Supervised Learning

Hermann Hans
Department of Computer Science, Georgia Institute of Technology

October 4, 2016

Abstract

In this document we'll take a closer look at some supervised learning techniques. Five different learning algorithms are explored: decision trees, boosting, k-nearest neighbors, neural network and support vector machines. Two different datasets were used throughout the analysis to compare and contrast the performance and results of those algorithms.

1 Datasets

1.1 Image Segmentation

The image segmentation dataset is a uniformly distributed classification dataset consisting of 19 continuous attributes, 7 classes and 2310 instances. The goal is to identify various parts of an image as one of the 7 classes based on 3x3 regions of pixels within the image.

While the dataset is very basic as far as image recognition / classification goes, the idea behind it has many interesting real-world applications. Being able to identify items, persons, etc is already being used in a broad range of fields from social networks, to industrial machines and more recently for autonomous vehicles.

In the context of this assignment the dataset should serve as an interesting contrast to the wine quality dataset. The wine quality dataset isnt uniformly distributed and the classes are more subjective than the factual classes in the image segmentation dataset.

1.2 Wine Quality

The wine quality dataset was split into red and white wines, consisting of 1599 and 4898 instances respectively. The datasets contain the same 11 input attributes and can be classified into 11 different qualities (0 to 10). The quality is the median score of at least 3 evaluations made by wine experts.

The dataset is not balanced (unlike the image segmentation set) and thus contains many normal wines, and few poor or exceptional wines. This should make it interesting to see which classification algorithms best can cope with the limited training data of those edge cases.

Figure 1: Unbalanced class distribution of the wine quality dataset

For my use case here I combined the dataset into a single one, and added another input attribute (wine type = red, white). The additional attribute was necessary upon combining the red and white datasets, as the quality is likely different for the continuous input parameters for the different wine types.

2 Decision Trees

The biggest point of interest in decision trees with regard to the datasets used was the size of the trees that were produced. Even with less attributes, the wine dataset produced significantly larger trees than the image segmentation set. This of course leads to a big discrepancy in the effect that pruning has on the two data sets.

Training data	Test data	CF	Error % (train)	Error % (CV)	Error % (test)	Tree Size	Leaves	Train (ms)	CV (ms)	Test (ms)
693	1617	0.1	1.5873	4.8341	5.7609	23	45	0	282	0
693	1617	0.25	1.2987	4.4012	5.5278	25	49	0	156	0
693	1617	0.5	1.2987	3.8961	5.2947	25	49	0	93	0
693	1617	unpruned	1.2987	4.2569	5.5278	26	51	0	109	0

Figure 2: Decision Tree - Image Segmentation

For the Image segmentation set, the small tree size is already close to optimal and pruning only has an insignificant effect on the size as well as the accuracy of the model.

Training data	Test data	CF	Error % (train)	Error % (CV)	Error % (test)	Tree Size	Leaves	Train (ms)	CV (ms)	Test (ms)
1945	4552	0.1	19.6401	34.3959	42.6676	204	407	0	653	16
1945	4552	0.25	13.6761	32.3907	42.0161	302	603	0	446	16
1945	4552	0.5	13.1620	31.5167	41.5778	315	629	0	454	0
1945	4552	unpruned	12.5964	31.2853	41.4001	345	689	0	454	15

Figure 3: Decision Tree - Wine Quality

On the other hand, for the wine dataset we can clearly see the difference between an unpruned (345 nodes) vs an aggressively pruned (204 nodes!) tree. The accuracy of the more aggressively pruned tree also declines slightly, which may hint at that the unpruned or less aggressively pruned trees are overfitting the model on the training data.

From the learning curves we can see that decision trees, even without boosting work pretty well for the image segmentation data set, getting down to an error percentage just above 5

