CS 7641 Machine Learning
Assignment 1

David Yun

September 23, 2018

Abstract

The implementation (in Python2.6 or Python3, depending on the algorithm), and basic fundamentals behind the following Machine Learning Algorithms will be discussed in detail:

1. Decision trees with some form of pruning
2. Boosting
3. Support Vector Machines (SVM)
4. k-Nearest Neighbors (kNN)
5. Neural Networks

The full code files are available on github[1] Please refer to the README.txt file for concise instructions on how to run the python files associated with each of the aforementioned algorithms. Further details behind the datasets, including where to download them, are provided in the README.

Contents

[1]David Yun's Github: https://github.com/tree-fiddy/Assignment_1

1 Datasets

The two datasets used are both sourced from the UCI Machine Learning Repository. The first one, the Credit Approval Data Set is a smaller dataset with 690 instances, and 15 attributes. As a full time employee at American Express, credit approval is highly relevant to my job. As such, I wanted to take this opportunity to see how Machine Learning can be directly applied to credit approval data. The second dataset, the MoCap Hand Postures Data Set is a much larger dataset with 78,095 instances, and 38 features. This dataset classifies 5 different hand gestures using 11 markers attached to each finger of a glove in a motion capture environment. The application of Machine Learning here is interesting because it provides a rudimentary step in the direction towards Virtual Reality- a burgeoning field directly applying Machine Learning to read & track objects.

I thought it would behoove me to use two very different datasets, both in scope, and size, to truly gain an understanding of the usefulness of Machine Learning. Not all models are built the same, and not all datasets benefit from a given model. Thus, getting the opportunity to apply the same, or similar, models on two fundamentally different datasets would help uncover where certain models prove useful, and where the same models might prove inadequate.

2 Decision Trees (via Random Forest) & Boosting

The accuracy of our decision tree relies on our tolerance for pre-pruning. In essence, one must specify *when* to stop growing the tree. On one extreme, we can set a condition to stop growing the tree once we reach a *pure*[2] leaf. However, this will likely make the tree longer than need be. A more depth-restrictive approach is to set an arbitrary threshold for Entropy in our Decision Tree, where we will halt further node and leaf generation if the current node produces leaves with Entropy values less than our specified threshold. To ensure the efficacy of our pre-pruning process, we also incorporate a limit to the depth of our tree to 5. This will handle extreme cases where our dataset is so large, that splitting across many features is possible. This also has the added benefit of dramatic performance improvement, without sacrificing accuracy, when compared to not setting a depth and letting the tree grow indefinitely.

As you can see, the model's accuracy reached an asymptote of around 50% accuracy for the credit card data, which indicates that setting a higher threshold for Entropy renders this

[2]A leaf is said to be "pure" if the leaf contains homogenous labels. By extension, this equates to a situation where Entropy equals 0.

model's efficacy no better than a coin flip. Thus, setting a pre-pruning specification is quite important here.

Our Boosting code uses the AdaBoost Classifier in sklearn, and uses Decision Trees as weak learners. We try to find the best model by iterating on different number of estimators (1,2,5,10,20,30,45,60,80,100) and alphas (-10^{-3} to 10^{-3}). We also iterate on training sample size to see if our model overfits the data, and to find a sufficient training sample size. In our case, our training accuracy did not converge with the validation set for either data sets.

2.1 Learning Curves Analysis

Hand Posture Data: The large dataset provided very great results using both algorithms. The Random Forest immediately achievs near 100% validation accuracy at low values of entropy, whereas Boosting produces a max accuracy of 84%. This boils down to Bagging vs. Boosting, and it is clear that our dataset performs much better by averaging the individuals results (Bagging) than by . It is also worth noting that tuning was a much more straightforward process via Random Forest Decision Trees, compared to Boosting. For one, tuning the alpha (regularization loss descent rate) was computationally expensive for little benefit, as shown in Figures **??** and **??**. As a result, I believe a Random Forest might work better with the Hand Posture data because the geospatial location of fingers associated with each hand posture has great discernability- perfectly leveraging the ID3 algorithm's strength in using Information Gain to identify which features discriminate effectively.

Credit Approval Data: Decision Trees with Random forests allow us to create a robust model with less variance which results in a reduction in overfitting[3]. With proper pruning (Entropy thresholds & Max Depth), one can further reduce the variance of the model and further decrease the chance of overfitting.

Table 1: Credit Approval: Pre-Pruning Spec Performance (forest size = 50)

Entropy Threshold	Model Accuracy
0.0	0.86
0.1	0.86
0.2	0.60
0.3	0.57
0.4	0.55
0.5	0.55
0.6	0.54
0.7	0.54

Hand Posture All the aforementioned patterns hold true for the much larger Hand Posture dataset. However, the computation was MUCH longer. While the small credit card approval data took 3 minutes for the highest max depth setting, the Hand Posture data took 3 hours. You can also see that the accuracy was much higher using this dataset. Having much more data points probably helped lift accuracy scores,

[3]By averaging several decision trees, the result is a less likely chance of overfitting

