Supervised Learning

Christian Heaney

Datasets

The first dataset I chose was the default payments of credit card clients in Taiwan from 2005. It contains whether a client defaulted on their credit card payment as well as attributes such as demographic data, their balance limit, their payment history among others. I was eager to work with this dataset because I've noticed that banks and financial institutions have been early adopters of machine learning technology. For sometime I've been curious about how financial institutions handle problems that appear difficult to solve with rule systems e.g. fraud detection and the issuing of credit. I assumed that they solved these with machine learning methods and I see this project as an opportunity to simulate how financial industries tackle these problems in industry.

The second dataset I chose was the heart disease dataset. Besides the heart disease diagnoses, the data contains demographic data and other bits of medical information such as patients cholesterol and chest pain type. Since I've become interesting in artificial intelligence, I've been interested to see machine learning will improve the state of medicine and human health. The use of medical data to diagnose patients appears to be an incredibly valuable use case for machine learning and I found it exciting to be simulating how this might be implemented. Similar to the credit card example, I found the heart diseases dataset to be related to a practical field of artificial intelligence that I am interested in.

Methodology

I utilized the python library scikit-learn to conduct my experiments. Scikit-learn has a module called that GridSearchCV that accepts a classifier and a series of parameters and then finds what set of parameters produces the best learner. I utilized this method module to test out different options for pruning, kernel functions and other ways one can customize the different classifiers. I found github user "jbzhang1986" had already implemented a nice CLI wrapper around the scikit-learner GridSearchCV. Thus I "stole" that code and tweaked it for the use cases I was interested in exploring!

Decision Tree

For decision trees, I tried parameterizing scikit-learn's max depth and min sample leafs values. The max depth sets a limit on how deep the tree's nodes can go while the min sample leafs value dictates the minimum number of examples that much exist before the algorithm will consider making a leaf node into a parent node and forking it into child nodes. In both cases, these can be considered "pruning" in that they prevent the tree from extending as deeply as possible. The benefit to this approach is that the pruning makes the learner more resistant to overfitting.

class weight	criterion	max depth	min sample leafs	mean_test_score	std_test_score	rank_test_score	mean_train_score	std_train_score
balanced	entropy		5	0.845792541	0.104480241	1	0.889230264	0.010652295
balanced	entropy	200	5	0.832156177	0.100184375	2	0.88999055	0.010875976
balanced	entropy	500	5	0.832156177	0.100184375	2	0.88999055	0.010875976
balanced	entropy	100	5	0.830034965	0.097098122	4	0.88999055	0.010875976
balanced	entropy	100	4	0.806509324	0.097553841	5	0.899596436	0.00850966
balanced	entropy	200	4	0.80247669	0.107437256	6	0.899596436	0.00850966
balanced	entropy		4	0.793385781	0.099673979	7	0.89949581	0.008422003
balanced	entropy	500	4	0.79287296	0.093839486	8	0.89949581	0.008422003
balanced	entropy	500	2	0.791092657	0.070299285	9	0.96800334	0.006781726
balanced	entropy		2	0.784842657	0.08116933	10	0.968892813	0.007058554
balanced	entropy	100	2	0.783868631	0.076746861	11	0.968892813	0.007058554
balanced	entropy	100	3	0.781423299	0.085872561	12	0.9315288	0.005201834
balanced	entropy	200	3	0.779799922	0.093344279	13	0.930401957	0.005214452
balanced	entropy	200	2	0.779323177	0.081182136	14	0.96800334	0.006781726
balanced	entropy	500	3	0.778252303	0.089277737	15	0.930851273	0.006145864
balanced	entropy		3	0.768268537	0.094304406	16	0.930859974	0.007203385
balanced	entropy		1	0.763282967	0.087190053	17	1	0
balanced	entropy	100	1	0.761141359	0.069380932	18	1	0
balanced	entropy	200	1	0.760442058	0.081092742	19	1	0
balanced	entropy	500	1	0.757844655	0.091011052	20	1	0