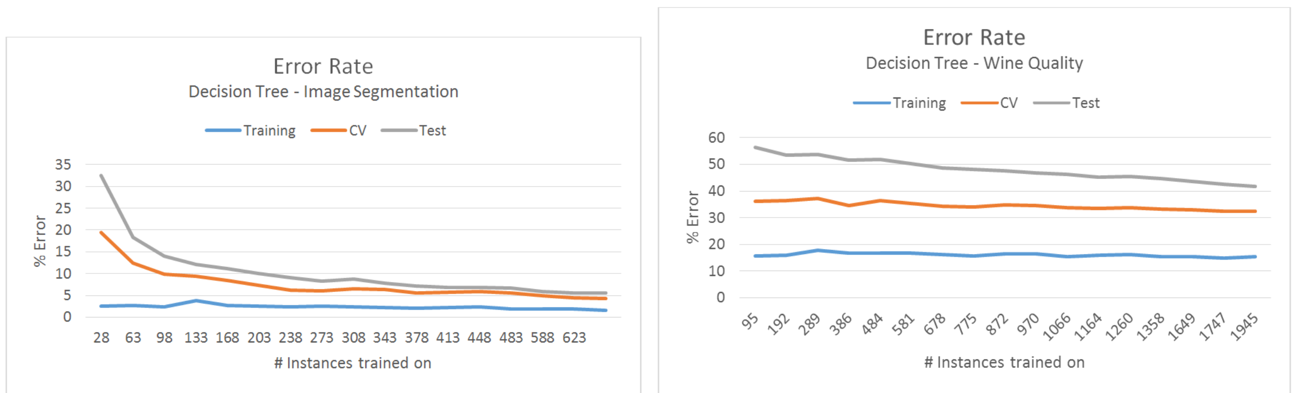

Figure 4: Error rate Image Segmentation [LEFT] and Wine Quality [RIGHT]

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	2	4	2	0	0	0
4	0	0	0	2	23	22	17	0	0	0	0
5	0	0	0	0	20	430	154	33	4	0	0
6	0	0	0	2	16	183	546	92	11	0	0
7	0	0	0	0	5	26	129	150	13	0	0
8	0	0	0	0	1	4	17	12	23	0	0
9	0	0	0	0	0	1	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0

Figure 5: Confusion Matrix on cross-validation of Wine Quality dataset

3 Boosting

I used AdaBoost on the J48 Decision Tree algorithm which was already analyzed above. Like expected, boosting brought improvements on both datasets, most noticeably on the wine dataset where an improvement of almost 10% can be seen on the largest training set.