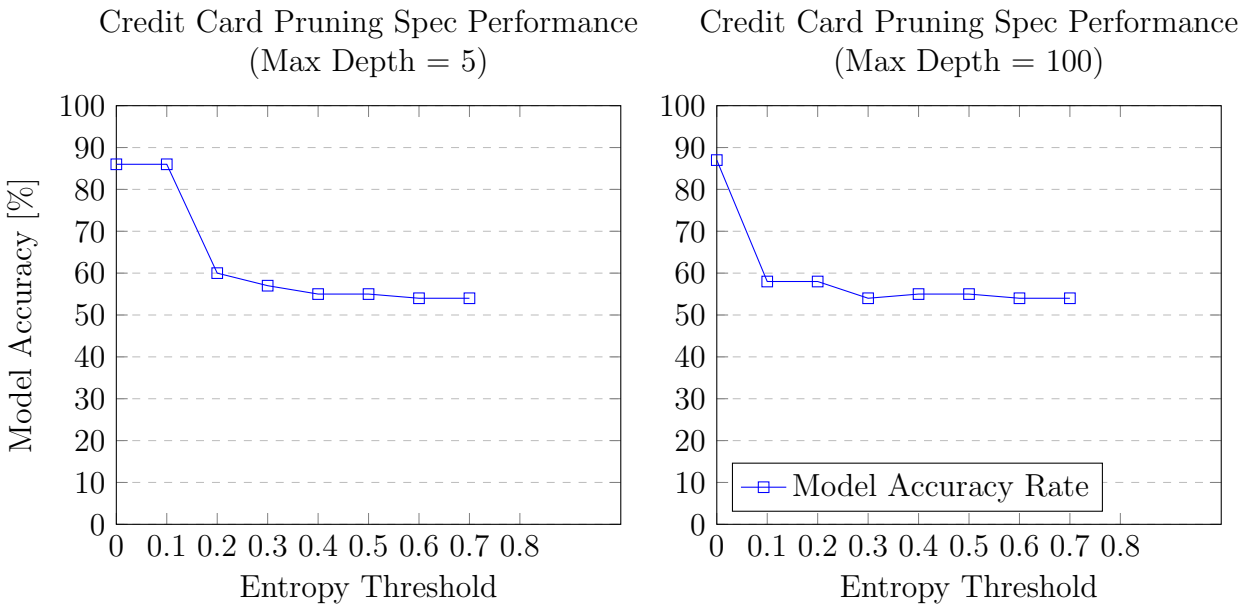


Figure 1: Learning Curves

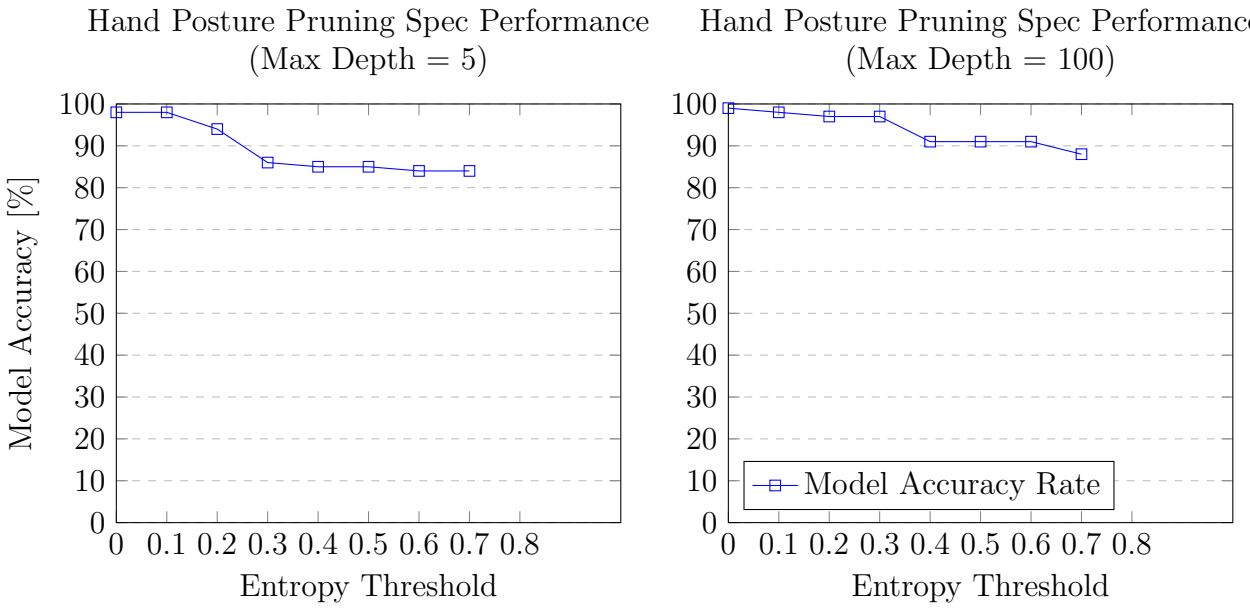


Figure 2: Learning Curves

Table 2: Hand-Posture: Pre-Pruning Spec Performance (forest size = 50)

Entropy Threshold	Model Accuracy
0.0	0.98
0.1	0.98
0.2	0.94
0.3	0.86
0.4	0.85
0.5	0.85
0.6	0.84
0.7	0.84

2.2 Model Complexity Analysis

Besides varying the Entropy Threshold & Max Depth, we can also vary the **forest size** hyperparameter in our random forest. Note here that we vary the hyperparameter, while holding the Entropy Threshold to 0.1 as supported by Figure 3. Note that the **Credit Card Data** doesn't really improve much with 10x more trees. However, the same isn't true with the much larger **Hand Posture Data**, where we see a meaningful improvement in performance as the number of trees is allowed to increase.