Figure 1 Table for Decision Tree permutations for the Heart Disease dataset

class_weight	criterion	max_depth	min_samples_leaf	mean_test_score	std_test_score	rank_test_score	mean_train_score	std_train_score
balanced	entropy	100	4	0.655194675	0.036694157	1	0.932866477	0.003792266
balanced	entropy		5	0.654861155	0.044207672	2	0.913662834	0.003738812
balanced	entropy		4	0.653644213	0.041149411	3	0.932729864	0.004136991
balanced	entropy	500	5	0.652435401	0.045402688	4	0.913062592	0.003051315
balanced	entropy	500	4	0.652397609	0.045494416	5	0.932996919	0.003620265
balanced	entropy	100	5	0.650407699	0.042576673	6	0.913011555	0.003510331
balanced	entropy	200	5	0.647576143	0.044576978	7	0.912511998	0.00333301
balanced	entropy	200	4	0.645935889	0.047908534	8	0.932572011	0.004116741
balanced	entropy		3	0.64408969	0.046083343	9	0.956845959	0.003447453
balanced	entropy	200	3	0.638866812	0.037981914	10	0.956139375	0.003349411
balanced	entropy	100	3	0.638753638	0.037637628	11	0.95660564	0.003266427
balanced	entropy	500	3	0.638335658	0.044891348	12	0.956430753	0.003185401
balanced	entropy		2	0.62768214	0.047810963	13	0.981198949	0.002260017
balanced	entropy		1	0.62596477	0.047214252	14	1	0
balanced	entropy	200	2	0.623154876	0.045349042	15	0.981435546	0.001677112
balanced	entropy	500	2	0.621025017	0.0383952	16	0.981196356	0.001876333
balanced	entropy	100	2	0.618203071	0.046324724	17	0.981399709	0.001882657
balanced	entropy	100	1	0.618184172	0.033769418	18	1	0
balanced	entropy	500	1	0.614366959	0.039526189	19	1	0
balanced	entropy	200	1	0.61308579	0.050323703	20	1	0

Figure 2 Table for Decision Tree permutations for the Credit dataset

For the Heart data set, the optimal parameters were no max depth and a min sample leaf value of 5. This resulted in a test score of .84 and a train score of .89. For the Credit Card data set, the optimal parameters were a max depth of 100 and a min sample leaf value of 4. This resulted in a test score of .65 and a train score of .93.

There's two interesting points to pick out from these tables. First, for both datasets, it appears the min sample leaf parameter exhibits a stronger influence on the test score. In both tables the low min sample leaf values have the worst performance while the larger values exhibit better performance. The second point is that enabling the pruning metrics has a non-trivial influence on the learner's test score and training score. In both cases, when the learner is not constrained by max depth of min sample leaf, it shows perfect performance on the training test. This comes at the expense of generality. As the pruning conditions become more aggressive, the learners perform worse on the training data but better on the test data (.10 and .05 better for Heart

Disease and Credit Card data respectively). This demonstrates how pruning makes Decision Tree's more resistant to overfitting.

