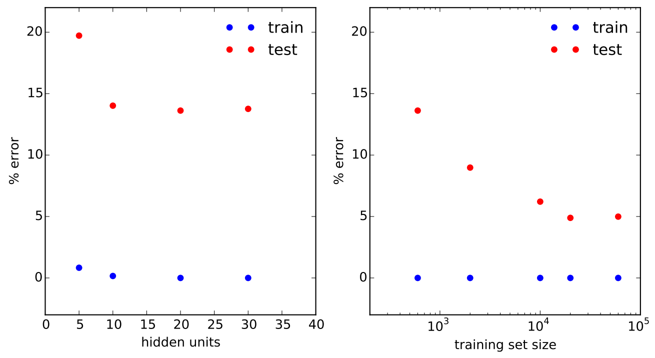

Fig. 8. Misclassification rate of the one-hidden-layer feed-forward neural network as a function of (a) number of hidden units and (b) training set size

Table 2. **Neural network best results:** Training and test errors for a 50 unit, one-hidden-layer feed-forward neural network (training set = 60,000, test set = 10,000 instances).

Neural network	**training error**	**test error**	**training time**
1-hidden layer, 50 hidden units	0.00166%	2.57%	3.5 hours

2.5. Neural networks

Our neural net architecture of choice is a feed-forward neural networks with a single hidden layer. With this choice, the only free parameter is the number of units in the hidden layer. The number of hidden units used for the MNIST dataset in the literature has varied between 300 and 1000 [1]. We found that training even a single-hidden layer feed forward NN with > 30 hidden units takes roughly 10 minutes on our laptop and prohibits exhaustive experimentation. We therefore study the behavior of the algorithm up to a modest number of hidden layer nodes (~ 50) and try to infer any visible trends.

Our neural net is implemented using the `nnet` library in R. This library is capable of modeling feed-forward neural networks with a single hidden layer and is a good choice given our computational resources. To study the neural network, we varied the number of hidden layer units between 5 and 30 and the sample size drawn from the training set between 600 and 60000 (i.e. 1% to 100% of the MNIST dataset). The test set size was 10000 for all the observations.

The evolution of the training and test errors are plotted in Fig. 8. The test error first decreases with number of hidden units (N_h) and saturates between $10 < N_h < 20$. The test error, however drops from 14.0% to 4.9% as the size of the training set is increased from 600 to 20000 samples and plateaus thereafter, indicating an optimal fit. For these parameters, the network training time also increased from 414 seconds to about 1800 seconds (30 minutes). As a final test, we trained the neural network with 50 hidden units on the entire MNIST training data and obtained the results displayed in Table 2.

Model improvement: The values reported by LeCun for a one-hidden-layer net with 300 hidden units is 4.7%. This compares favorably to our value of 4.9% obtained with just 20 hidden units. With 50 hidden units the test error drops to 2.57% and performs better than some reported models which employ pre-processing and up to 1000 hidden units[1]. It seems that we've pushed the single-hidden-layer neural network to its limit (without using any pre-processing). As suggested by the MNIST website [1] further improvements will need us to either use multi-

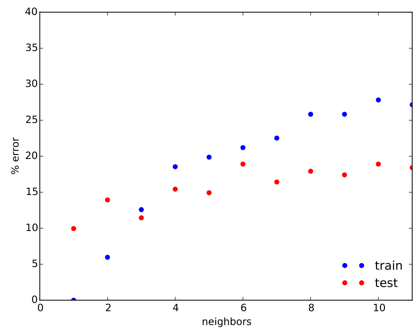

Fig. 9. Misclassification rate of the KNN algorithm on the ionosphere training and test data for different nearest neighbors

layer networks or use the domain knowledge of the images to perform preprocessing operations such as de-skewing and elastic distortions.

3. Supervised learning algorithms on the Ionosphere data

We now turn to applying various learning algorithms to our second dataset, the ionosphere data. The algorithms perform mostly similarly to how they did on the MNIST data. Our descriptions will therefore be shorter compared to the previous section; the differences, when they arise, will be pointed out.