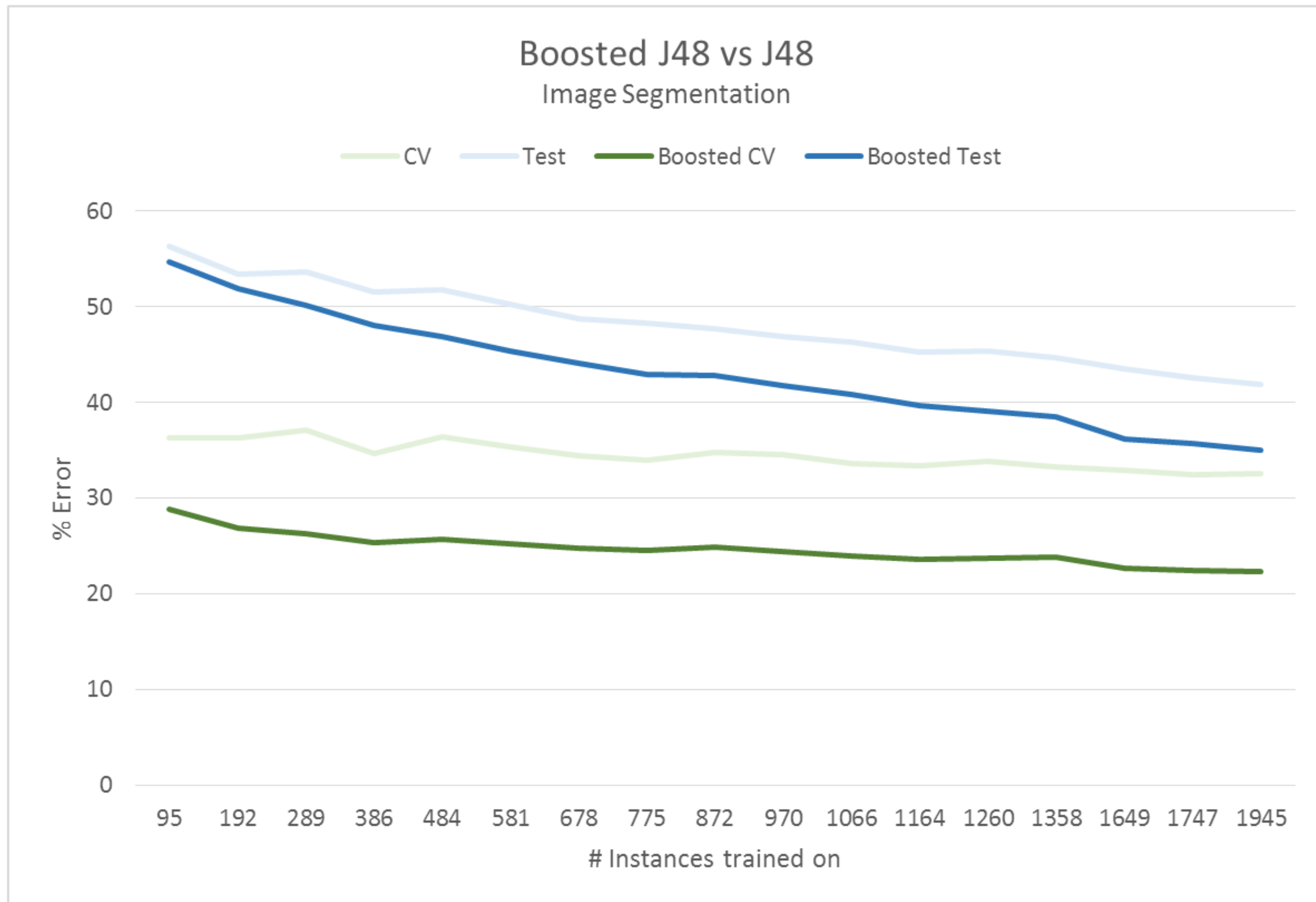

Figure 6: Boosted vs non-boosted J48 Decision Tree

Since this was pretty much expected behavior, I decided to take a closer look at the number of iterations parameter (call it M) for boosting. The number of iterations that we loop over our algorithm and update the weights based on previous results should continuously improve the accuracy of our algorithm (like shown in the lectures where it was also mentioned that boosting is very resistant to overfitting!).

For both datasets, slowly upping the value of M, the improvements are immediately visible even for small values of M. The images dataset for example showed that training error arrived at 0% with a value of 4, with the wine dataset doing the same with a value of 6. For reference, the unboosted J48 training error was well over 10% for all cases! Ok, so knowing how Boosting works this is probably not interesting, but the realization of how quickly boosting works to improve on a weak learner like J48 made me excited enough to mention it here as well. Furthermore, the CV and test error rates

also improve most quickly in the beginning and then trailing off with only marginal improvements relative to each other for larger values of M.

With the above analysis and the observation that training time is negatively impacted (a result of looping M times over our classifier that we are boosting) it makes sense to keep in mind that upping the value of M for larger values will only have marginal gains and may not be desirable when training time is important.