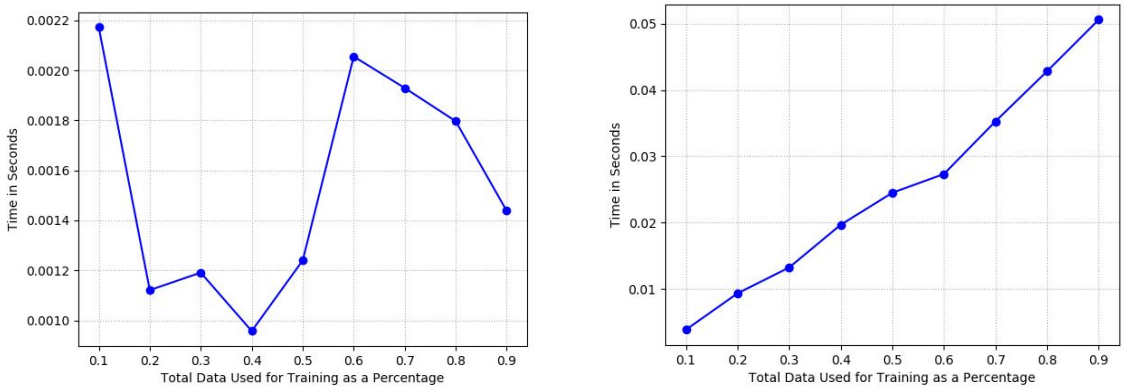

Figure 3 Training timing for Heart Disease (left) and Credit Card (right) datasets

With respect to the time it takes to train the data, the Credit Card dataset training time grows linearly as the amount of data increases. The Heart Disease dataset does not have a clear relationship between the amount of data and the training time. This might be for two reasons. First, the Heart Disease data set is only 1/10th the size of the Credit Card data set. Perhaps there isn't enough data to set a clear relationship between data size and training time because there is a small amount of Heart Disease data. The second potential explanation is that, since the Decision Tree performs better on the training set for the Heart Disease dataset (.85 versus .65), perhaps the Heart Disease dataset is more entropic and thus quickly trained regardless of the the size of the data.

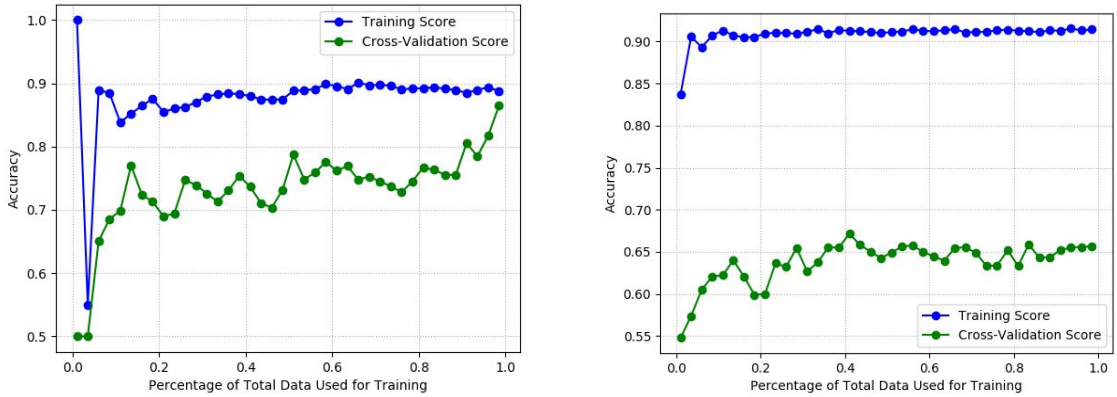

Figure 4 Learning curve for Heart Disease (left) and Credit Card (right) datasets

With respect to the learning curve, the Heart Disease dataset sees the cross-validation score converge more closely to the training score. As mentioned above, this might suggest that the Heart Disease dataset is more entropic, meaning that there are attributes that offer the classifier significant information gain.

Neural Networks

For Neural Networks, I parameterized the activation function and the size of the neural networks' hidden layers. For the threshold function I tried the logistic and relu options. For the hidden layers, I tried having one hidden layer of size 32, 64 and 128 and three hidden layers of sizes [32, 64, 32] and [64, 128, 64].

activation	hidden_layer_sizes	mean_test_score	std_test_score	rank_test_score	mean_train_score	std_train_score
relu	64	0.794427656	0.105217972	1	0.763840959	0.072073185
relu	(64, 128, 64)	0.714442641	0.142396049	2	0.76491987	0.042933683
relu	128	0.71335928	0.144749333	3	0.719196193	0.095198
relu	(32, 64, 32)	0.702771257	0.099896982	4	0.741940554	0.053402984
logistic	64	0.68961039	0.175164971	5	0.653006361	0.121459938
logistic	128	0.653184316	0.139132828	6	0.640977881	0.112453274
relu	32	0.620030803	0.115147165	7	0.656728036	0.068129664
logistic	32	0.589186508	0.10623881	8	0.605555685	0.096585216
logistic	(32, 64, 32)	0.587985348	0.108720471	9	0.571086044	0.072452286
logistic	(64, 128, 64)	0.5	1.23E-16	10	0.5	8.95E-17