Hand Posture Data To assess the impact of our hyperparamters, I adjust the alpha to -1, learning rate to 0.02, and increase the number of estimators to 20. Figure 6(a) shows no discernable difference between the previous two results, which could be due to being "stuck" at a local minima and not finding the global minima of the loss function in the gradient descent step. However, even after setting the learning rate hyper parameter to an exaggerated number (100) yielded similar results, as shown in **??**.

3 SVM & k-Nearest Neighbors

We run an **SVM** model twice for each dataset- A linear SVM to start, and then an SVM with a RBF kernel for each data set. SVM, on average took considerable amount of time to train, while kNN was almost instant. We also run a **kNN** analysis by varying our k parameter, our loss function hyperparameter (Euclidean vs. Manhattan distance), and weight distance.

3.1 Learning Curves Analysis

Hand Posture Data: In Figure 10(a), we see that the **SVM** model performed poorly. To start, our training data just barely reaches 50% accuracy. A large part of this results is due to the high dimensional nature of our data. Our SVM is drawing the best fit line amongst all dimensions, and it is NOT able to reasonably classify the given data points. This is further evidenced by tuning alpha, which is a regularization parameter that will aid

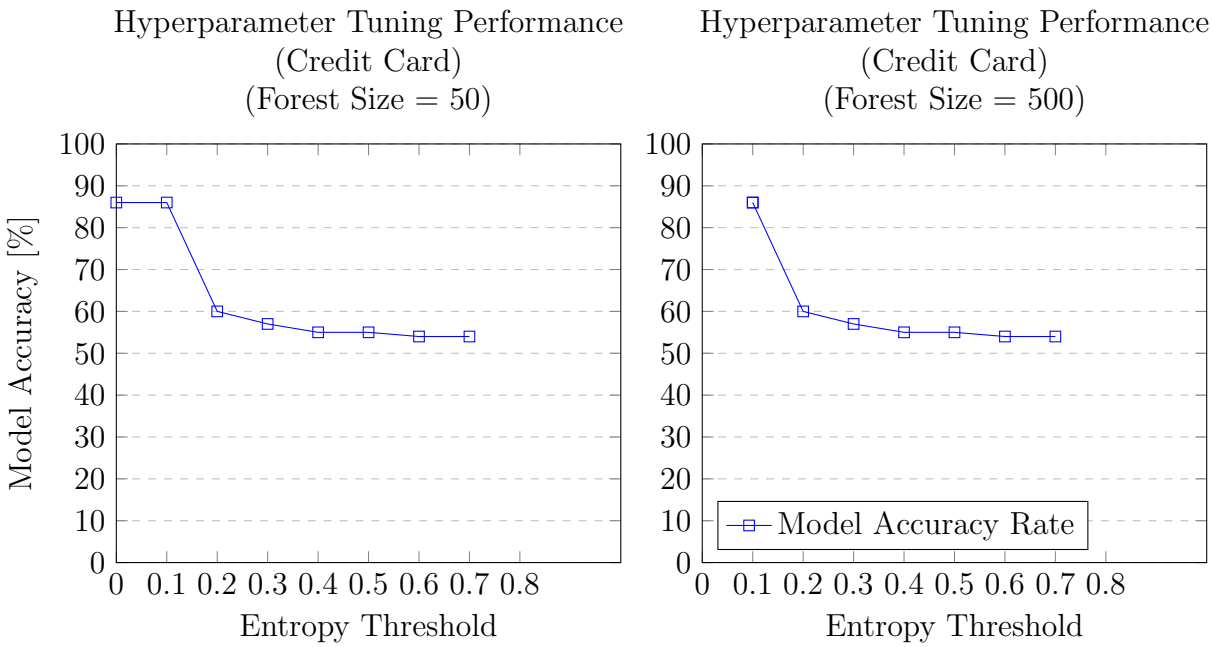


Figure 3: Model Complexity Curves (Credit Card Data)