3.1. k nearest neighbors (KNN)

As mentioned previously, accurate predictions using KNN requires large amounts of data, especially as the number of attributes increase. Here we have 150 data points to determine nearest neighbors in 34 dimensions. We trained on the entire training data and varied nearest neighbors from 1 to 11. As depicted in Fig 9, $k = 3$ achieves the lowest error rate of 11.44%. As we increase k, the model suffers from under-fitting as evidenced by increase in both training and test errors.

Model improvement: Although use here for comparative purposes, KNN is not a good model for this dataset because of the small amount of available training data relative to number of dimensions. As with MNIST, one could explore reducing dimensionality of the problem to get rid of correlated features and make the KNN model work better. One could also perform a different train:test split of the input data. But that is cheating and is likely not possible in real life :)

3.2. Decision trees

This is an ideal problem for decision trees as we have a moderate number of discretely valued input features. As in the case of MNIST we determined the optimum pruning point by varying the tree depth from 2 (stump) to 30 (max allowed by `rpart` package in R). Fig. 10 shows the variation of the training and testing errors for trees of increasing complexity (depth). From the figure, a decision tree with depth of 2 achieves an 10.1% error rate an appears to be an optimum pruning point.

Model improvement: Achieving optimum performance at `maxdepth` = 2 indicates that a single feature is dominating the information gain in the data. It also appears that at `maxdepth`

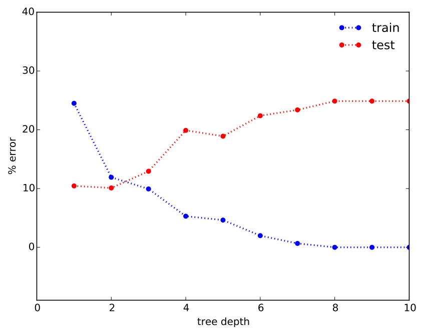

Fig. 10. Misclassification rate of the single decision tree algorithm for increasing tree depth.

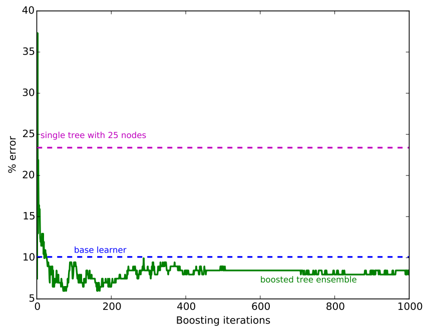

Fig. 11. Misclassification rate of boosted tree ensemble with number of boosting iterations (base learners).

= 2 the training error is still 11% which indicates underfitting. As we increase the tree depth, the training error goes down, but and the test error goes up. Both these aspects strongly suggest that gathering more data is likely to improve model performance. It might also be worthwhile to see if tree ensembles lead to better generalization. However, if the underlying issue is the lack of adequate data, then boosting is unlikely to produce significant benefits.

3.3. Boosted tree ensembles

We construct boosted ensembles of weak trees using the AdaBoost.M1 algorithm provided by the `adabag` package in R. Fig. 11 shows the results of fitting a boosted tree to the data. The base learner here is a stump achieves about 10.1% error rate. We applied 1000 boosting iterations to this base learner which brought its error rate down to about 7.96%. For this dataset, boosting provides a modest 2% improvement over individual trees.

Model improvement: It appears that the accuracy has plateaued after 1000 iterations. As seen in the previous section, the issue in this case appears to be inadequate amount of data. This is perhaps the reason why we saw boosting provide only modest gains. As such, the recommendation in the previous section (‘gather more data’) still applies.