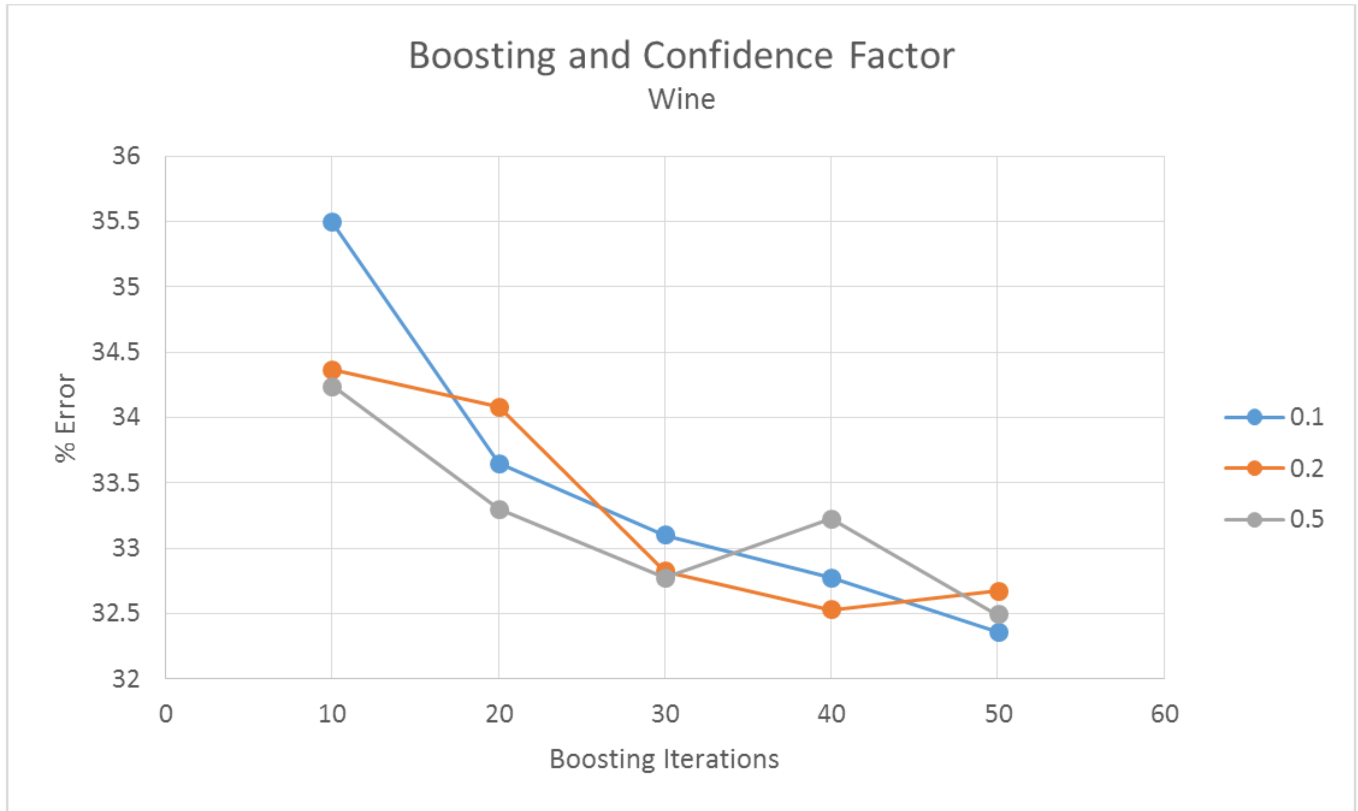

Figure 7: Boosting iterations with various confidence factors

4 K-Nearest Neighbors

For kNN, I ran some experiments with modifying the number of neighbors for each of the distance functions that were supplied by WEKA. In the graph below (its a bit of a mess, so bear with me please), the top 5 lines represent the CV error % for different values of k, while the bottom 5 lines that are bunched together represent the training error %.