Figure 5 Table for Neural Network permutations for the Heart Disease dataset

activation	hidden_layer_sizes	mean_test_score	std_test_score	rank_test_score	mean_train_score	std_train_score
relu	(64, 128, 64)	0.638349382	0.033040787	1	0.659807632	0.025546908
relu	64	0.620235233	0.034116084	2	0.627820092	0.020572793
relu	32	0.617621231	0.034577813	3	0.619818793	0.015797279
relu	128	0.614192389	0.032931628	4	0.623973084	0.027994355
relu	(32, 64, 32)	0.600481272	0.039420061	5	0.617507423	0.044754822
logistic	32	0.590574033	0.041140673	6	0.601232521	0.036315391
logistic	128	0.566792857	0.05163006	7	0.555924796	0.036898837
logistic	64	0.561871755	0.049644714	8	0.559978977	0.040809734
logistic	(32, 64, 32)	0.5	6.57E-17	9	0.5	1.11E-16
logistic	(64, 128, 64)	0.5	6.57E-17	9	0.5	1.11E-16

Figure 6 Table for Neural Network permutations for the Credit Card dataset

For the Heart Disease dataset, the optimal parameter for the activation/ function was the relu function and for the hidden layer sizes is was one hidden layer of size 64. It had a training score of .76 and a testing score of .79. It's interesting to note that the learner did better on the testing set than the training set, the sign of a general learning. For the Credit Card dataset, the optimal parameter for the activation function was also relu but for the hidden layer it was three layers of size 64, 128 and 64. It had a training score of .65 and a testing score of .64.

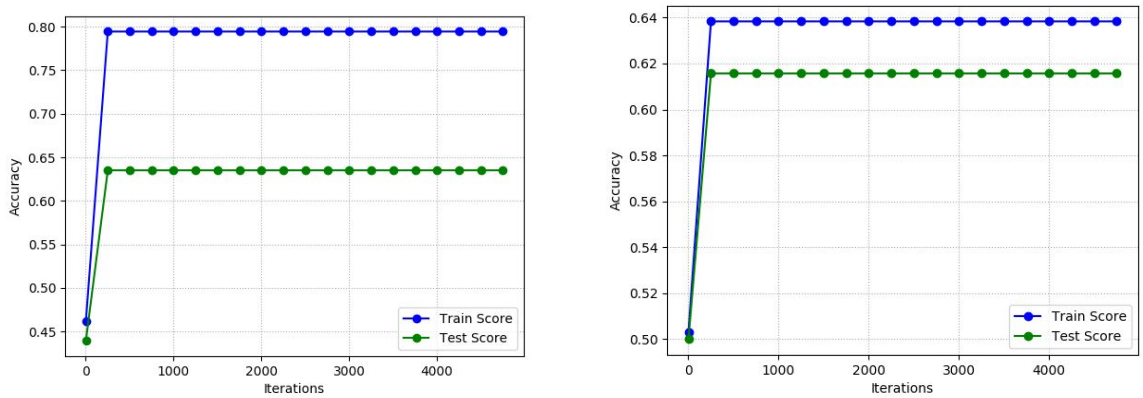

Figure 7 Iteration chart for Heart Disease (left) and Credit Card (right) datasets

With respect to the training iterations, in both cases the models stopped improving after a small number of iterations.

