Anish Moorthy

CS 4641 Project 1: Supervised Learning

Datasets

CIFAR-10: For my first dataset, I chose the CIFAR-10 dataset. Here, a learner's objective is to classify 32x32 color images into one of ten classes (e.g. airplanes and horses). The dataset consists of 60,000 image/label pairs, with each of the classes represented equally at 6,000 instances each. The task of image classification/processing is in general a very interesting problem with a plethora of real-world. However, it is also difficult. First, the data is relatively high-dimensional (3072 attributes). To reduce problem complexity somewhat, I grayscale all images and standardize (zero centerer and unit variance) the data. However, this preprocessing does not change the fact that in images, dimensions are highly locally interlinked: it is difficult to learn anything (or at least, anything useful) from a single pixel, for instance. Being able to exploit locality of information is therefore a key determinant in the performance of the algorithms demonstrated.

IMDB: For my second dataset, I chose the IMDB Large Movie Review Dataset. The objective here is binary sentiment classification: given a string of words, the objective is to predict whether the review is positive or negative. The dataset consists of 50,000 image/label pairs, with the number of positive and negative examples being equal. This dataset is different from CIFAR-10 in that a single word may encode much more information than a single pixel: for instance, "fantastic," "amazing," or "awful" all strongly indicate a review's label label: though on the other hand, common words (eg "the," "a") reveal almost nothing. However, there are important similarities between natural language comprehension and image classification: the structure of natural language lies not just in words, but in the sequences (sentences) they form. Despite this, I largely destroy the sequential nature of the data by simplifying each review down to a count-of-words vector, with each of the 10,000 most common words corresponding to a dimension. Although the representation is simplistic, it allows the learners to generalize well over the data.

Methodology

My general process for training the learners was as follows: First, I split the data into static train, validation, and test sets. I then trained learners using several hyperparams on the the training set, and compared performances on the validation set. When cross-validation was feasible, I performed it in initial hyperparameter evaluation. When cross-validation would have been too slow, I tried to cross-validate just the top-performing set of parameters to ensure accuracy. The best-performing learner was then used for classification of the test set and generation of further graphs such as Accuracy vs Dataset-Fraction/Num-Iterations.

Note: The exact train/val/test fractions varied for each learner, and are reported at the beginning of each section. Test/Total fraction is the fraction of data used as the testing set, while Val/Non-Test is the fraction of *non-test* data used for validation. In all Accuracy vs Dataset-Fraction curves, the fraction reported on the x-axis is the fraction of *non-test* data used in training. Thus at no point or context is a model *ever* trained on elements designated as part of the holdout set for a given class of learner.

Decision Trees

In all configurations in this section, I use the entropy/information metric to construct the decision trees. I also tested the below configurations using the GINI metric, but changes in performance were minimal. Furthermore, I was unable to find an efficient implementation of post-pruning. Therefore, I resorted to pre-pruning my trees by specifying the maximum number of leaf nodes, maximum depth, and minimum number of examples required to split a node.

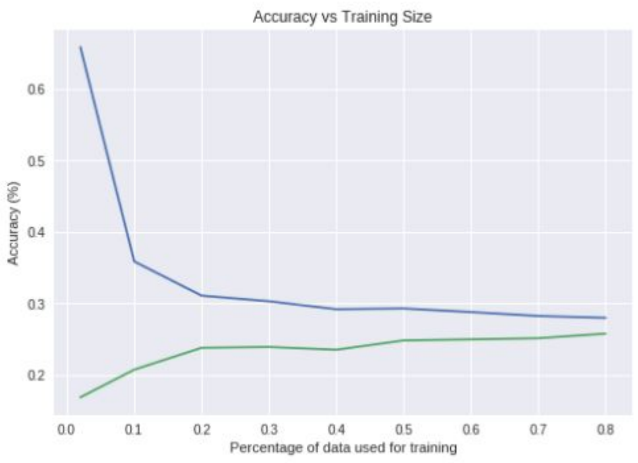
CIFAR-10 :: (Test/Total = .2, Val/Non-Test = .2)

Max Depth	Min Split	Max Leaves	Wall Clock (min)	Train Acc	Val Acc
1	2	-1	0.45	0.14	0.13
5	4		2.12	0.23	0.23
10	2		4.13	0.39	0.25
10	20		4.03	0.38	0.24
20	30		5.10	0.53	0.23
-1 (no max)	2		5.49	1.00	0.22
-1	300		3.85	0.31	0.24
2	2		0.85	0.18	0.17
20	2		5.43	0.95	0.22
40	2		5.45	1.00	0.22
-1	20		5.38	0.64	0.24
-1	60		4.90	0.43	0.25
-1	100		4.73	0.38	0.25
-1	2	10	2.00	0.22	0.21
-1	2	100	3.63	0.28	0.25
-1	2	1000	5.02	0.43	0.25
-1	2	10000	5.45	1.00	0.22

After testing many configurations on the data, I was surprised to find that most seemed to perform similarly, even in cases where the some hyperparameters had seemingly very different values. For instance, the runs with maximum depth of 5, 20, and 40 and similar other parameters achieved 22-23% accuracy. The same phenomenon occured as I the min split was varied from 20 to 100 in a grid search as well.

