SUPERVISED LEARNING

Aayush Kumar

T-SQUARE ID: akumar372

Introduction

Datasets

To explore the algorithms below, I chose two datasets such that one that offered a handful of features and arguably had related patterns in the labels discernible to human observation and intuition, while the other one offered more features and far more intricate relations that governed the labels of each instance. As a result, for the former I settled on a gym dataset from Kaggle that documents the crowdedness of campus gyms over the course of a year, a real world situation that I have domain knowledge of and wanted to see how well and quickly each algorithm picked up on certain patterns that we students observe along with those which we perhaps overlook. For the second, I chose a song dataset from the UCI ML Repository that archived the timbre of songs since the mid 1900s and could be used to predict which year a song was released in based on its sound characteristics.

	GYM DATASET	SONG DATASET
Total Instances:	62184	515345
Total Features:	10	90
Used Instances	20000 (15000 Train + 5000 Test)	75000 (52500 Train + 22500 Test)
Used Features	8	12

Transforming from Regression to Classification

While I found both datasets to be quite interesting, I quickly realized that I would have to modify the labels to convert them from Regression Problems into Classification Problems by bucketing ranges of values as individual labels. At first, I questioned how well my classifiers would do on a "converted" regression problem relative to one which is more typical of classification, but decided that this would be a quality that makes my problems interesting- perhaps there would be some algorithms that are more robust to this type of situation. In the case of the Gym Dataset, the data labels were originally in units of people from 0 to 160, so I replaced these labels with corresponding 20-person buckets and thereby producing labels of "0-19 people", "20-39 people", "40-59 people", etc. Similarly, the year prediction dataset had to be bucketed from individual years into decades, such that the regression problem was converted into a decade classification. In doing so, I wondered whether doing this label bucketing would be better to do before the labeled data was fed to my algorithms (pre-bucketing) or after my algorithms had predicted labels (post-bucketing). For the first, we give the algorithms more opportunities to capture the common characteristics of a larger label space, but this could also reduce the specificity of certain cases. For instance, if a song in 1991 had characteristics that clearly distinguished it from that of a song from 1998, then pre-bucketing these under the same umbrella of the 1990s decade hides this information from our training algorithms. On the other hand, if the songs from the 1990s decade had a unique quality to them that overall distinguish them from 2000s decade then post-bucketing will not allow our algorithms to recognize this pattern. In the end, I opted for pre-bucketing simply because I felt that the latter scenario took precedence and therefore it would be better to pre-bucket my labels.

Normalizing Training Input

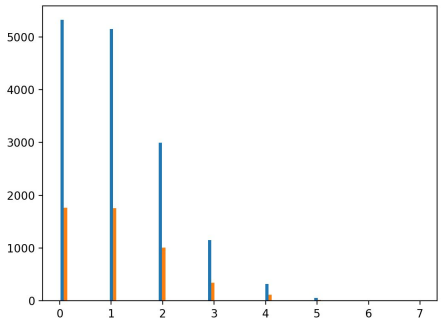
While incorporating scikit-learn's Multi-Layer Perceptron classifier, I noticed they mentioned how "Multi-layer Perceptron is sensitive to feature scaling, so it is highly recommended to scale your data. For example, scale each attribute on the input vector X to [0, 1] or [-1, +1], or standardize it to have mean 0 and variance 1."[1] This got me thinking about how normalizing my data could affect other algorithms, especially something like K-Nearest Neighbors where the distance between data points is fundamental to the algorithms prediction. Intuitively, it made sense that scaling my inputs would ensure that no single feature that happens to have a larger range of values would play a disproportionate role in the overall Minkowski metric. Upon standardizing my inputs for this, I noticed approximately a __ % improvement in overall performance. From here I begin testing the effects normalization would have on other algorithms- if I saw an improvement in the Decision Tree classifier then I could justify using normalization for Boosting, as I had been using an implementation of AdaBoost with a Decision Tree as my weak learner. In the end, however, I decided against this as I saw no noticeable improvement in the accuracy- logically speaking, it would not make as much sense to normalize data as a Decision Tree will only be altered if the apparent ordering of the data is changed somehow. It is, in essence a series of if statements that govern split across the levels of the decision tree, and the cutoff which splits the attributes would simply be normalized as a result of standardizing the input. Thus we are left with our Support Vector Machine- where as per the literature available online[2] also benefits from standardization of input during the preprocessing stage.

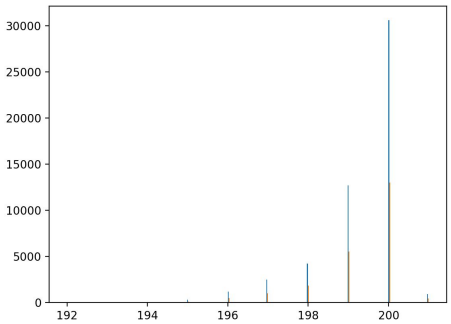
Cross Validation Procedure

As per the guidelines of this project, I first aimed to optimize the hyperparameters of each of my algorithms via Cross Validation, but did so in two stages: (1) Used Grid Search to narrow down the originally large search space, often logarithmically and (2) Tested out hyperparameters within now refined search space via K-Fold Cross Validation. In both stages, I ensured that my training data was split for cross validation via Stratified Sampling, to ensure that the distribution of labeled instances was held consistent across all subsamples I took from the gym and song training data. The below figures show the distribution of the __ set and the __ set. To gauge performance of each algorithm, I used number of correct predictions divided by number of total predictions.

Visualization of Sampled Datasets

We see that the distribution of labeled data is retained from our Training Data (Blue) in our sampled, held out Testing Data (Orange). On the left, we see the distribution for our Gym Crowdedness Data, and on the right is our Song Prediction Data. This was accomplished via the Stratified Variation of K Folds. In the end, training this way helps us account for unseen similarly distributed data.