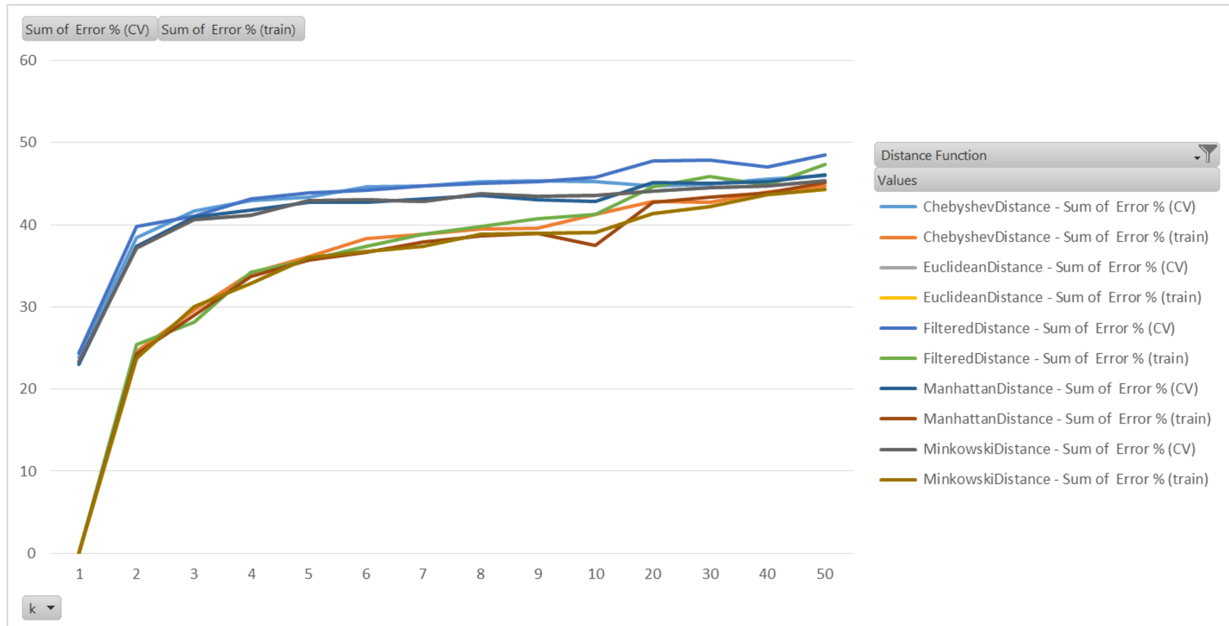

Figure 8: Error curves of different distance functions for kNN

Different distance functions dont have much effect on kNN in this case, as each the rates for both CV and training are fairly similar. Whats interesting is that for k=1 we get 0% training error in all cases, where k=2 immediately bumps the training % error up into the 20+% range. This suggests to me that for k=1 we probably overfitting the data as a single data point as neighbor is not enough information to create a model that will also fit well for our test data then.

Training da	Test da		Distance Function	Error % (trai	Error % (C\	Error % (te	Train (m	CV (m	Test (m
693	1617	1	ManhattanDistance	0	2.597402597	4.129204129	0	141	514
693	1617	3	ManhattanDistance	2.308802309	3.968253968	4.695304695	0	177	623
693	1617	1	EuclideanDistance	0	3.246753247	4.728604729	16	640	750
693	1617	1	MinkowskiDistance	0	3.246753247	4.728604729	0	140	313
693	1617	4	ManhattanDistance	3.03030303	4.834054834	5.760905761	0	261	553
693	1617	5	ManhattanDistance	2.886002886	4.978354978	5.827505828	0	245	610
693	1617	3	EuclideanDistance	3.751803752	5.339105339	5.894105894	0	204	687
693	1617	1	ChebyshevDistance	0	4.112554113	5.894105894	0	109	359

Figure 9: Distance Functions and various kS sorted by Error % on test set for Image Segmentation

On the images dataset, varying the distance function actually has a more significant effect on the kNN model. The CV curves below show that Manhattan distance along with Euclidean (Minkowski and Euclidean are on the same line here, due to order p=2). Greatly increasing the number of neighbors from 10 to 50 adversely affects our error rate for this data set.

Trainin	Test da	k	Distan	Error %	Error %	Error %	Train (	CV (ms	Test (m
1945	4552	1	Manhatta	0	23.05913	35.96304	0	656	1610
1945	4552	1	Euclidean	0	23.3162	36.12888	0	1750	2594
1945	4552	1	Minkowsl	0	23.3162	36.12888	0	542	1424
1945	4552	1	Chebyshe	0	23.88175	36.68562	0	775	2031
1945	4552	1	FilteredDi	0	24.37018	38.5809	0	9698	28528
1945	4552	2	Euclidean	23.7018	37.17224	44.06539	0	910	2114
1945	4552	2	Minkowsl	23.7018	37.17224	44.06539	0	735	1511
1945	4552	2	Manhatta	24.26735	37.4036	44.21938	0	734	1791
1945	4552	2	Chebyshe	24.57584	38.38046	45.33286	0	917	2036
1945	4552	10	Manhatta	37.48072	42.80206	45.42762	0	840	2156

Figure 10: Distance Functions and various kS sorted by Error % on test set for Wine Quality