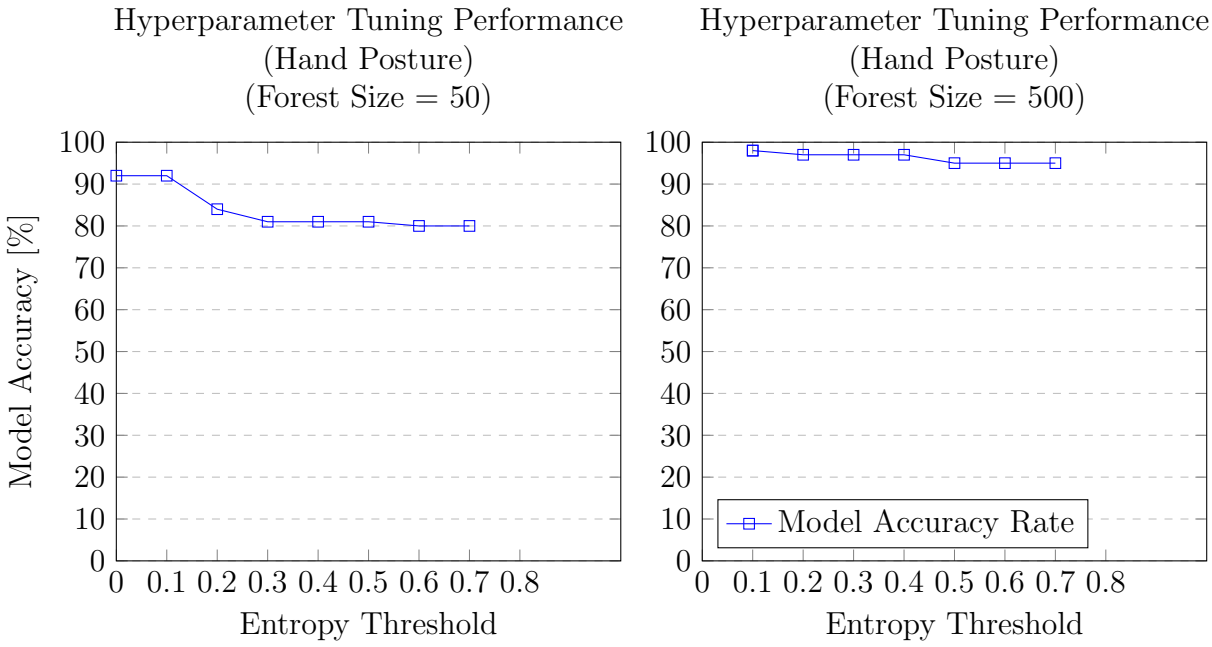
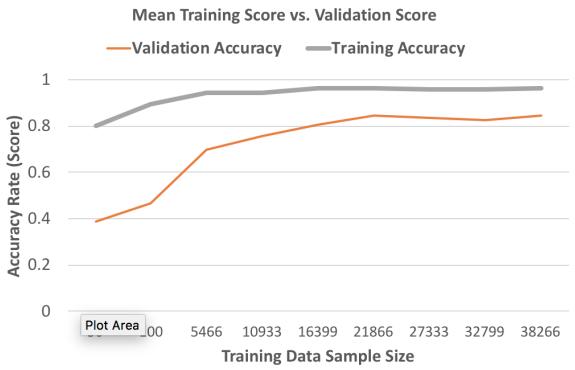
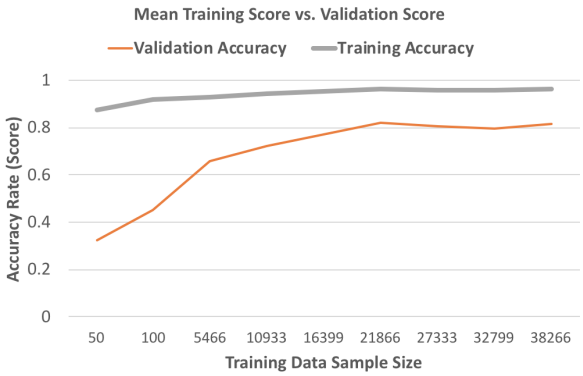


Figure 4: Model Complexity Curves (Hand Posture Data)

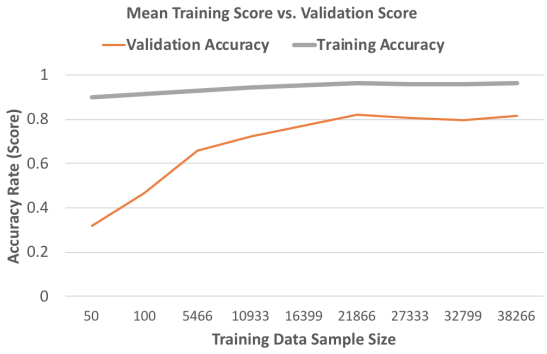


(a) n_estimators = 10



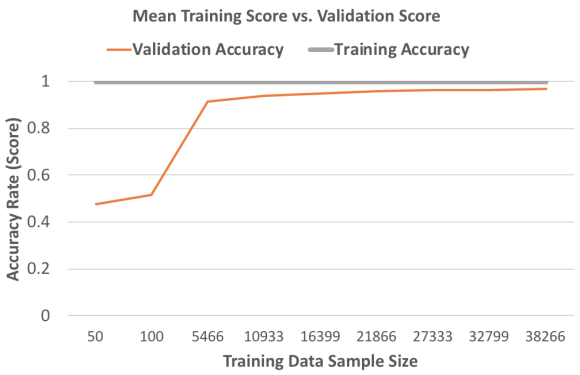
(b) n_estimators = 10

Figure 5: Learning Curves: Training vs. Validation Accuracy w/ Varying Training Set Size