[1] 1.17.8. Tips on Practical Use, http://scikit-learn.org/stable/modules/neural_networks_supervised.html
[2] http://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html

Algorithms & Analysis

1. DECISION TREES

Grid Search on Max Depth and Min Samples Split

When using Sci-Kit's Decision Tree Classifier on both datasets, I decided to incorporate pruning via enforcing a max-depth on the generated trees- in both cases, this did not necessarily result in a better score, but more so a less complex tree that described the data just as well. I also varied the minimum samples split parameter of my classifier to see if changing the number of samples needed to warrant a possible split at each node of the decision tree would make a difference instead of the default 2. My thought in doing so was that if any split at a node of the decision tree only accounts for differentiating between 2 samples, then it is likely to be far too tailored to my training data, and that any splits on large amounts of samples would indeed reveal the actual rules that governed each dataset.

GYM DATASET

Max Depth \ Min Samples Split	2	22	42	62	82	102	122	142	162	182	202	222	242	262	282	302	322	342	362	382
1	0.552	0.552	0.552	0.552	0.552	0.552	0.552	0.552	0.552	0.552	0.552	0.552	0.552	0.552	0.552	0.552	0.552	0.552	0.552	0.552
3	0.579	0.579	0.579	0.579	0.579	0.579	0.579	0.579	0.579	0.579	0.579	0.579	0.579	0.579	0.579	0.579	0.579	0.579	0.579	0.579
5	0.612	0.612	0.612	0.612	0.612	0.612	0.612	0.612	0.612	0.612	0.612	0.612	0.612	0.612	0.612	0.610	0.603	0.601	0.599	0.598
7	0.644	0.643	0.643	0.642	0.643	0.642	0.642	0.639	0.639	0.637	0.635	0.634	0.633	0.632	0.631	0.628	0.620	0.618	0.615	0.615
9	0.668	0.666	0.666	0.663	0.662	0.662	0.660	0.657	0.657	0.652	0.647	0.646	0.645	0.645	0.644	0.641	0.633	0.631	0.628	0.628
11	0.687	0.687	0.683	0.678	0.674	0.673	0.670	0.667	0.665	0.661	0.656	0.655	0.653	0.652	0.649	0.647	0.638	0.636	0.633	0.632
13	0.697	0.696	0.692	0.686	0.682	0.678	0.674	0.670	0.667	0.663	0.658	0.657	0.655	0.654	0.651	0.646	0.638	0.636	0.633	0.632
15	0.708	0.702	0.696	0.690	0.685	0.681	0.677	0.673	0.670	0.665	0.660	0.658	0.655	0.654	0.651	0.646	0.638	0.636	0.633	0.633
17	0.719	0.709	0.701	0.694	0.689	0.683	0.678	0.673	0.670	0.665	0.660	0.658	0.655	0.654	0.651	0.646	0.638	0.636	0.633	0.633
19	0.720	0.710	0.701	0.694	0.688	0.682	0.677	0.673	0.669	0.664	0.659	0.658	0.655	0.654	0.651	0.646	0.638	0.636	0.633	0.633
21	0.719	0.710	0.701	0.694	0.689	0.683	0.678	0.673	0.669	0.664	0.659	0.658	0.655	0.654	0.651	0.646	0.638	0.636	0.633	0.633
23	0.722	0.711	0.701	0.694	0.689	0.683	0.678	0.673	0.669	0.664	0.659	0.658	0.655	0.654	0.651	0.646	0.638	0.636	0.633	0.633
25	0.718	0.710	0.701	0.694	0.689	0.683	0.678	0.673	0.669	0.664	0.659	0.658	0.655	0.654	0.651	0.646	0.638	0.636	0.633	0.633
27	0.716	0.711	0.701	0.694	0.689	0.683	0.678	0.673	0.669	0.664	0.659	0.658	0.655	0.654	0.651	0.646	0.638	0.636	0.633	0.633
29	0.718	0.710	0.701	0.694	0.689	0.683	0.678	0.673	0.669	0.664	0.659	0.658	0.655	0.654	0.651	0.646	0.638	0.636	0.633	0.633

Without any pre-pruning of the maximum depth, the Decision Tree trained on the Gym Dataset achieved a 72% Cross Validation Accuracy at a depth of 31. Therefore, I grid searched maximum depths ranging from 1 to 29 to see how fruitful these last levels of the tree were and if I could generalize my model better. I found that capping the depth at 29 levels yielded the same exact accuracy so in favor of Occam's Razor, I opted for the slightly more simpler model in the end. In regards to my assumption about tuning the Min Samples Split, cross validation revealed that I was wrong and that accuracy benefitted slightly more from splitting a minimum of 2 samples.