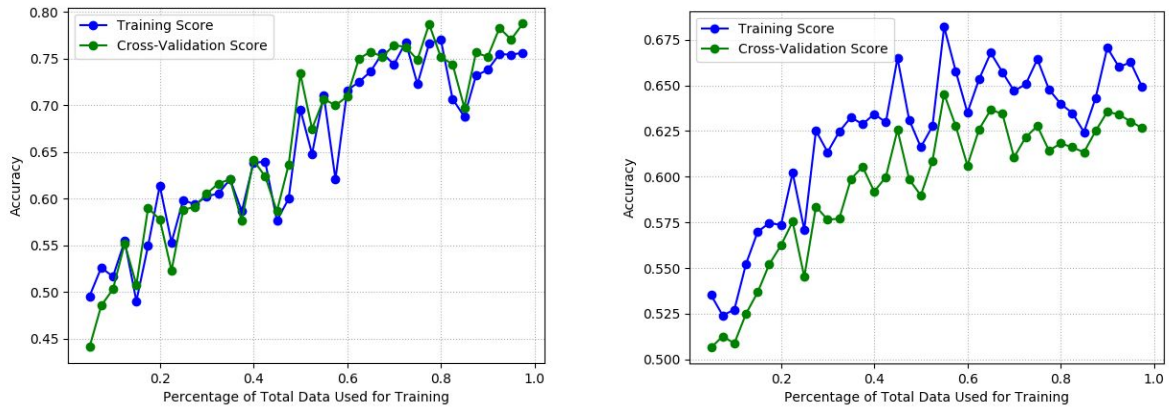

Figure 8 Learning curve for Heart Disease (left) and Credit Card (right) datasets

For the learning curves, both datasets continue to improve as they receive more data. This means that we can not rule out if the learners would have performed better if there was more data. This is especially interesting for the Heart Disease dataset as it has a pretty good performance of about ~.8 on the cross-validation testing even though it is a small dataset.

Boosting

For the boosting learner, I used a Decision Tree classifier with a min sample leaf value of 3. For the Heart Disease dataset, the learner performed best with 18 estimators and had a test score of .84. For the Credit Card dataset, it did best with 8 estimators and had a test score of .66

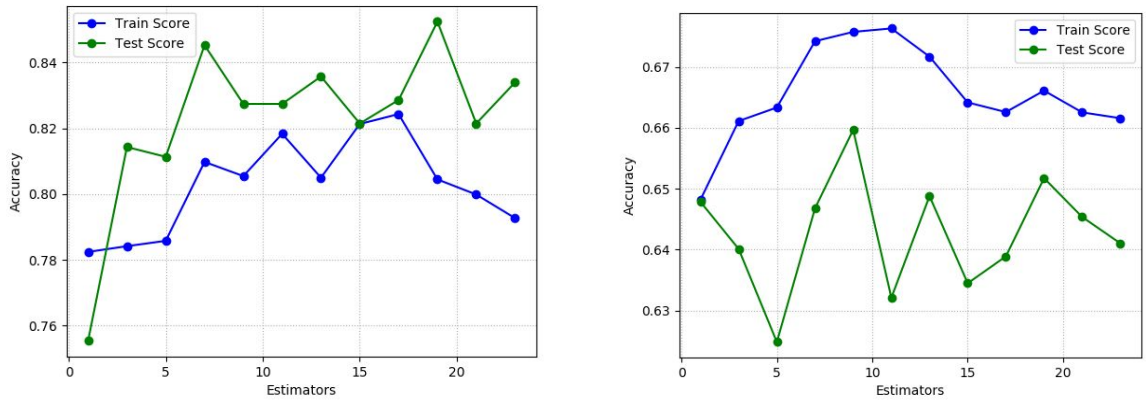

Figure 9 Estimator curve for Heart Disease (left) and Credit Card (right) datasets

It's interesting that the boosting algorithm does not do better than a decision tree learner. In fact, for the Heart Disease dataset, the single Decision Tree performed better than the booster.