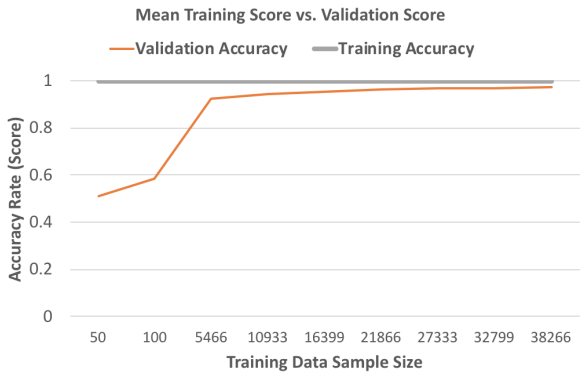


(a) alpha = -1, n_est = 20, learning rate = 0.02

Figure 6: Learning Curves: Training vs. Validation Sets w/ Varying Learning Rate

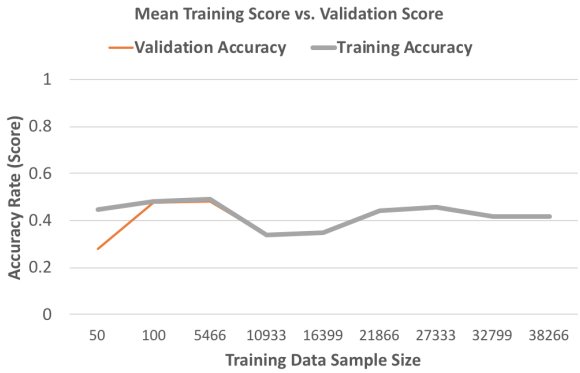


(a) kNN k = 10

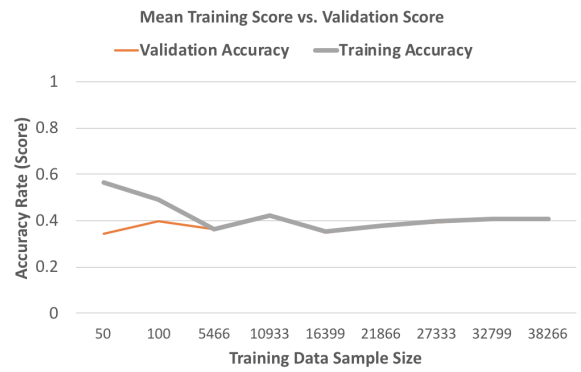


(b) kNN k = 1

Figure 7: Hand Posture kNN: Training vs. Validation Accuracy w/ Varying Training Set Size



(a) SVM: alpha = 1

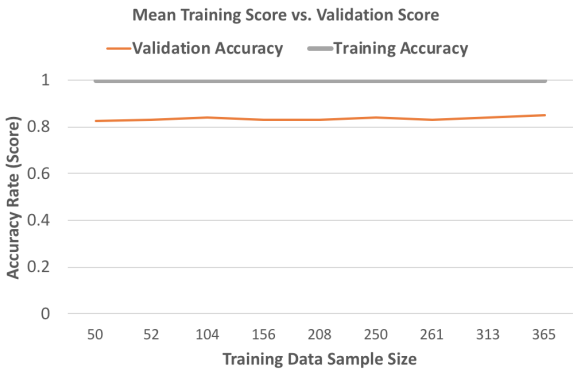


(b) SVM: alpha = 0.0001

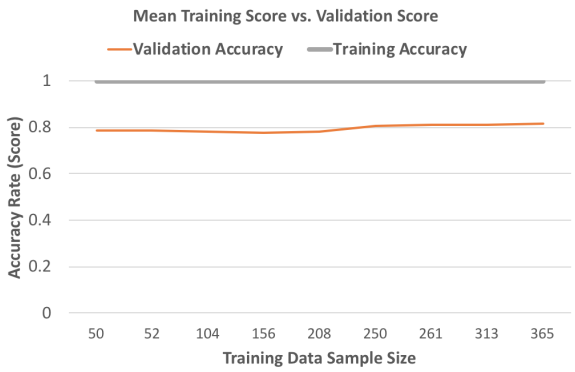
Figure 8: Hand Posture SVM: Training vs. Validation Accuracy w/ Varying Training Size

in misclassifications (lower the value, the "broader the street"). However, according to **??** even with an alpha of 0.0001, our results did not improve that much. On the other hand, our **kNN** model performed very well, and was fairly quick to converge around **98%** accuracy. Furthermore, our model was not sensitive to our specification of k, and $k = 1$ performed just as well as $k = 10$. This suggests our high dimensional data contains inherent "clusters" of classes. The figure in Figure 7 shows the results with the aforementioned specifications, but letting our k vary from 10 to the optimum $k = 1$ outputted via $GridSearchCV.best\_params\_$.

Credit Approval Data: The **kNN and SVM** models didn't work quite well for our small Credit Approval dataset, mainly due to lack of convergence. For one, it was hard to achieve convergence using **kNN**, suggesting overfitting. This result is contrary to the results we saw on the larger Hand Posture dataset (Figures 7). However, **SVM** on the smaller Credit Approval dataset performed better in terms of accuracy (80%) when compared to the larger Hand Posture dataset.