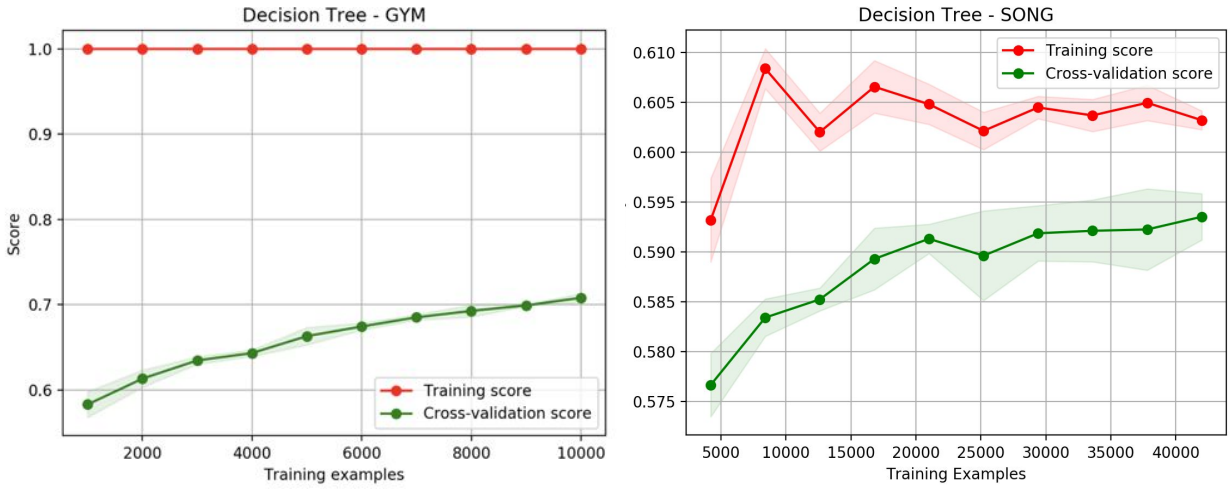
SONG DATASET

Max Depth \ Min Samples Split	2	22	42	62	82	102	122	142	162	182	202	222	242	262	282	302	322	342	362	382
1	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582
3	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582	0.582
5	0.590	0.590	0.590	0.590	0.590	0.590	0.590	0.590	0.590	0.590	0.590	0.590	0.590	0.590	0.590	0.590	0.590	0.590	0.590	0.590
7	0.592	0.592	0.592	0.592	0.592	0.592	0.592	0.592	0.592	0.593	0.593	0.593	0.593	0.593	0.593	0.593	0.593	0.593	0.593	0.593
9	0.585	0.585	0.586	0.586	0.587	0.587	0.588	0.589	0.590	0.590	0.591	0.591	0.591	0.591	0.591	0.592	0.592	0.593	0.593	0.593
11	0.567	0.568	0.570	0.574	0.577	0.579	0.581	0.583	0.585	0.586	0.587	0.588	0.589	0.589	0.590	0.591	0.592	0.592	0.593	0.593
13	0.545	0.549	0.558	0.564	0.569	0.573	0.577	0.579	0.582	0.583	0.585	0.586	0.587	0.588	0.589	0.590	0.591	0.592	0.592	0.592
15	0.521	0.532	0.547	0.556	0.563	0.569	0.574	0.577	0.580	0.582	0.585	0.586	0.587	0.588	0.589	0.590	0.591	0.592	0.592	0.592
17	0.504	0.520	0.540	0.552	0.561	0.567	0.572	0.575	0.579	0.582	0.584	0.586	0.586	0.587	0.588	0.589	0.591	0.591	0.592	0.592
19	0.490	0.512	0.535	0.548	0.558	0.566	0.571	0.575	0.579	0.581	0.584	0.586	0.586	0.587	0.588	0.590	0.591	0.591	0.592	0.592
21	0.480	0.506	0.532	0.547	0.557	0.565	0.571	0.574	0.579	0.581	0.584	0.585	0.586	0.587	0.588	0.589	0.591	0.591	0.592	0.592
23	0.473	0.503	0.530	0.545	0.556	0.564	0.570	0.574	0.578	0.581	0.584	0.585	0.586	0.587	0.588	0.589	0.591	0.591	0.592	0.592
25	0.471	0.501	0.530	0.545	0.556	0.564	0.570	0.574	0.579	0.581	0.584	0.585	0.586	0.587	0.588	0.589	0.591	0.591	0.592	0.592
27	0.469	0.500	0.529	0.545	0.556	0.564	0.570	0.574	0.578	0.581	0.584	0.585	0.586	0.587	0.588	0.589	0.591	0.591	0.592	0.592
29	0.469	0.501	0.529	0.545	0.556	0.564	0.570	0.574	0.578	0.581	0.584	0.585	0.586	0.587	0.588	0.589	0.591	0.591	0.592	0.592

The much more interesting case of pruning came when I trained on the song dataset. Here, I found that without any pre-pruning, my decision tree grew to a depth of 37. However, after running my grid search on max depth and min samples split, I found that aggressively pruning my tree to just 7 levels and setting my minimum samples to split on at 342 increased my cross validation accuracy by nearly 10%. This shows that a decision tree without any pruning can drastically overfit to data and that the decision within the first 7 levels were rules that were most likely valid rules for predicting the decade of a song's

release while those thereafter if left unpruned would simply draw false conclusions based on believing our training data too much.

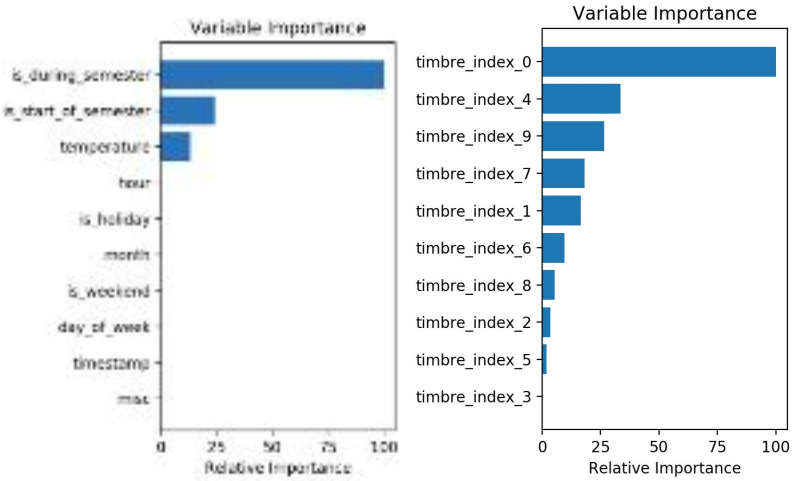
Score vs Varying Training Examples



Although the Decision tree does show some promise with the datasets, we see their core flaw in that they are prone to overfitting to training data and don't learn relatively enough despite being given more and more training examples. To combat this, we could perhaps consider even more forms of pruning but in the end, decision trees are indeed notorious for having high variance and low bias, thus not generalizing terribly well to some datasets. Nevertheless, we do see that there were some fundamental rules that the tree was able to extrapolate for both datasets, and brings us to another insight that we can make about both problems.

Feature Importance

Specifically for classifying the crowdedness of gyms, we see that 3 factors that heavily influence crowdedness are whether a semester is under way, whether a semester has begun, and the temperature (left). Because the song dataset benefited greatly from pruning, we know that the notable features found in these first several layers would be useful for the general classification task. In this case the first numeric attribute in the 12-length timbre vector describing each song was massively important in classifying the decade said song was released. However, for further demonstrating where decision trees greatly suffer.classification, the data that demands more intricate patterns to be extrapolated and simply cannot be fully understood by more fundamental rules.