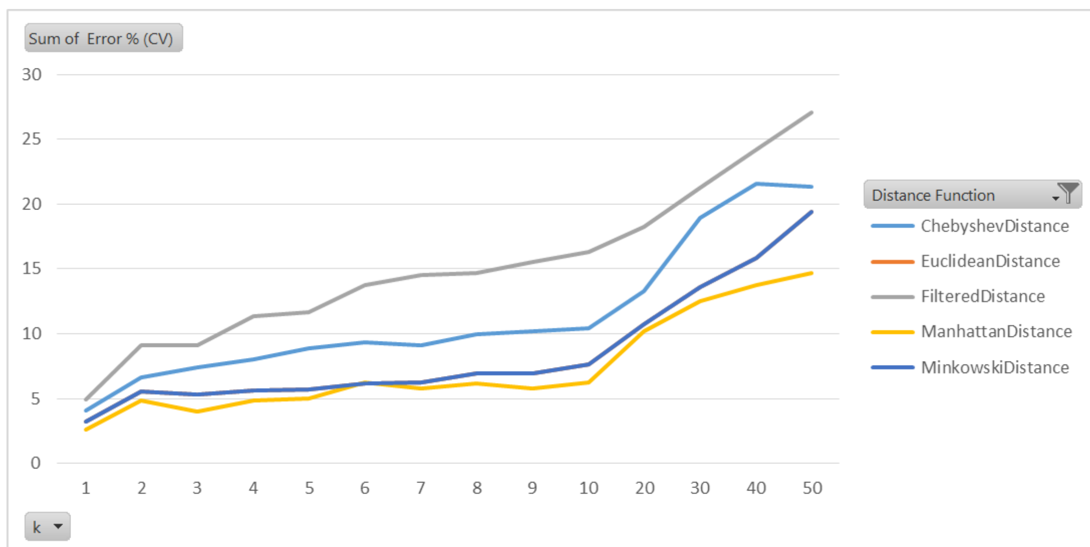

Figure 11

5 Neural networks

Both datasets learning curves seemed to converge fairly quickly to an error percentage where increasing the number of training instances only showed minor improvements on the CV and test curves. For both sets, just around 10% (give or take a bit) was usually enough to get to an area where this behavior could be seen.

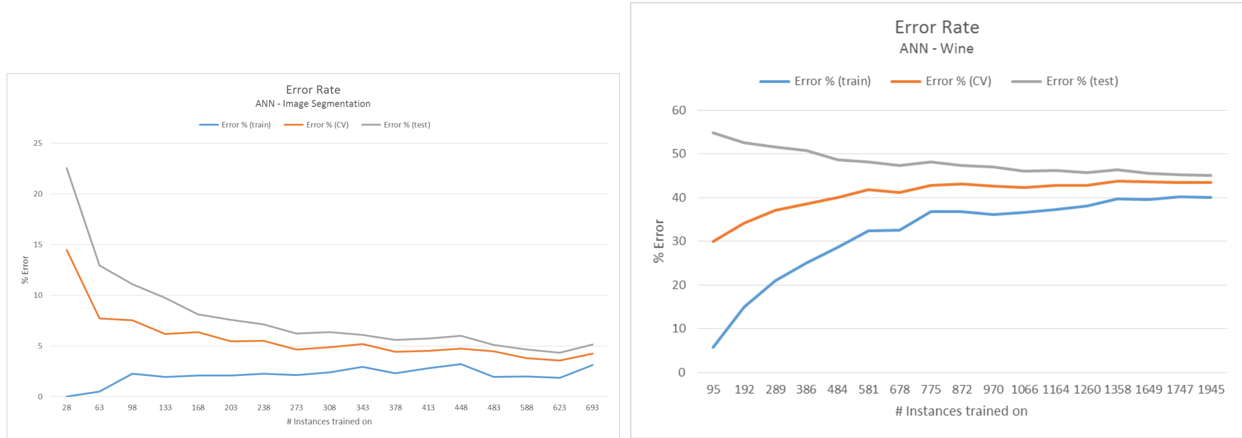

Figure 12

Modifying the learning rate and the training time showed further improvements on the accuracy for the image segmentation data set. Unfortunately this also drives the training time in a linear fashion as well as can be seen in Figure 13.