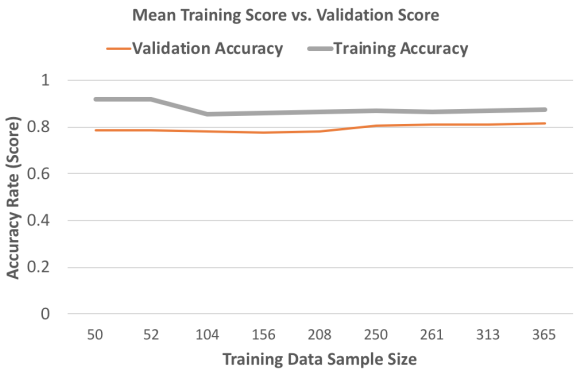


(a) kNN k = 10

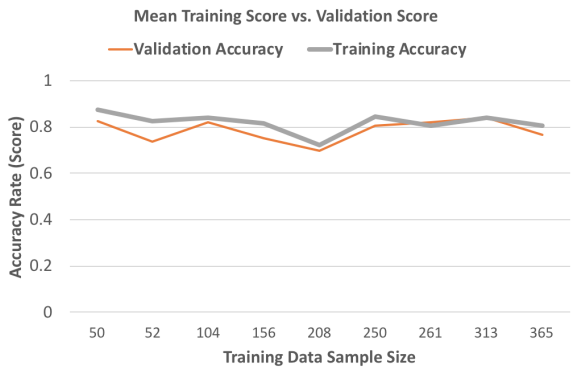


(b) kNN k = 1

Figure 9: Credit Approval kNN: Training vs. Validation Accuracy w/ Varying Training Set Size

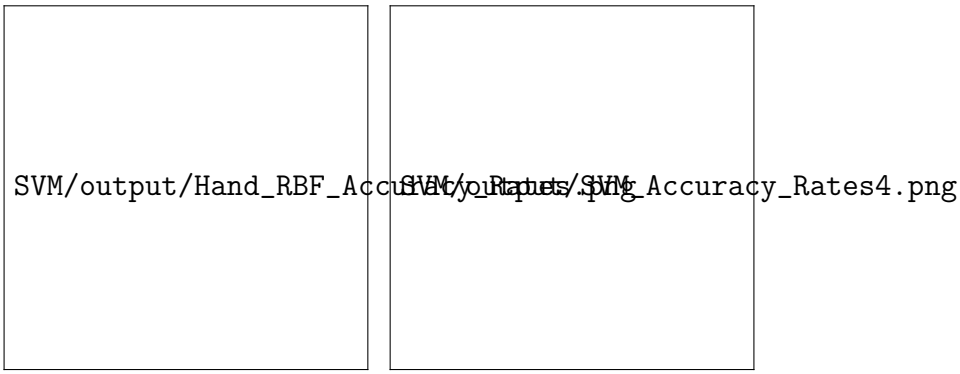


(a) SVM: alpha = 0.1



(b) SVM: alpha = 0.0001

Figure 10: Credit Approval SVM: Training vs. Validation Accuracy w/ Varying Training Size



(a) Gamma = 10, alpha = 0.001 (b) Gamma = 1000, alpha = 0.001

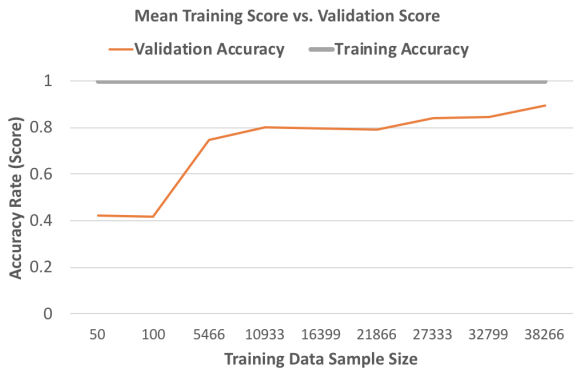
Figure 11: Learning Curves: Training vs. Validation Sets w/ Varying Learning Rate

3.2 Model Complexity Analysis

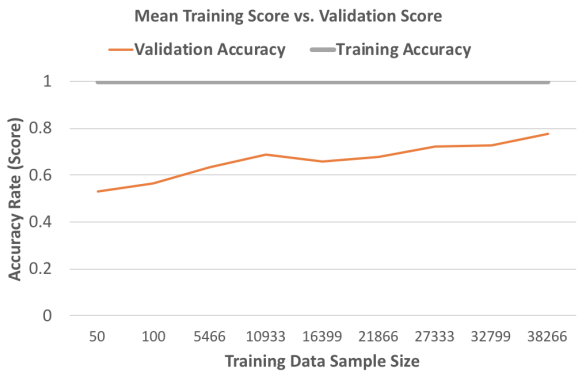
Hand Posture Data: We can resort to using an SVM with RBF kernel, instead of a linear SVM to dissect our high dimensional data. However, this comes at the expense of **very** large computation times. Figure **??** shows a reasonably well accuracy, with hyperparameters gamma = 10, and alpha = 0.001. The mere presence of an RBF kernel drastically improved our model, and adjusting how much each data point contributes to the formation of a decision boundry (gamma) allows us to squeeze out more accurate classifications, as shown in Figure **??**.