The clustering of validation accuracy is puzzling. I thought that perhaps that the extra freedom that large values of depth/leaves yielded was going unused: that is, that tree construction was terminating before hitting restrictive limits. However, the train accuracies refute this claim: even models with very similar validation accuracies diverge in accuracy on the training set. For instance, increasing the Max Laves clearly increases training but not validation accuracy. In the end, I cannot definitively answer why the accuracies of my learners cluster strongly around ~25%. This limit is chosen

The accuracies of Green Row (chosen for accuracy) and Blue Row (chosen for speed, for use in boosting later) rows recalculated using 5-fold cross validation: the results were almost exactly the same, with validation accuracy means and standard deviations being 0.2456/0.219 and 0.0025/0.0031 respectively. The Green Row's test accuracy of 0.247, and was used to generate the following



Note: In this graph, results are mislabeled as percents rather than fractions. Therefore, the accuracies reported here are 65%, 35%, etc and *not* .65%, .35%, etc. The data points here correspond to Data-Fraction=2% and then at multiples of 10% afterwards

Note: Here and elsewhere, the blue and green lines represent train & test accuracies respectively

The shape of the graph is not unexpected: more training data is harder to memorize, but reveals more about the testing distribution. Increasing the size of the training set here might give fractionally better performance, but the learning curve has more or less plateaued: for this set of hyperparameters at least, this performance is probably about optimal.

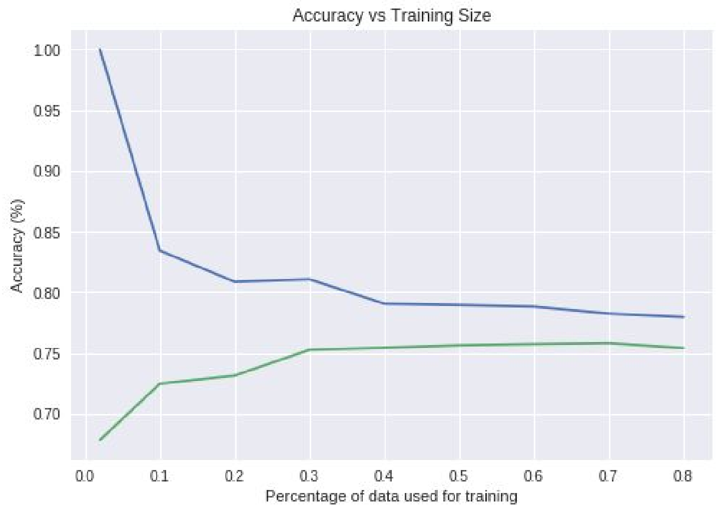
Going into this experiment, I expected that decision trees would learn almost nothing: however they achieve adequate performance on CIFAR-10, performing significantly better than the baseline of 10%. In fact, it's possible that with proper post-pruning decision-tree learners may even be able to significantly improve performance beyond what I've demonstrated

IMDB :: (Test/Total = .2, Val/Non-Test = .2)

Max Depth	Min Split	Max Leaves	Wall Clock (min)	Train Acc	Train Acc std	Val Acc	Val Acc std
1	2	-1	0.200	0.620	0.004	0.615	0.011
5	4	-1	0.733	0.716	0.006	0.702	0.010
-1	2	100	0.580	0.815	0.005	0.733	0.011
-1	2	-1	0.967	1.000	0.000	0.710	0.006
6	50	-1	0.200	0.728	0.008	0.707	0.002

Based on the better-performing sets of hyperparameters on the CIFAR-10 dataset, I reduced the hyperparameters tested on the IMDB dataset (though this itself introduces bias). The accuracies reported here are all calculated via 10-fold cross validation, and as such standard deviations are reported as well. The low standard deviations may be explained by the fact that each learner is trained on a majority of a relatively large dataset: thus, the training distributions may be expected to resemble each other relatively closely. Here again, we observe the tight clustering of validation accuracies. Its repeat appearance in a different problem indicates that this sort of clustering may be a result of the ID3 algorithm (or perhaps this neighborhood of hyperparameters) rather than any peculiarities of the CIFAR-10 dataset

The Green Row achieved an test accuracy of 76%. Data-Fraction values are the same as above



Its performance over Dataset-Fraction has a familiar form. However the plateau in validation accuracy is more pronounced here (and actually occurs at only 50% of the training set). Thus it is unlikely that more data will increase performance

Overall, decision tree performance model on the IMDB dataset was disappointing. I expected decision trees to perform much better here than on CIFAR-10, but results are more modest. Again, however, I suspect that the lack of a proper post-pruning evaluation is a significant handicap to the models I've trained. Therefore, the conclusions which may be drawn from these experiments remain limited

Support Vector Machines

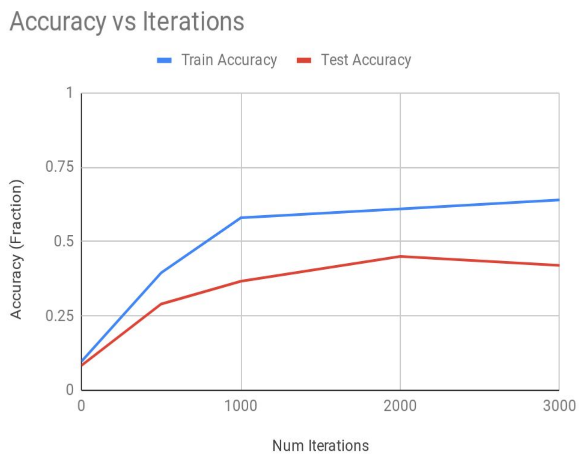
CIFAR-10 :: (Test/Total = .2, Val/Non-Test = .2)