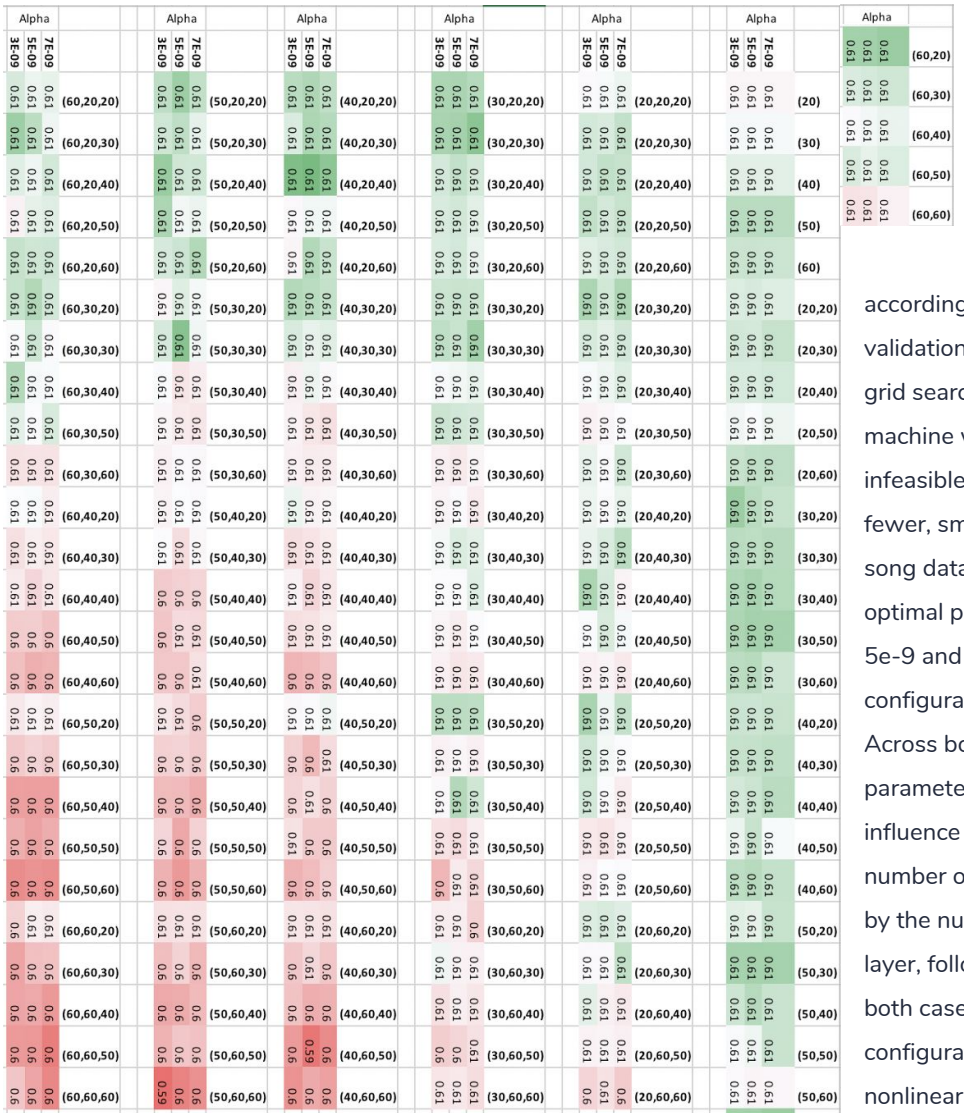
2. NEURAL NETWORKS

Grid Search on # of Hidden Layers, # of Neurons/Layer, Alpha

When attempting to find the optimal architecture of my neural network, I varied the # of Hidden Layers, # of Neurons Per Layer, and Alpha, since I felt that the necessary complexity of a neural net comes from these structural aspects.

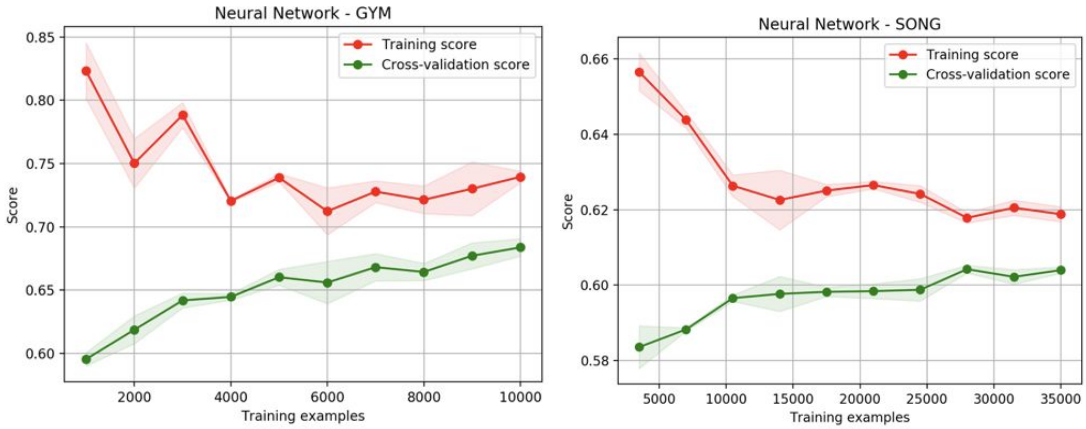
In the case of the Gym classifier, we see a general trend of improved accuracy with 3 hidden layers as opposed to less, and a better perceptrons/layer configuration at (60, 50, 30)[3]. I was genuinely surprised to see how much the gym problem benefited from large numbers of perceptrons in each layer, simply because it had only 10 attributes. Smaller learning rates doesn't seem to come into effect as much here, as our gradient descent loss function appeared to be able to find to global minimum regardless of the magnitude of our alpha. In other words, taking bigger steps to find the optimal weights for each configuration did not seem to skip the best weights.