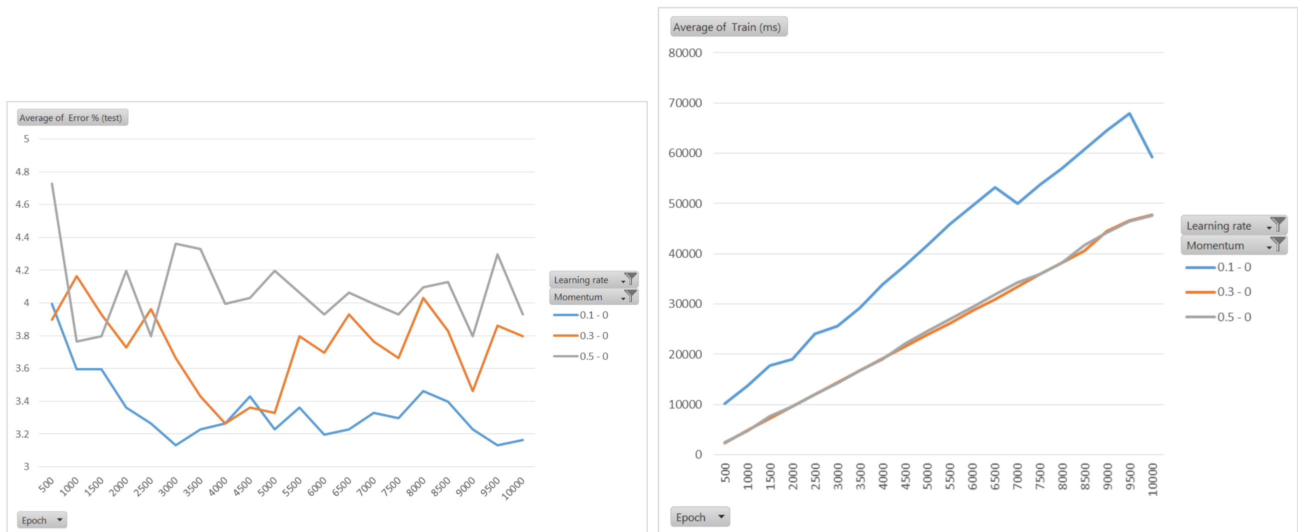

Figure 13

6 Support Vector Machines

I ran some trial and error experiments with all four supported kernels before deciding on Polynomial and Radial Basis Functions based on those results.

Training data	Test data	Kernel	Degree or Gamma	Error % (train)	Error % (CV)	Error % (test)	Train (ms)	CV (ms)	Test (ms)
1945	4552	Polynomial	1	46.4781491	47.14652956	47.23999052	84038	577585	828
1945	4552	Polynomial	2	43.0848329	46.14395887	47.07415304	236930	2228995	678
1945	4552	Polynomial	3	61.33676093	58.7403599	62.34304667	372910	3633374	510
1945	4552	Polynomial	4	61.18251928	64.55012853	64.23833215	399979	3484445	469
1945	4552	Polynomial	5	73.36760925	73.3933162	73.64368633	394683	3819087	372
1945	4552	RBF	0.01	43.03341902	47.30077121	50.34352049	663	6266	1146
1945	4552	RBF	0.1	14.91002571	31.8251928	41.53044302	1987	15980	1644
1945	4552	RBF	0.5	1.748071979	25.52699229	38.24923004	2509	21774	2193
1945	4552	RBF	1	0.462724936	25.42416452	38.33214878	2828	23467	2457
1945	4552	RBF	5	0	25.08997429	38.34399431	3031	25909	2613

Figure 14

For the wine data set, the polynomial kernel showed some of the worst performance of any algorithms I tried. Increasing the degree led to even greater error margins, but I also suspect that I may have had an issue with the algorithm itself since I kept getting a warning about reaching the maximum number of iterations. My best guess is that this was happening due to the unbalanced data in the wine data set. I didnt get around to it, but it wouldve been interesting to use the w parameter to assign weights to the different classes. Also, the runtime of the polynomial kernel was enormous for this dataset.

The RBF kernel performed much better and was almost on par with a boosted decision tree. Given my limited parameter tuning of the RBF kernel, its likely that a SVM with an RBF kernel may give us the best classification model for the wine quality dataset.

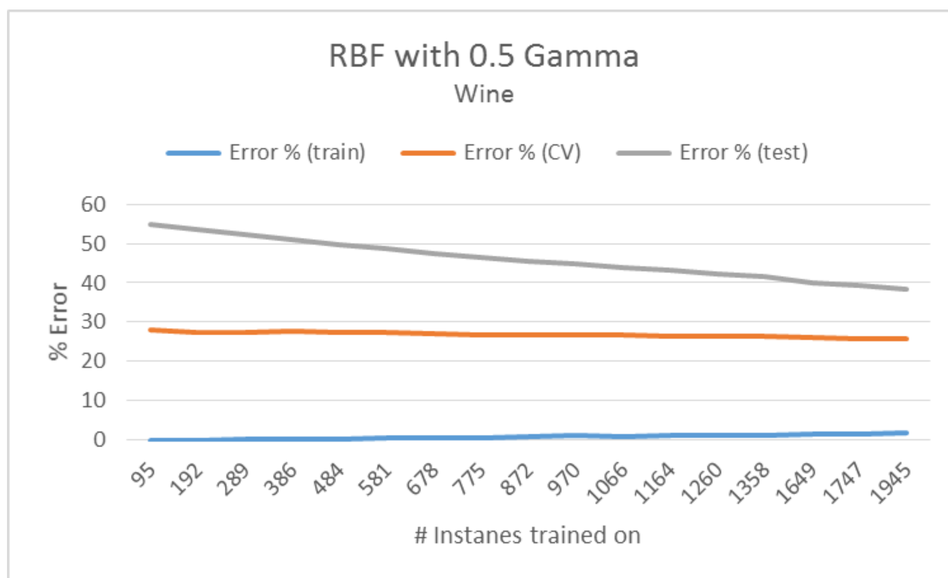

Figure 15

My reasoning behind why RBF performs so much better with the wine data set than a polynomial