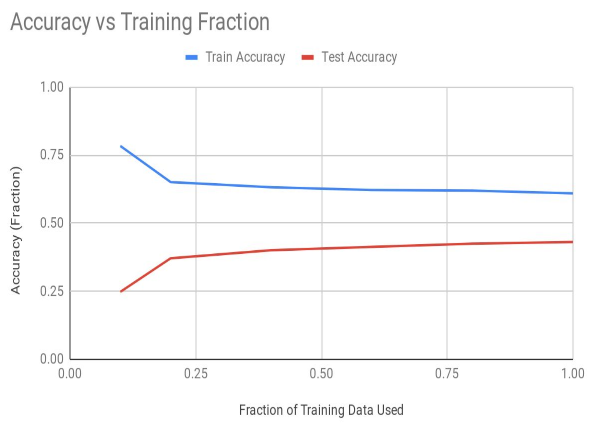
C	Kernel	Max Iter	Wall-clock (min)	Train Acc	Val Acc
1	linear	500	16.35	0.16	0.15
1	linear	2000	26.67	0.17	0.15
1	poly (degree 3)	500	15.68	0.22	0.14
1	poly (degree 3)	2000	19.03	0.51	0.30
1	poly (degree 5)	2000	34.25	0.46	0.22
1	rbf	2000	37.53	0.62	0.44
1	poly (degree 2)	3000	39.50	0.54	0.36
1	poly (degree 3)	3000	41.67	0.64	0.35
1	rbf	3000	38.33	0.62	0.42
0.5	rbf	3000	39.58	0.51	0.41

The single biggest hurdle in training SVMs on this dataset turned out to be time: while the training progressed relatively quickly (the optimizer iteration limits were usually hit within minutes), evaluating the SVM on train/validation data dominated cost

The reason for this is that the cost of prediction in a kernelized SVM is cubic in the number of training examples, and since CIFAR-10 has tens of thousands of examples, slow evaluation times were guaranteed [1]. Linear SVMs have reduced complexity, but their performance is inadequate: not surprising, since the image classes are probably not linearly separable. Polynomial kernels do much better up to a point: the degree 5 kernel overfits in comparison with degree 3 kernel. However by far the best kernel in both train and validation accuracy is the Radial Basis Function: Given more time, it would be useful to test other kernels such as sigmoid to determine whether even RBF accuracy can be surpassed.

Due to large training times, no cross-validation was performed., and the Green Row achieved a test accuracy of 43%. In the Iteration graph, iterations tested were 500, then at multiples of 1000. In the Dataset-Fraction graph, the fractions were .1, then multiples of .2 afterwards. Long evaluation times precluded more granular sampling.



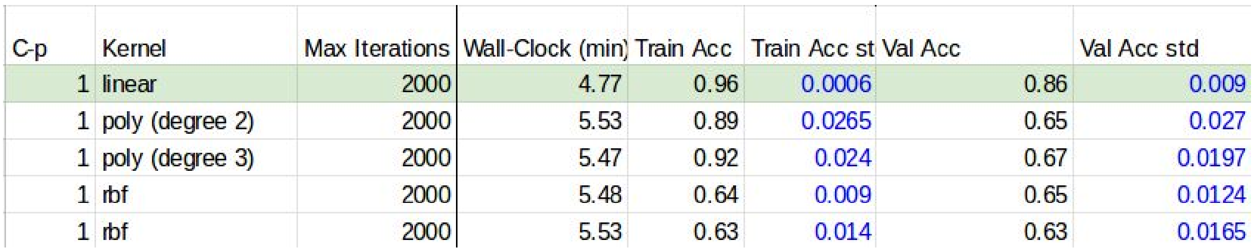


The first graph clearly shows overfitting past the 2000 iteration mark. The second appears to show that while the model has high bias, this reduces steadily, almost linearly, (albeit slowly) given more training data. By increasing the size of the dataset, a test accuracy of 50% may be reachable. However, more data would balloon to tens of hours cubically, which is a significant barrier.

Overall, SVMs yield an extremely significant improvement over decision trees. Unfortunately the complexities associated with kernelization hold them back from being employed to learn on larger datasets.

After training SVMs on CIFAR-10, I questioned whether grayscaling was hurting accuracy. Although it reduces data dimensionality, I believed SVMs might perform better given color. SVMs attempt to form separating hyperplanes (in some transformed space), and RGB format allows simple construction of hyperplanes which separate data based on the amount of RGB it a given pixel- thus allowing for better expressiveness. To test this hypothesis, I retrained the Green Row on non-grayscaled data. This did result in a new high of 51% accuracy on the test set, showing that the dimensionality reduction of grayscaling increases the difficulty of learning

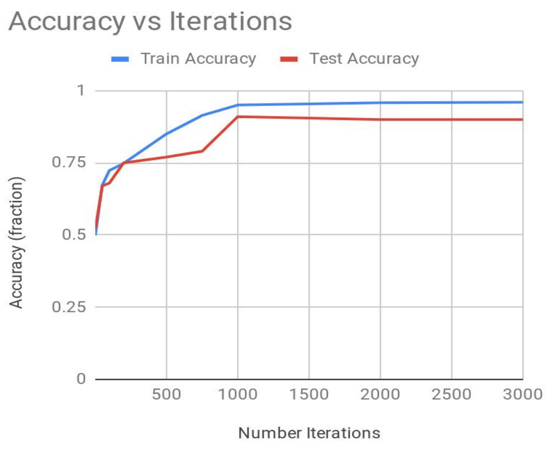
IMDB (Test/Total = .2, Val/Non-Test = .2)