We can add complexity by altering the distance measure (between Euclidean & Manhattan) as well as adjusting our treatment of distance weights of training data. By "discounting" training data points that are farther away from the query data point, we are effectively prioritizing the classification of the query point based on proximity, rather than prioritizing solely based on aggregate average classification of **all** training points (which is the case in the "uniform" distance weighting scenario).

Using the figure 7(a) as a reference (which uses Uniform Distancing), you can see a subtle, yet important change in the accuracy of our validation set when using **Weighted Distance** as a hyper-parameter. Namely, figure 12(a) shows decreased performance for almost every training set sample size, reaching a maximum validation accuracy rate of 88%. Uniform distancing provides slightly better performance, as opposed to discounting data points further away from a test point's nearest neighbor. Logically, this makes sense- a single instance of a hand gesture is subject to measurement bias. By discounting all the other finger marker measurements, which also was subjected to measurement bias, we are not letting the model take this type of error into account. Instead, the distance weighted model is simply placing a higher emphasis on the closest points to make an assessment of hand posture- a misplaced emphasis which most likely took outlier data points from other hand gesture readings, and misclassified our query point as such. Lastly, it is worth noting that no combination of parameter + hyper-parameter tuning was able to correct the overfitting issue. My initial hypothesis was to blow up k to 5000 as shown in 12(b), but it didn't do

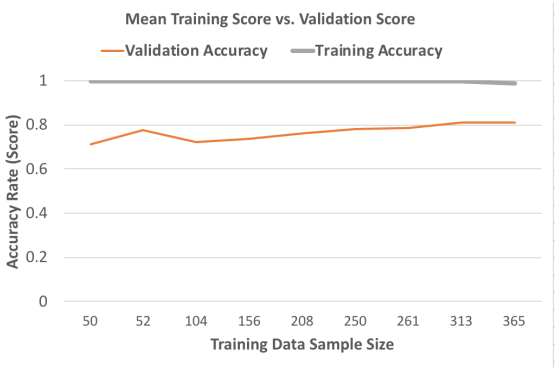


(a) kNN w/ Manhattan, Weighted Distance



(b) kNN w/ Manhattan, Weighted Distance (k=5000)

Figure 12: Complexity Curves: Training vs. Validation Accuracy w/ Varying k



(a) Gamma = 0.1, alpha = 0.001

SVM/output/CC_RBF_Accuracy_Rates2.png

(b) Gamma = 1000, alpha = 0.001

Figure 13: Learning Curves: Training vs. Validation Sets w/ Varying Learning Rate

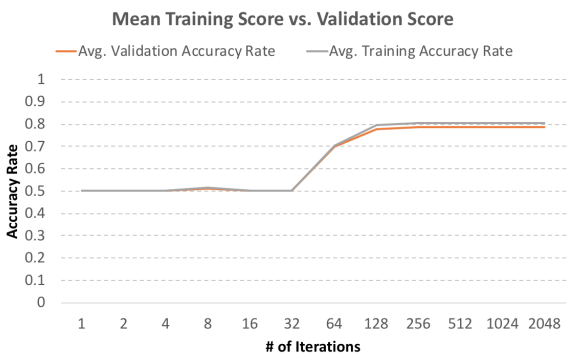
much- it just nearly blew up my laptop. In fact, it ended up doing worse.

Credit Approval Data: Compared to the larger dataset, we were not able to achieve convergence on the smaller Credit Approval dataset using **SVM w/ RBF Kernel**. Looking at figure **??**, we can clearly see the overfitting.

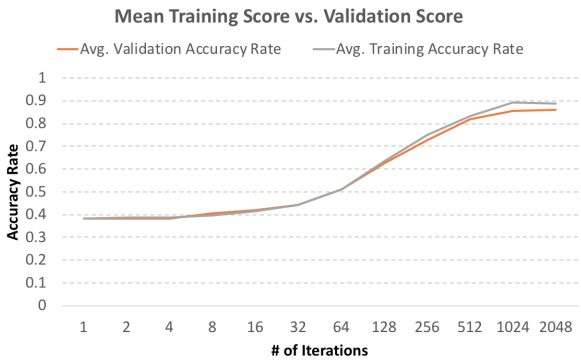
4 Artificial Neural Networks

4.1 Learning Curves Analysis

Credit Approval Data We can tune our parameters by increasing both the depth, and size of our hidden layers. Looking at figure 15, you can see that our ANN model with 3 hidden layers, each of size 30 performs slightly better at lower iterations, but worse at higher number of iterations, as compared to specifying 1 hidden layer of size 1 node. The overall



(a) ANN: Credit Card Data
3 Hidden Layers, 30 Nodes Each



(b) [ANN: Credit Card Data
1 Hidden Layer, 1 Nodes

Figure 14: ANN Credit Approval Complexity Curves: Testing vs. Validation Sets w/ Varying Depth and Number of Hidden Layer Nodes

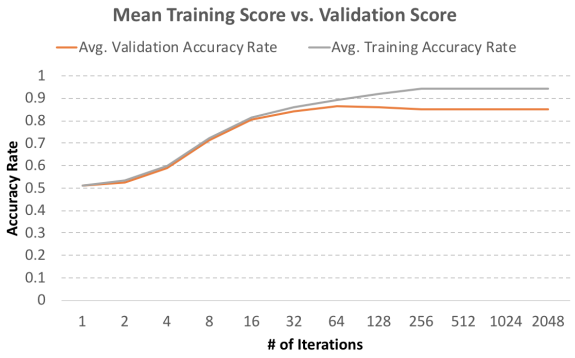
conclusion here is that the simple model with 1 hidden layer using 1 node actually performs better than one with more hidden layers at high number of iterations.