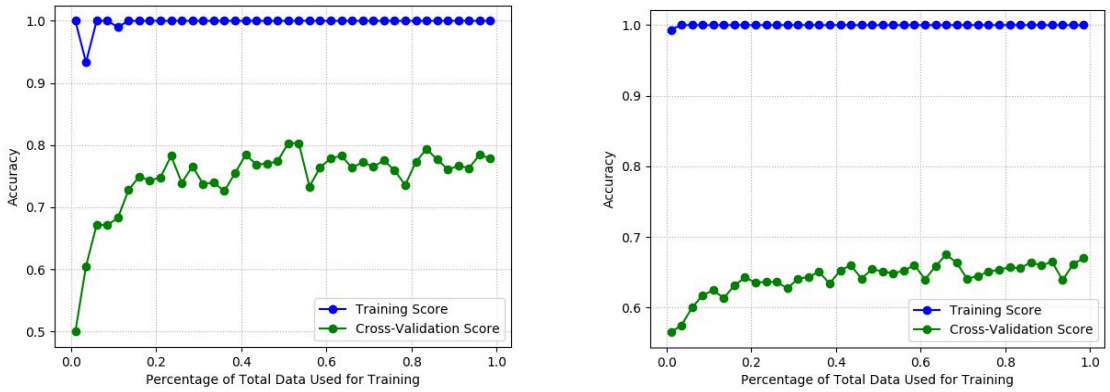

Figure 10 Learning curve for Heart Disease (left) and Credit Card (right) datasets

When we look at the learning curve, we see that the testing score does not improve much after 20% of the data is being used to train the classifier. This coupled with the fact that the boosting classifier did not outperform the simple Decision Tree classifier suggests that the Decision Tree classifier is approaching an upper bound on its performance, especially when compared with the neural net that continues to improve as it receives more data.

KNN

For the KNN classifier, I parameterized the number of neighbors for the comparison (K) and the distance metric used to compare the instances.

metric	neighbors	mean_test_score	std_test_score	rank_test_score	mean_train_score	std_train_score
manhattan	10	0.863544372	0.063352999	1	0.869034874	0.014188992
manhattan	28	0.844556277	0.085308366	2	0.845467929	0.011871007
euclidean	7	0.839736097	0.063917274	3	0.860388246	0.009179233
manhattan	4	0.837877123	0.083110588	4	0.889264793	0.015989031
manhattan	1	0.830907981	0.079686457	5	1	0
euclidean	4	0.829897325	0.094482385	6	0.881185149	0.016227489
manhattan	13	0.826617965	0.093946599	7	0.858633997	0.006644892
manhattan	25	0.826617965	0.102855629	8	0.83605393	0.012261189
euclidean	10	0.826230852	0.065999949	9	0.857457242	0.009225842
manhattan	7	0.821837468	0.059371101	10	0.864940674	0.007541516
manhattan	16	0.81729687	0.06603362	11	0.853530914	0.011642119
euclidean	1	0.815373515	0.081289643	12	1	0
euclidean	19	0.81530969	0.092918917	13	0.827280436	0.013107558
manhattan	19	0.814193723	0.088187862	14	0.831067083	0.015038028
euclidean	25	0.811738262	0.101309238	15	0.823028914	0.014790536
euclidean	22	0.811738262	0.09499417	15	0.830850441	0.012211153
euclidean	16	0.811738262	0.099889387	15	0.844911289	0.008802963
manhattan	22	0.808674242	0.101020492	18	0.836260148	0.011000016
euclidean	28	0.791197136	0.103568591	19	0.817226293	0.015349629
euclidean	13	0.791161061	0.098022603	20	0.839858349	0.012778807