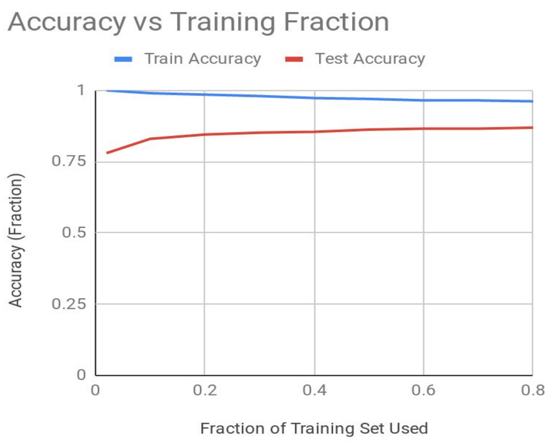


C-p	Kernel	Max Iterations	Wall-Clock (min)	Train Acc	Train Acc st	Val Acc	Val Acc std
1	linear	2000	4.77	0.96	0.0006	0.86	0.009
1	poly (degree 2)	2000	5.53	0.89	0.0265	0.65	0.027
1	poly (degree 3)	2000	5.47	0.92	0.024	0.67	0.0197
1	rbf	2000	5.48	0.64	0.009	0.65	0.0124
1	rbf	2000	5.53	0.63	0.014	0.63	0.0165

The IMDB dataset was much easier to apply SVMs to. Evaluation times were faster, thus allowing accuracy calculation via five-fold cross validation. Interesting, the results here are almost flipped: polynomial and rbf kernels perform almost equally on the validation set, but the linear kernel far outperforms them both. This results are more in-line with my initial intuition about the bag-of-words representation. Because each word is a separate dimension of the feature vector, it is easy to construct a hyperplanes which distinguish samples containing any word (or even a set of words). Therefore the SVM may distinguish examples based upon highly positive/negative words. I have not proven this to be the case, but it is a valid hypothesis.

The Green Row achieved a very high test accuracy of 85%. Below, iterations tested were 1, 50, 100, 200, 500, 750, 1000, 2000, and 3000, while the dataset fractions tested were .02 and then at multiples of .1





As with CIFAR-10, there is evidence of overfitting with iterations, and of a bias gap which reduces with dataset size. Overall, the Linear SVM achieves excellent performance on the task of sentiment analysis.

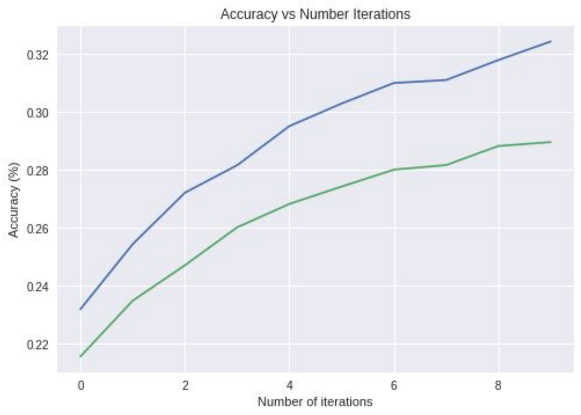
Boosting

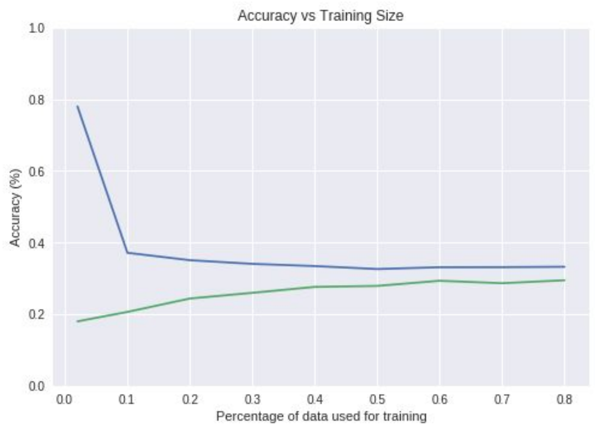
CIFAR-10 (Test/Total = .2, Val/Non-Test = .2)

Parameters				Results	
num estimators	learning rate	Learner	Wall Clock (min)	Train Acc	Val Acc
5	1	Blue Row	11.48	0.3040	0.2690
5	2		11.55	0.2420	0.2140
5	5		9.15	0.1340	0.1304
5	1	Depth 1	0.63	0.2010	0.2105
5	0.5		0.62	0.1940	0.1920
10	1	Blue Row	22.05	0.3303	0.3006
50	1	Depth 1	6.35	0.2610	0.2575