[3] Hidden layer/Perceptron configurations are denoted by a tuple where the length of the tuple indicates how many hidden layers and the value of each element in the tuple indicates how many perceptrons are in that layer



For the song classifier, we saw an optimal configuration at 3 hidden layers once again with a (40, 20, 40) perceptron configuration also according to grid search cross validation. Because doing a similar grid search as before on my machine would be computationally infeasible, I opted for exploring fewer, smaller learning rates for the song dataset. Here, we see that the optimal parameters are an alpha of 5e-9 and a perceptron configuration of (40,20,40). Across both datasets, the parameters ordered by their influence appeared to be the number of hidden layers, followed by the number of perceptrons per layer, followed by learning rate. In both cases, we see that the best configurations imply that our data is nonlinearly separable.

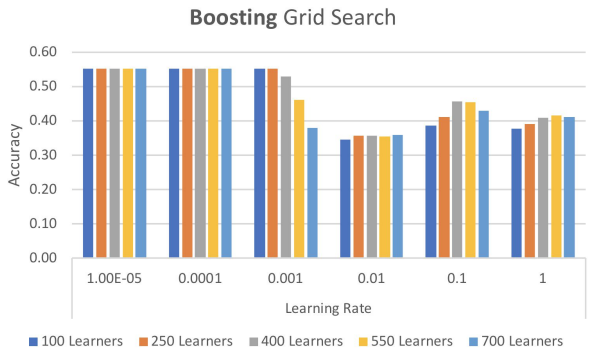
Score vs Varying Training Examples



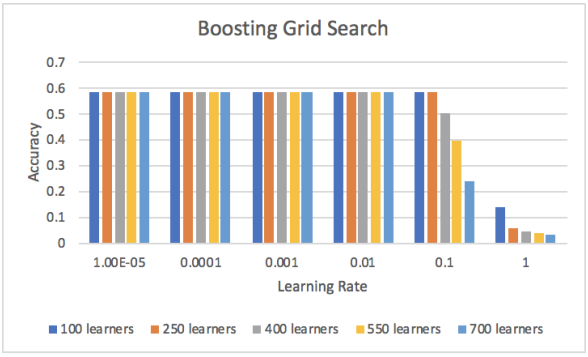
Perhaps more noteworthy was how ideal both models turned out to be via the graphs above, with a optimal middle ground between high bias and high variance. We see that in both cases, our training accuracy and cross validation test accuracy converged to at most a 5% difference, which shows that we neither overfit to our training data and have high variance nor did we underfit to it and have high bias. Such graphs show that we have encountered an optimal balance between how much we are learning from data given and generalizing to accommodate unseen data.

3. BOOSTING

Grid Search on Number of Learners and Learning Rate



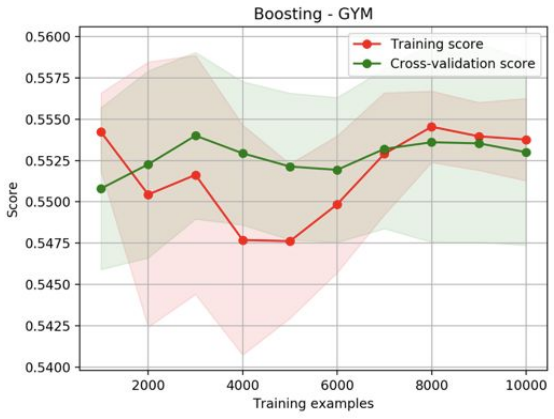
We see that Boosting was not particularly effective when it came to running on the gym dataset (left). In it, we see that smaller learning rates and higher number of learners did not particularly help after the 0.001, 250 mark. As a result, I decided to settle on that for my parameters to follow Occam's Razor. Interestingly enough, we see that a learning rate of 0.1 out performed a learning rate of 0.01 regardless of the number of weak learners used for each.

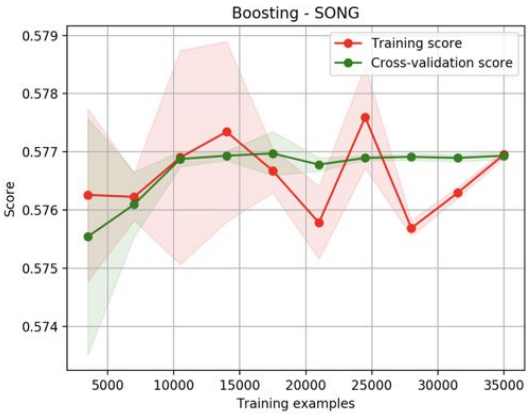


In the case of the song dataset (left), we saw mediocre performance again where smaller learning rates and higher number of learners did not help after the 0.1, 250 mark. Again, I followed Occam's Razor but noted that, unlike the previous dataset, we see a much more expected behavior with smaller learning rates and larger numbers of learners linearly improving until plateauing.

In both cases, we see that the extremely small learning rates play a significant role in finding optimal weights for the final prediction only up until a certain point. After that, despite trying to strike a balance between smaller learning rates complemented by larger numbers of learners, there comes a point when no matter how small the Learning Rate is, each weak learner is only able to contribute so much. Overall, I realized that several decision stumps as part of Adaboost don't do nearly as well as a decision tree, and this might show that when combining multiple "decisions", the information gain/entropy approach can be significantly more effective than the averaging approach of boosting.