Figure 11 Table for KNN permutations for the Heart Disease dataset

metric	n_neighbors	mean_test_score	std_test_score	rank_test_score	mean_train_score	std_train_score
manhattan	7	0.644541603	0.049778712	1	0.679997769	0.007894473
euclidean	7	0.630938234	0.038848581	2	0.682097908	0.006813344
manhattan	13	0.626896557	0.043055982	3	0.649247674	0.008896734
euclidean	13	0.619719052	0.035600184	4	0.645216219	0.006883762
manhattan	19	0.617541879	0.036454983	5	0.62855066	0.00521111
manhattan	10	0.614358912	0.041744017	6	0.634768561	0.007810008
manhattan	22	0.614188876	0.035524363	7	0.614916807	0.004963249
manhattan	25	0.614084873	0.034585691	8	0.618448037	0.004444686
manhattan	16	0.612157085	0.035944169	9	0.623603619	0.007310051
euclidean	16	0.610968076	0.032730377	10	0.621700342	0.005419896
euclidean	10	0.60961044	0.040306022	11	0.640334988	0.005662284
euclidean	19	0.608536748	0.034299581	12	0.623907604	0.005703979
euclidean	25	0.607655764	0.031689468	13	0.61546885	0.006162488
euclidean	22	0.607439227	0.028759502	14	0.61166263	0.005564303
manhattan	28	0.607408593	0.03419533	15	0.610050898	0.005958969
manhattan	4	0.607377991	0.034887003	16	0.661398288	0.007121331
euclidean	4	0.60554145	0.039996668	17	0.660914712	0.0086253
euclidean	28	0.602996346	0.02914625	18	0.608065137	0.004833798
euclidean	1	0.59930389	0.032301209	19	1	0
manhattan	1	0.588535819	0.042497074	20	1	0

Figure 12 Table for KNN permutations for the Credit Card dataset

For the Heart Disease dataset, the optimal distance metric was the manhattan distance and the optimal number of neighbors was 10. It's score on the test data and training data was .86. For the Credit Card dataset, the optimal distance metric was also manhattan distance but the optimal numbers of neighbors was 10. It's training performance on the test data was .64 and on the training data it was .68. An interesting thing to note about the KNN learner is that it was the most consistent across parameters. The Heart Disease dataset clustered around .8 and the Credit Card dataset clustered around .6.

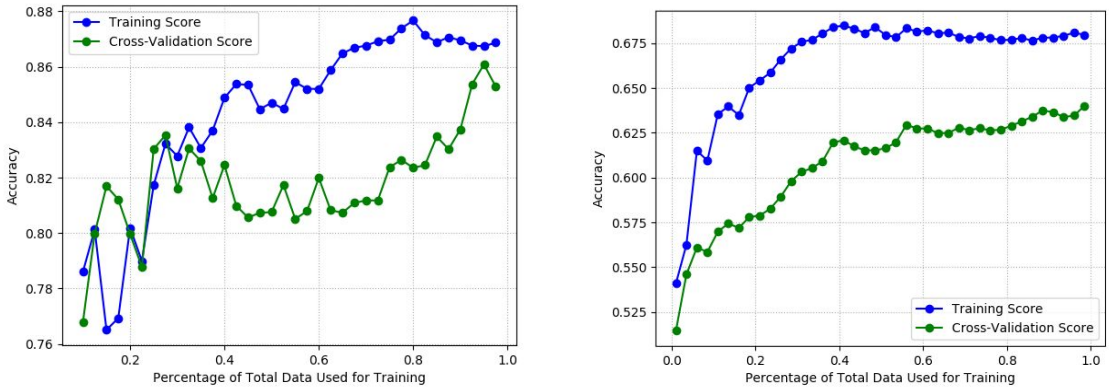

Figure 13 Learning curve for Heart Disease (left) and Credit Card (right) datasets

For the learning curve, the Heart Disease dataset exhibits behavior similar to its Neural Network pattern; as it gets more data it continues to improve. For the Credit card dataset, on the other hand, it appears to bump into an upper bound of about .65 for its testing error. This is consistent with the other learners and suggests that there is pattern with the Credit Card dataset.

SVM

For the SVM, I tested the linear, poly and rbf kernels. The rbf performed the best for the Heart Disease dataset and had a test score of .82. The rbf kernel was also the best kernel for the Credit Card data set; it had a performance of .65.