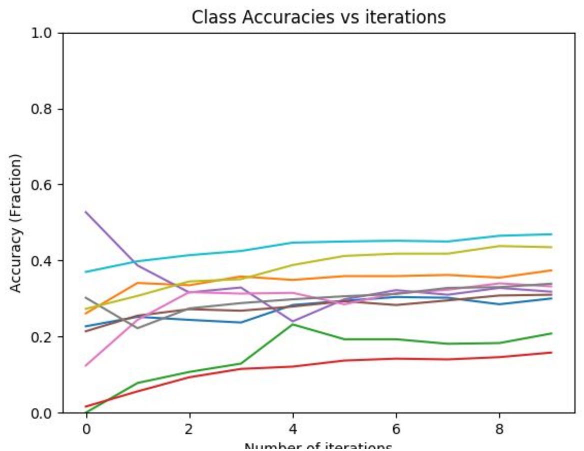
The boosting process performed much more poorly than I expected. On CIFAR-10, the "Depth 1" (a depth 1 decision tree, the default learner) and Blue Row" (refer to decision tree results) learners achieved validation accuracies of 0.13 and 0.23 respectively. Given that "Depth 1" performs only slightly better than random, slow learning was expected there. However, over the course of 10 iterations, the accuracy of Blue Row is only boosted by 7%. Even after another 20 iterations of boosting, validation accuracy only reaches 31%.

Iterations were tested at increments of 1, dataset fractions were .02 and then intervals of .1





To try to understand the slowness of boosting, I tried to visualize the class accuracies over each iteration: the graph is given below (each color is a different class). For the initial lowest two classes, boosting performs as expected, and the class accuracy increases over time. However, the remaining class accuracies remain relatively tightly clustered around 25%. This indicates that the the component weak learner has trouble with telling these classes from one another with the reliability.



Therefore, such classes with similar error rates will tend to stay at similar error rates after a boost, achieving similar weights in the next distribution, and cycling in this manner

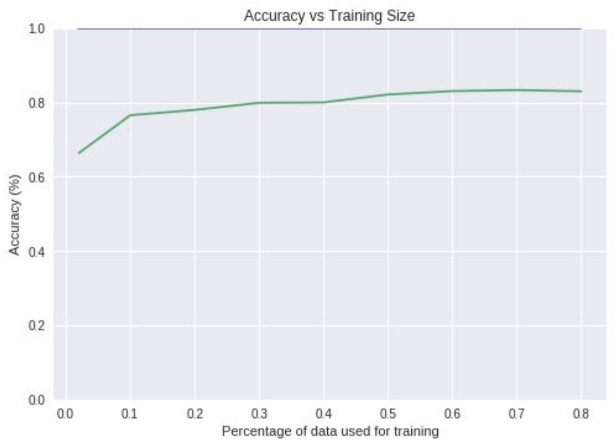
The visualizations indicate that boosting needs to be run many more iterations to create a “strong” learner. However, the curves also indicate that progress will be slow: test accuracy is beginning to level off with iterations, and has long plateaued with training size. To make effective use of boosting over CIFAR-10, a more effective weak learner is necessary: an SVM might do the trick, however evaluation would be painfully slow and prevents testing of said theory.

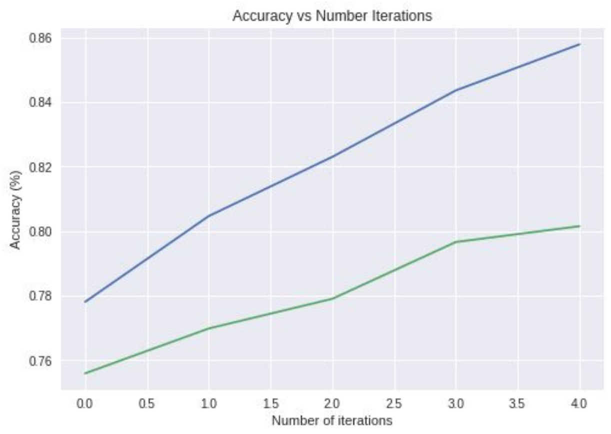
IMDB (Test/Total = .2, Val/Non-Test = .2)

Parameters				**Results**				
num estimators	learning rate		Learner	Wall Clock (min)	Train Acc	Val Acc	Tacc-std	Vacc-std
5	1		Green Row	16.03	0.8690	0.8020	0.0014	0.00257
10	2			24.25	0.6490	0.5790	0.0257	0.0258
5	1		Depth 1	0.57	0.6990	0.6980	0.0041	0.00501
5	0.5			0.56	0.7010	0.6990	0.001	0.00389
5	2			0.57	0.6190	0.6150	0.00146	0.00147
30	1		Green Row	40.02	1.0000	0.8370	0.0011	0.0013

On the IMDB dataset the process of fitting decision trees was much faster (a running theme). This allowed for many more iterations of boosting. Theoretically, this would create a much more accurate boosted learner. However as with CIFAR-10, only a modest increase in accuracy was obtained. The Green Row estimator (refer to previous section) achieves a validation accuracy of 76%. After 30 rounds of boosting that accuracy is only bumped to 83%.

Using the same fractions as above, learning curves are given below





In this case, however, the weak learner is too strong rather than too weak. The boosted learner achieves perfect training accuracy across all fractions of the training set and, as a function of iterations, quickly fits the training data perfectly. As a perfect fit is neared, the AdaBoost algorithm disturbs the training probability distribution less. Therefore progressively less information is extracted from the training set, even when the learner needs more to generalize to the testing set. This situation illustrates the boosting flavor of overfitting. The primary means to combat this overfitting would be to decrease the relative strength of the weak learner, either by (a) increasing the size of the training set to make it harder to memorize or (b) changing parameters to directly reduce the weak learner’s capacity.