Score vs Varying Training Examples



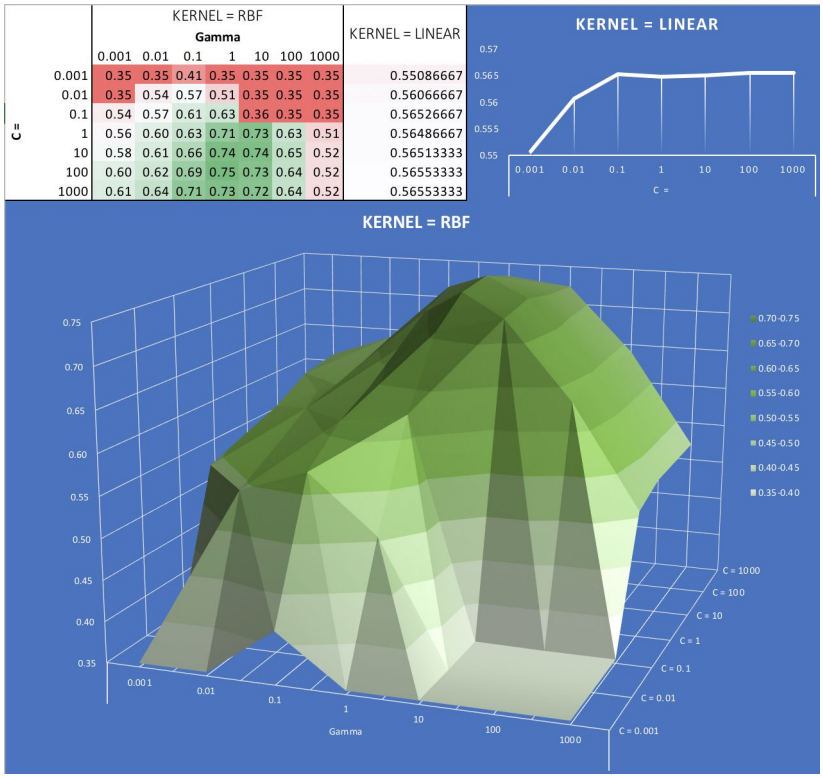


With boosting, I noticed that my classifiers didn't have any particularly noticeable pattern for bias and variance. As a result, the only thing I could extrapolate is that I certainly did not overfit much in both cases, something that we have

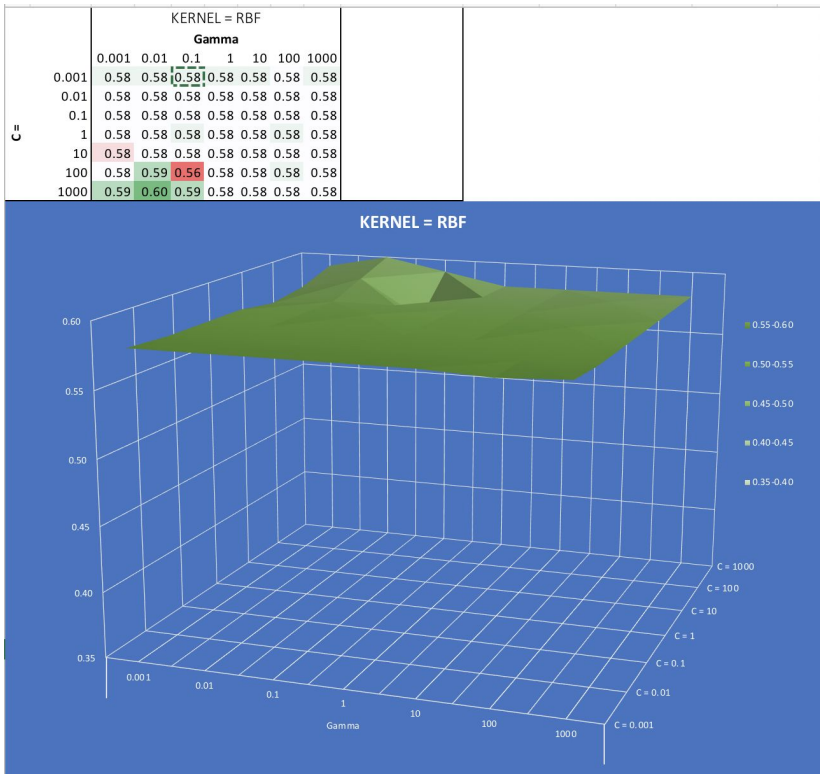
known to be difficult as the algorithm itself is quite robust to it. Nevertheless, we see that the algorithm does not benefit much from increased number of training examples, and in the end this is undesirable character

4. SUPPORT VECTOR MACHINES

Grid Search on Kernel Type, C, and Gamma



C= \ Gamma	0.001	0.01	0.1	1	10	100	1000	KERNEL = LINEAR
0.001	0.35	0.35	0.41	0.35	0.35	0.35	0.35	0.55086667
0.01	0.35	0.54	0.57	0.51	0.35	0.35	0.35	0.56066667
0.1	0.54	0.57	0.61	0.63	0.36	0.35	0.35	0.56526667
1	0.56	0.60	0.63	0.71	0.73	0.63	0.51	0.56486667
10	0.58	0.61	0.66	0.74	0.74	0.65	0.52	0.56513333
100	0.60	0.62	0.69	0.75	0.73	0.64	0.52	0.56553333
1000	0.61	0.64	0.71	0.73	0.72	0.64	0.52	0.56553333



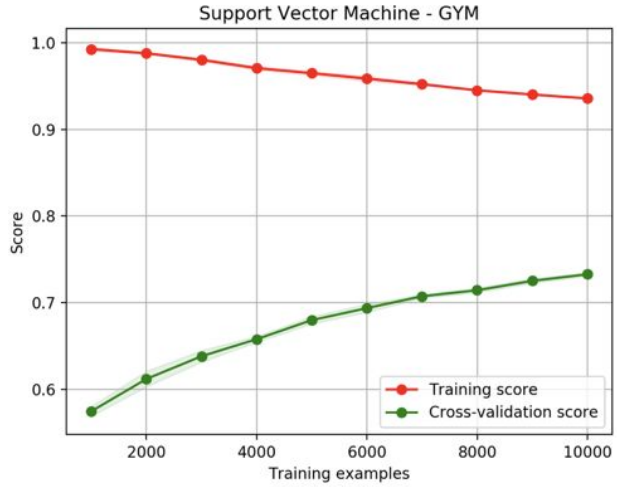
C= \ Gamma	0.001	0.01	0.1	1	10	100	1000
0.001	0.58	0.58	0.58	0.58	0.58	0.58	0.58
0.01	0.58	0.58	0.58	0.58	0.58	0.58	0.58
0.1	0.58	0.58	0.58	0.58	0.58	0.58	0.58
1	0.58	0.58	0.58	0.58	0.58	0.58	0.58
10	0.58	0.58	0.58	0.58	0.58	0.58	0.58
100	0.58	0.59	0.56	0.58	0.58	0.58	0.58
1000	0.59	0.60	0.59	0.58	0.58	0.58	0.58