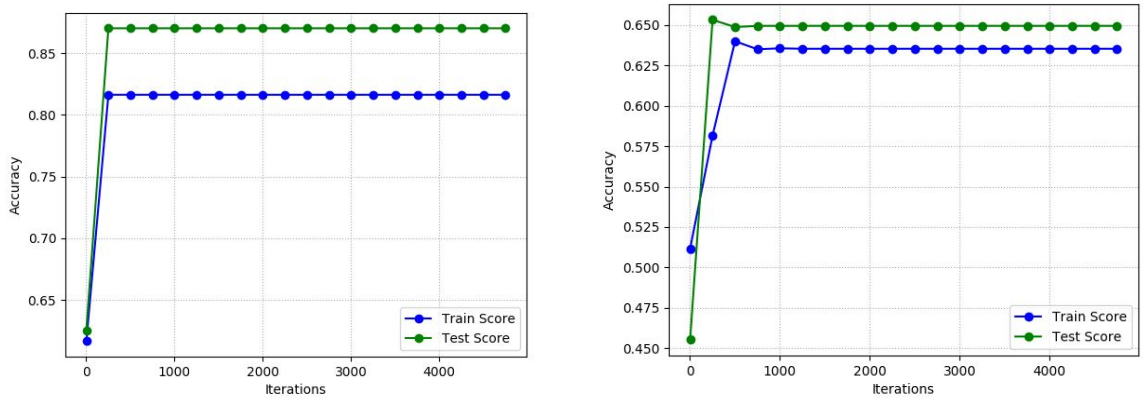

Figure 14 Iteration chart for Heart Disease (left) and Credit Card (right) datasets

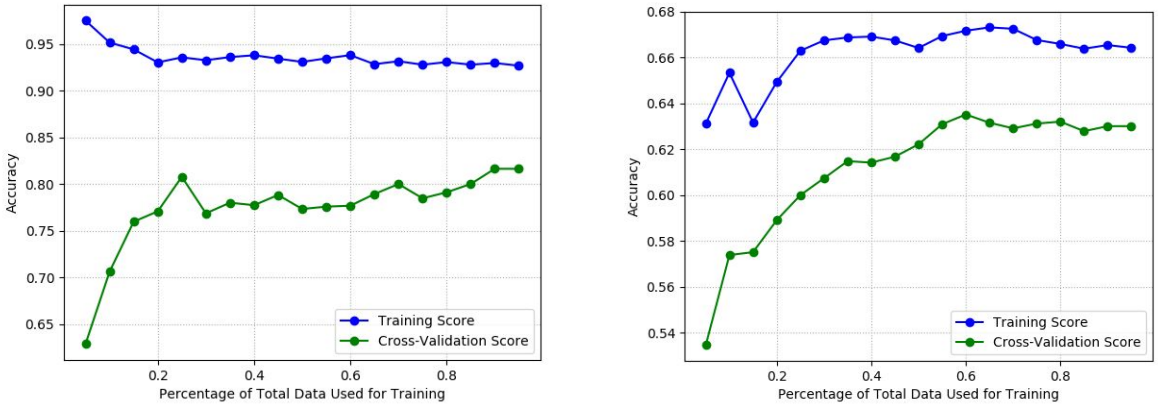

Figure 15 Learning curve for Heart Disease (left) and Credit Card (right) datasets

Similar to the neural networks, with respect to the iterations, the SVM reached its upper bound of performance quite quickly; in less than 1000 iterations it reached it's top score. With respect to the learning curve, the learner would not have benefitted from much more data. For both datasets, by the time it used half of the data available, it reached its upper bound on performance.

Conclusion

In general, the learners had similar results; the Heart Disease dataset test score clustered around .85 while the Credit Card dataset clustered around .65. If i had more time, I'd like to try two things. First I'd like to invest in more feature engineering. Since the learners clustered with similar performance, it suggest there was a limitation to the data I was using. Secondly, I'd like

to procure more data and see if the Neural Network continued to improve. It seems promising that the Neural Network continued to improve as it received more data!