The first figure in 15 produces very promising results. With 30 nodes in each of the 3 Hidden Layers, we can only achieve 80% fit in the training set, and 80% in the test set no matter the number of weight-adjustment iterations (backpropogation). The performance is quite poor all around. The right graph uses the *basicResults.best_params_* method made available in scikitlearn to choose the best parameter for our ANN. The best parameters are: (1) L2 Penalty: 0.00316 (2) Hidden Layer Depth: 1 (3) Hidden Layer Size: 1 (4) Activation Function: RELU. Note that the figure on the right suggests we can reach a generalizable and accurate (88% test accuracy rate with 90% training accuracy rate) model with anywhere between 1,024 to 2048 weight-adjustment iterations. Figure 13(b) also highlights the importance of back-propogation. Giving our model time to make iterative adjustments to the weights drastically improves the model.

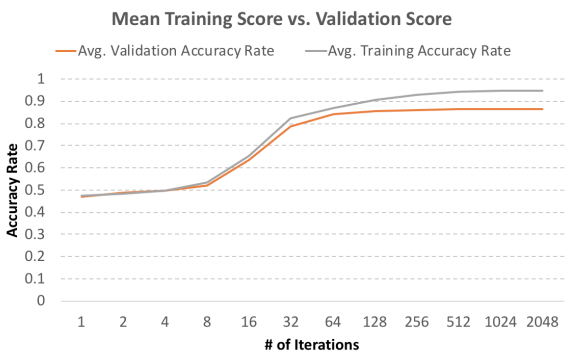
4.2 Model Complexity Analysis

Credit Approval Data To get a basic understanding of how our model performed, we first start off with a "basic" neural net with the following specifications: (1) 1 Hidden Layer of size 30. (2) Activation Function = 'relu' (rectified linear unit function, returns f(x) = max(0,x)) (3) L2 Regularization Alpha = 0. The figure in Figure 14 shows the results with the aforementioned specifications, but letting our L2 Regularization/Penalty term to vary between 0 & 10. Recall that the L2 Norm is essentially a squared error specification, and by it's nature, penalizes weights in a manner to reduce the likelihood of fitting the noise of the training data.

Hand Posture Data My preconceived notion was that a larger dataset would fare better using ANN, since more features and more instances should allow for better opportunities to classify. But this turned out not to be the case. With many different combinations of

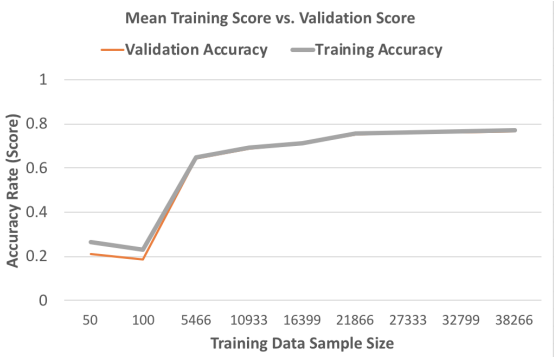


(a) L2 Regularization = 0

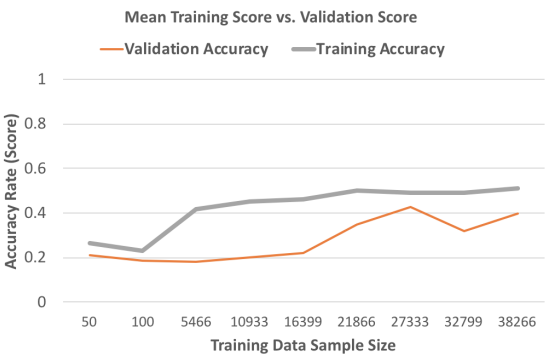


(b) L2 Regularization = 10

Figure 15: Learning Curves: Training vs. Validation Sets w/ Varying Alpha (L2-Norm)



(a) ANN: Hand Posture Data
1 Hidden Layer, 50 Nodes, Adaptive LR, 0.1 Alpha



(b) [ANN: Hand Posture Data
1 Hidden Layer, 50 Nodes, Constant LR, 0.1 Alpha

Figure 16: ANN Hand Posture Complexity Curves: Testing vs. Validation Sets w/ Constant + Adaptive Learning Rate Behavior

tuning, the best validation score I was able to achieve was 78% as shown in Figure 15(a). I didn't get reasonable results until I switched the Learning Rate hyperparameter from 'Constant' to 'Adaptive,' which suggests a finer tuning of alpha was necessary. Given this result, I changed alpha to 0.0001, but counterintuitively, that caused my results to again suffer. Eventually, I arrived at a 0.1 alpha specification (with adaptive learning rates) to achieve 78%.