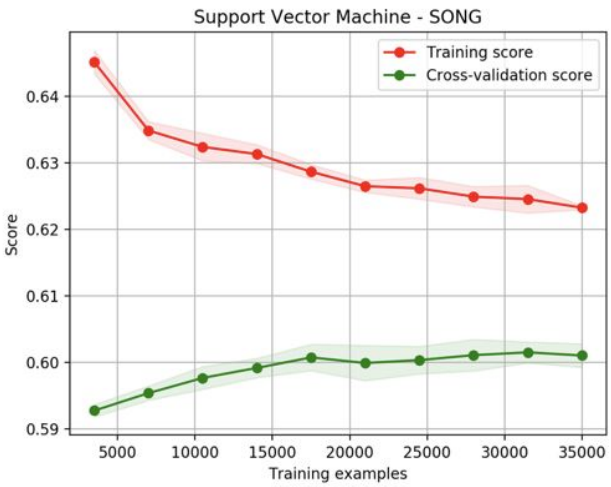
Support Vector Machines by far took the longest to train, especially when testing the 'rbf' parameter. I assume that when it comes to projecting our training data into higher dimensions, that is where this algorithm suffers the most in terms of runtime and computational efficiency. Nevertheless, I was able to grid search and find a distinct global maximum for my parameters, as is visible by the 3D surface plotted above. If we were to take runtime into heavier consideration, however, a linear SVM would most likely be more appropriate as it offers a much better accuracy to runtime ratio, a metric we will explore in our final conclusions. We see that the C parameter, which controls the influence of each individual support vector, plays a large roll in the performance for the gym dataset- having a larger amount of influence shows that the decision boundaries should be trusted.

Because the song dataset (left) was evidently a nonlinear classification problem, I decided to only grid search possible rbf configurations. Here, however, I saw a much smaller variation in my grid searched results, as everything ranged from 0.58 plus or minus (0.0015) % accuracy. Nevertheless, there is a global maximum here as well, albeit less

prominent than the previous example. Here we see that higher C confidence in our individual support vectors pays dividends, as well as a relatively lower Gamma which inches us closer to somewhat low bias and slightly high variance models.

Score vs Varying Training Examples



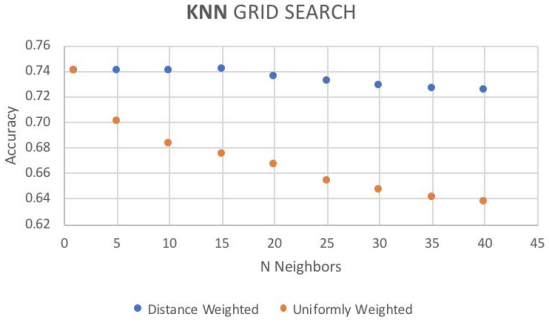


We see that in the case of the both classification problems, SVM's did a fairly good job when observing their learning curves. In both cases, they seem to be doing better slowly but steadily as the training accuracy and cross validation testing accuracy converge with more and more training examples. This is a clear indication of a relatively ideal scenario where there is not excessive bias and underfitting nor is there excessive variance and overfitting. I assume that the nonlinear kernel of the SVM has a large part to play in this, as projecting data into higher dimensions, although computationally expensive, allows for much more analysis to draw our hyperplane separation. As this and KNN take the n-space location of each training data to make inferences about the classification problem, we can see that this method takes a more complex approach to analyzing and, more importantly, manipulating those n-space locations for improved analysis. I do feel that svm would be useful for grouping with other classifiers, something which we can detail later.

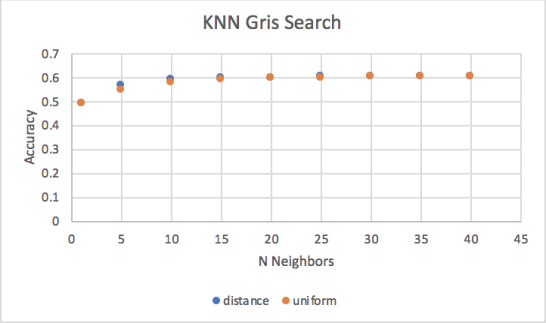
5. K-NEAREST NEIGHBORS

Grid Search on N Neighbors and Weight Metric (Distance or Uniform)

KNN was exceptionally fast with few neighbors, and begin to slow down noticeably when the number of neighbors approached 40. This computationally makes sense, as larger k's force our algorithm to find more and more closer points to the instance in question. When I realized that the default implementation of scikit-learn K-Nearest Neighbors algorithm gives uniform weighting to all of these points, I had a hunch that weighting them by euclidean distance would noticeably improve prediction accuracy. Remarkably, as naive as K-Nearest Neighbors is relative to the other algorithms in question, it does a very good job of making accurate predictions. In the case of the gym dataset, we see how 15 neighbors produced the optimal accuracy, while in the case of the song data we found an optimum at 35 neighbors. This type of local maxima can be explained by the notion that not only do we need enough neighbors to get better context for our prediction, but having too many