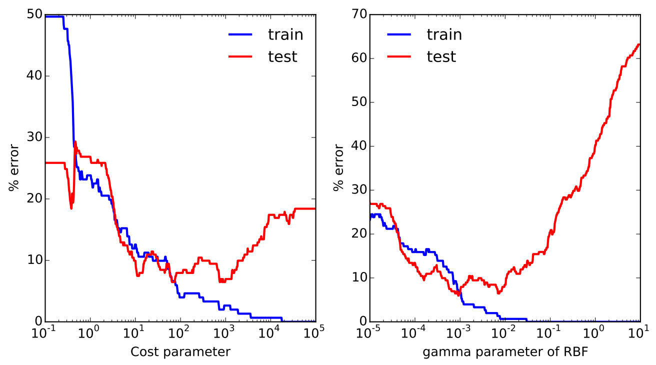

Fig. 12. Misclassification rate of the SVM algorithm under variation of (a) the cost (regularization) parameter for $\gamma = 10^{-3}$ and (b) the kernel width γ for cost $= 70$.

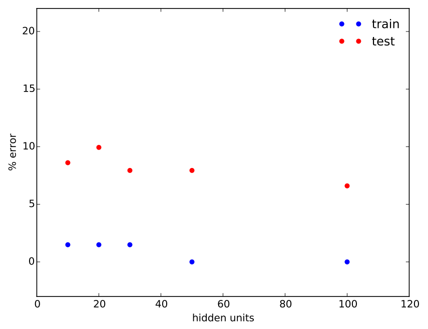

Fig. 13. Misclassification rate of one-hidden-layer feed-forward neural networks with different number of hidden units.

3.4. Support vector machines (SVM)

SVMs were implemented using the `e1071` package in R. Fig. 12 shows the variation of training and test error for variation in the 'cost' parameter (which controls regularization) and the kernel width γ. The results show that the optimum point is obtained at cost $= 71$ and $\gamma = 10^{-3}$ leading to a test error of 6.5%.

Model improvement: The accuracy numbers obtained by SVMs, $\sim 93.5\%$ are comparable to the ones reported in the literature (using other algorithms) [2]. It also seems that the optimum point is obtained where the model is still underfitting. This calls, as before, for more training data.

3.5. Neural networks

The final model we train on the data is a one-hidden-layer, feed-forward neural network with varying number of hidden units. As before we used R's `nnet` package. With neural nets we're able to achieve 6.6% test error rate for 100 hidden units (Fig. 13).

The small size of the training data allowed us to reach network sizes up to 100 hidden units at which point the error plateaued.

Table 3. Comparison of different algorithms. The reported numbers for MNIST can be found in [1] and those for the ionosphere study can be found in [2]

Dataset	**Algorithm**	**Training set size**	**Test error**	**Best reported**
MNIST	KNN, $k = 10$	6000	6.6%	5.0% using full data
MNIST	single decision tree	60000	12.0%	unknown
MNIST	boosted trees, 200 iters	6000	20%	1.26% with 10^5 iterations, using full data
MNIST	SVM, $C = 10, \gamma = 0.001$	60000	2.2%	1.4%
MNIST	Neural net, 1 hidden layer, 50 hidden units	60000	2.57%	4.7%
Ionosphere	KNN, $k = 3$	150	10%	9% (train/test split unknown)
Ionosphere	single decision tree	150	10%	unknown
Ionosphere	boosted trees, 1000 iters	150	7.96%	unknown
Ionosphere	SVM $C = 71, \gamma = 0.001$	150	6.6%	unknown
Ionosphere	Neural net, 1 hidden layer, 100 hidden units	150	6.5%	4.5% (train/test split unknown)

Model improvement: Figure 8 indicates how crucial the size of the training data is for neural net performance. So even for neural nets, improving performance will depend on getting more training data for this problem. In addition, it might be worthwhile to explore data normalization to see if it results in extra performance gains.

4. Model comparison

Finally we summarize our studies by collecting in one place the performance of our implementation and comparing them with the numbers reported in the literature for these datasets. The primary limiting factor for our algorithms has been the training time or, equivalently, the computational resources. Also in the case of ionosphere data, the train/test split is not known for to enable us to make fair comparisons (we cannot access the original papers).