K Nearest Neighbors

As with SVMs, I had issues of slow evaluations using KNNs. This is due to the fact that classifying a single point of data requires traversing every point of the training set and calculating a distance, a linear operation in data dimensionality. for each. Even “training” on 10,000 examples and testing on 10,000 thus requires ~(10,000)^2 * 1000 = 10^9 operations, which is infeasible. Therefore, with KNNs, I *only use 15% of each dataset: all fractions are relative to that*

CIFAR-10 (Test/Total = .2, Val/Non-Test = .2)

num neighbors	Minkowski-P	weighting	Wall Clock (min)	Train Acc	Val Acc
1	2	uniform	0.67	1.0000	0.2276
5	2		3.03	0.4234	0.2220
3	1		3.02	0.5067	0.2530
3	1	distance	3.08	1.0000	0.2692
7	2		60.33	1.0000	0.2453
10	1		3.10	1.0000	0.3098
8	5		97.50	0.2819	0.1625

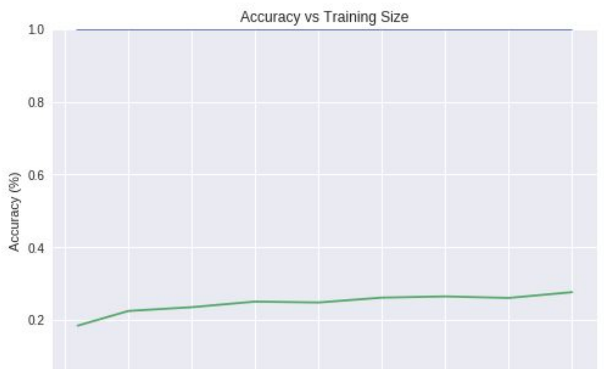
After testing several configurations of KNN, some trends were apparent. Weighting neighbors by the inverse distances (“distance” weighting) did seem to improve performance over uniform weighting, and increasing the neighborhood size also seemed to increase performance significantly: the Green Row improves on the nearest competitor by 5%, which is due only to an increase in neighborhood size. However, further increasing the neighborhood to 20 actually decreased validation accuracy to 28%, indicating that the effect of neighborhood on accuracy eventually diminishes

All distances were computed with the Minkowski p-norm, which is equivalent to Euclidean distance when p=2, Manhattan distance when p=1, etc. Though simple, this is not an ideal metric for comparison of images in pixel space. It exploits none of the structure of an image, and even small modifications such as shifts in position and hue may radically alter image distance. Therefore, it is surprising that minkowski clustering is able to perform better than random at all.

Taking a look at the per-class accuracies given by Green Row, we see they vary highly

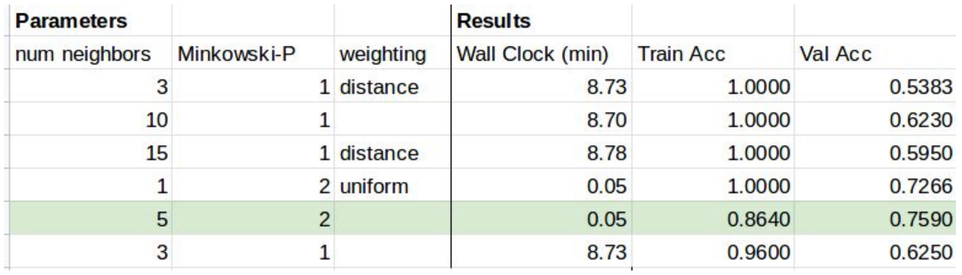
Airplane	Car	Bird	Cat	Deer	Dog	Frog	Ship	Truck
0.415	0.123	0.354	0.156	0.379	0.153	0.22	0.559	0.242

The overall accuracy is largely driven by high accuracies on the “Airplane” and “Ship” classes. Representatives of both of these classes are include large portions of same-color background as opposed to the others, and therefore Minkowski distances within these classes are lower [2]. Thus, KNN is learning what it is able under the limitations of the metric. It is very possible that further accuracy could be obtained with a more specialized (and presumably complicated!) metric.



On the test set, Green Row achieved an accuracy of 28%. The nearly flat relationship between accuracy and Data-Fraction is surprising. One might expect performance to increase given more data; however this is probably due to Minkowski metric clustering based on background color, of which there are just a few general varieties.

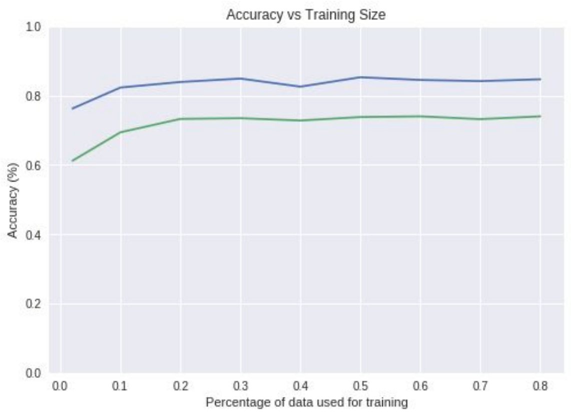
IMDB *(Test/Total = .2, Val/Non-Test = .2)*



Parameters			**Results**		
num neighbors	Minkowski-P	weighting	Wall Clock (min)	Train Acc	Val Acc
3	1	distance	8.73	1.0000	0.5383
10	1		8.70	1.0000	0.6230
15	1	distance	8.78	1.0000	0.5950
1	2	uniform	0.05	1.0000	0.7266
5	2		0.05	0.8640	0.7590
3	1		8.73	0.9600	0.6250

Upon this dataset, certain parameters seem to have more of a definitely effect. The uniform distance weighting would seem to have a rather strong advantage in accuracy over the inverse-distance weighting: for a 3-neigbour, 1-norm classifier, it gives a 10% increase in performance. However, as the IMDB data is also represented as a count vector, the performance of KNN on this dataset is also limited by the semantics of the Minkowski metric. Even applying mean-centering and variance standardization to the count vectors had negligible effect upon the accuracy of the model. The model performs significantly better than random, but given the simplistic representation of the data, it is unlikely that more performance can be gained without the use of a specialized metric.