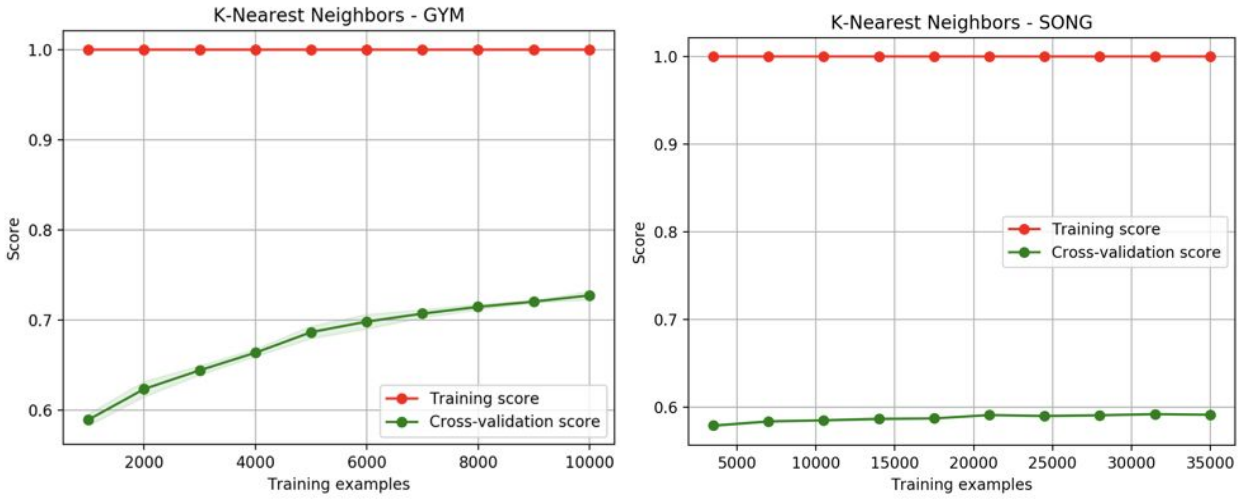


neighbors in consideration can cause too much noise, despite being weighted by distance, to making a correct prediction.



In the case of the song dataset, we saw a different situation where when weighted by distance, the algorithm performed consistently the same as when weighted uniformly. This could be as a result of how in some cases, normalization of our data affects the necessity of weighting distance and how simply taking the K-Nearest Neighbors' vote is enough to produce a respectable prediction. In the end, the best parameters of k = 35 neighbors with a distance metric only marginally outperformed to the other configurations.

Score vs Varying Training Examples



Here we see that K nearest neighbors is fairly prone to high variance, and therefore overfitting. I was not terribly surprised that this happened with the gym dataset, as using only 15 neighbors can understandably result in relying too much on certain conditions observed in gym crowdedness. However, I believe that the Gym KNN model does not get hindered by this, as from a logical standpoint one could argue that people use their prior experiences of the crowdedness to assess if the gym will be similarly crowded. Personally, if I see that the gym is crowded friday afternoons, then I grow to expect that behavior regularly and this is something KNN would be able to implicitly leverage because the conditions for each friday afternoon would be relatively close via euclidean distance. However, the overfitting present in the song dataset was unexpected because the use of 35 neighbors made me expect less overfitting in the long run- the idea being that with more neighbors we become less reliant on certain nuances of the training data and instead have a more general vote that more training examples can contribute to. Nevertheless, I am still incredibly surprised at how well the algorithm performs considering how naive it is. Time and time again, we see that KNN is able to produce respectable results- in fact, an NLP seminar I recently visited noted how well the algorithm does on basic Text Classification.

Reflections

Overall Performance (Accuracy / Runtime)

	Gym Dataset	Song Dataset
Decision Trees	73.22% / 0.009738 sec = 75.19	58.37% / 0.099206 sec = 5.88
Neural Networks	69.04% / 1.662186 sec = 0.42	60.83% / 2.673170 sec = 0.23
Boosting	55.64% / 0.542818 sec = 1.03	57.83% / 8.222503 sec = 0.07
Support Vector Machines	75.86% / 1.808803 sec = 0.42	59.88% / 46.895473 sec = 0.01
K-Nearest Neighbors	75.34% / 0.037465 sec = 20.11	59.12% / 0.013283 sec = 44.51

Overall, I found that the best performing algorithm strictly based on accuracy for my gym classification problem was the Support Vector Machine while that of the Song classifier was Neural Networks. However, if we take runtime into consideration and formulate adjusted scores for each of the algorithms based on (Accuracy / Runtime) in order to penalize algorithms that take extremely long to run, we see that the best performing algorithm based on our new heuristic is Decision Trees for the Gym Dataset with a score of 75.19 while that of the Song Dataset is K-Nearest Neighbors. I had a hunch going in that Decision Trees would do relatively well with the Gym dataset, as when tasking a human with the same prediction problem, I would argue that we use a decision tree approach. We know some factors to be true: how if it's extremely hot or cold then people are less likely to go exercise, how students are more motivated/have more time to work out at the beginning of the semester, etc. It only makes sense that this pseudo-decision tree approach could justify why such an algorithm would perform so well on this classification problem. We also see that SVMs take three orders of magnitude longer to train for a extra 2.5% accuracy, something which all things considered does not seem as worthwhile if we were choosing a classifier to stick with moving forward. Frankly, I was surprised at how well KNN predicted the Song Test set, achieving within 2% accuracy of our most accurate algorithm, and thus its final adjusted score becomes so high compared to the rest of the algorithms. However, it does make sense why the best performing algorithm was Neural Networks from an accuracy-only perspective. Much of the numeric qualities of ANN were able to properly leverage the numeric information in the timbre vector of the songs provided.

Combining The Best of Multiple Classifiers

One afterthought I had from this project was how we might benefit from combining multiple classifiers to produce a better overall. I found that Decision trees and KNN are both quite good at extrapolation simple but effective classification criteria for datasets. Perhaps these could be used as the first steps of a more complex classifier, where if we still have some indecision, we could resort to using more complex classifiers like SVM and Neural Nets.

Citations

Algorithms

1. http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
2. http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
3. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
4. http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
5. http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Datasets

1. https://www.kaggle.com/nsrose7224/crowdedness-at-the-campus-gym
2. http://archive.ics.uci.edu/ml/datasets/YearPredictionMSD