or a linear kernel is that the data there is not linearly separable at all. Its more likely that the data is clustered in a way where a moving data into a higher dimension (like mentioned in the lectures) helps out to separate the data into clusters which give the model a higher accuracy on the test set.

What was interesting as well, was that the different kernel types had the inverse effects on the image segmentation dataset. The 2nd order polynomial kernel did best while the RBF kernel only managed 21% at best while varying the gamma.

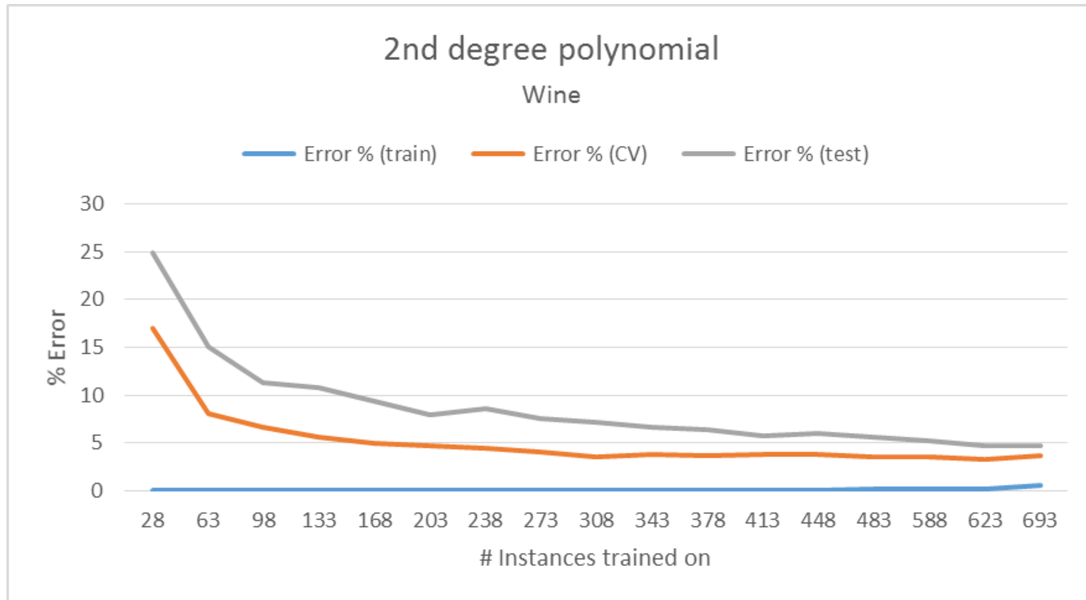

Figure 16: Wrong label! This is image segmentation!

Training data	Test data	Kernel	Degree or Gamma	Error % (train)	Error % (CV)	Error % (test)	Train (ms)	CV (ms)	Test (ms)
693	1617	Polynomial	1	3.751803752	4.256854257	4.728604729	907	2766	15
693	1617	Polynomial	2	0.288600289	3.463203463	4.495504496	22791	125984	78
693	1617	Polynomial	3	0	3.463203463	5.028305028	3682	25040	100
693	1617	Polynomial	4	0	3.391053391	5.294705295	2777	13459	15
693	1617	Polynomial	5	0	3.67965368	5.661005661	2391	13804	16
693	1617	RBF	0.01	0.144300144	13.41991342	21.07892108	328	2667	313
693	1617	RBF	0.1	0	35.71428571	51.71495171	453	4005	563
693	1617	RBF	0.5	0	39.8989899	60.90576091	453	4141	657
693	1617	RBF	1	0	40.83694084	61.87146187	468	4081	719
693	1617	RBF	5	0	40.98124098	62.17116217	484	4448	719

Figure 17: Image segmentation - different kernels and values for exponent or gamma