The green row of hyperparameters was further tested via 5-fold cross validation, with the mean train/val accuracies being .845/.737 with standard deviations of .003/.009. On the test data, this model achieved an accuracy of 74%. Its data-fraction learning curve, and my analysis for it, are similar to that which I give on the CIFAR-10 dataset.



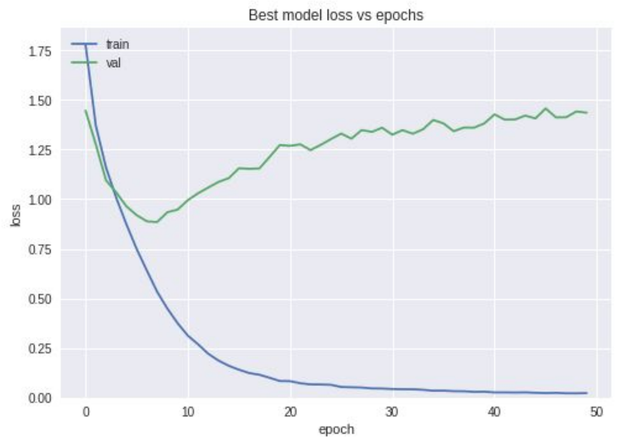
Neural Networks

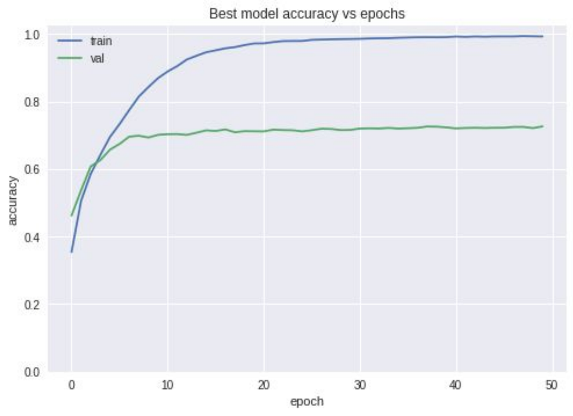
CIFAR-10 *(Test/Total = .4, Val/Non-Test = .2)*

Epochs	Learning Rate	momentum	Wall Clock (min)	Val accuracy
10	0.01	0.9	1.61	0.6527
20			3.18	0.7055
50			7.92	0.759
100			15.99	0.7255
10	0.01	0	1.6	0.4343
20		0.1	3.07	0.5545
50		0.3	8	0.686
100		0.5	16	0.7055
10	0.02	0.9	1.6	0.683
20			3.13	0.702
50			1.46	0.711
100			15.99	0.7056
10	0.05		1.5	0.649
20			3.1	0.692
50			2.3	0.099
100			1.7	0.6659

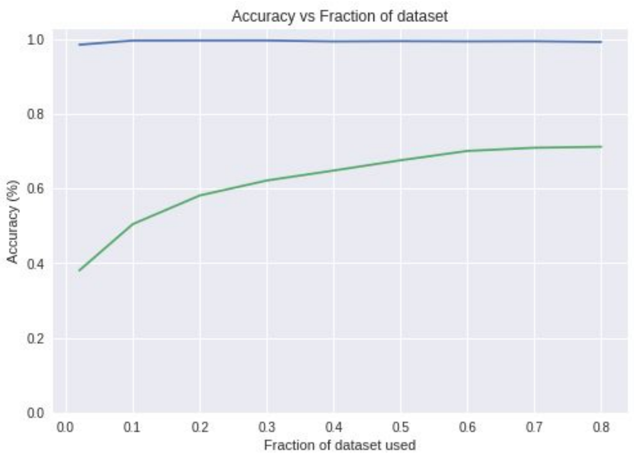
For neural networks, I performed a grid search on hyperparameters. Each network was able to memorize/achieve perfect accuracy on its training set, and thus that statistic is omitted from the table. Though I tested a variety of configurations, in the end it was the default values which achieved the optimal validation accuracy. The network of 3 groups of convolutional /dropout / pooling layers, with a fully connected layer of 512 neurons at the very end.

Green Row achieved a test accuracy of 66.72%, making it by far the most effective learner on the CIFAR-10. However, as with neural networks in general, it is important to terminate the model before it begins to overfit on the training data. Observe the graphs below





Although the validation accuracy quickly stabilizes epoch 5, from about the 7th epoch training loss and accuracy are optimized at the expense of generalization on the validation set. Although it seems that the local optima of both sets coincides here, given other dataset it is likely that too many epochs would cause generalization collapse. The "catastrophic forgetting" problem even appears here. The table shows that with a learning rate of 5%, the model eventually collapses to a random guesser until it relearns. Thus, even this toy problem shows pitfalls of neural networks. The Data-Fraction curve (fractions are 2, 10,25,50, 75%), demonstrates a smooth increase in performance given more data. However, the large gap between training / validation accuracies and the plateau of the latter indicate the



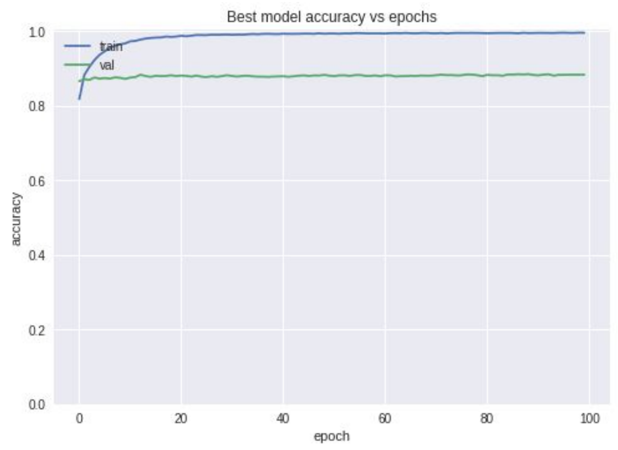
model is overfitting. Therefore, a simpler architecture might provide better generalization.
IMDB (Test/Total = .4, Val/Non-Test = .2)

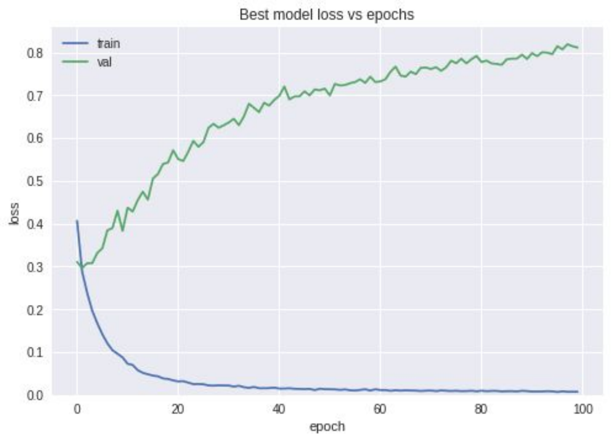
Epochs	Learning Rate	momentum	Wall Clock (min)	Val accuracy
10	0.01	0.9	0.805	0.879
20			1.59	0.878
50			3.99	0.878
100			8.04	0.879
10	0.01	0	0.801	0.8721
20		0.1	4.05	0.8823
50		0.3	8.12	0.8775
100		0.5	0.83	0.8831
10	0.02	0.9	1.62	0.8773
20			4.14	0.878
50			8.25	0.873
100			0.88	0.8751
10	0.05		1.66	0.8795
20			0.99	0.8796
50			4.12	0.8745
100			0.996	0.8837

On the IMDB dataset, I testing the same configurations which I did with CIFAR-10. The network architecture here was simple: 3 stacks of dense / dropout layers, followed by a final dense layer: each layer being 50 nodes wide. An interesting improvement would be to use a convolutional architecture which would be able to exploit the full sequential nature of language. However, I refrained from such an implementation to facilitate comparison with other learners, which all use count-vectors.

We again see strong clustering of validation accuracies. Moreover, the results are *highly* clustered: none of the validation accuracies differ by more than two percent. Frankly, I don't have any explanation for why the data exhibits such strong clustering, even with a variety of learning rates and epochs. However, the results are at least good: 88% validation accuracy is the highest yet.

The Green Row achieved a test accuracy of 87%. Graphs of loss and accuracy / epochs below:





Somewhat incredibly, validation accuracy is maxed out after a single epoch (pass over the data): in fact, it's surprising that validation accuracy remained as high and steady as it does given that the validation loss immediately begins to increase over time. Again, the network is overfitting on the training data, and thus the architecture is much too expressive. In fact, neural networks fail to appreciably outperform *Linear SVMs* on this dataset by an large margin: thus the data is probably linearly separable (or very close to it) and so even a single-layer network would probably suffice

Conclusion

Completing this project was a struggle, but it was a good learning experience. The first conclusion I drew was that CIFAR-10, while an interesting problem, was probably too hard for most of the learners presented here. The complexity of the image classification task made the problem difficult for learners such as decision trees and KNN, and I suspect (though, as with many theories in ML, cannot directly verify) that these learners were mostly failing to represent the true semantics of the classes. A dataset with more easily separable dimensions and even just a few thousand examples might have enabled better comparisons between models, and I do at least feel that the IMDB dataset fell into this category and admitted easier model analysis.

There were some surprises: SVMs performed much better than expected on both datasets. In fact, I'd say that relative to their representational capacity, they far outperformed neural networks in both sets. The abject failure of boosting in both cases was a huge surprise. In fact, it makes me doubt whether decision trees are really suited to either dataset though to be fair, continuous-input problems are not exactly where decision trees shine and so I can forgive their poor performance. Finally, the fact that grayscaling the images degrades performance demonstrates that even commonly applied techniques for "simplifying" inputs should be rigorously evaluated before assuming that they help for any given problem.

Citations/References

1. Abdiansah, Time Complexity Analysis of Support Vector Machines
2. CS 231n, Pros and Cons of Nearest Neighbor