Datasets

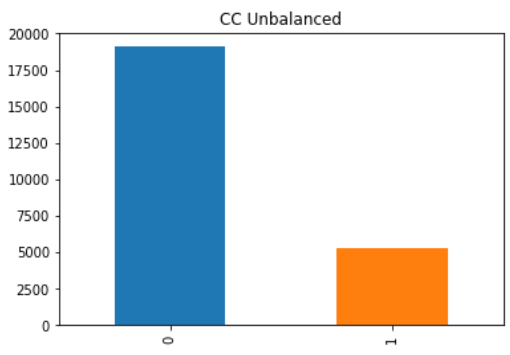
The two datasets used in this report are the Default of Credit Card Clients (CC) dataset from the UC Irvine Machine Learning Repository and the Challenges in Representation Learning: Facial Expression Recognition Challenge (FER) dataset from Kaggle.

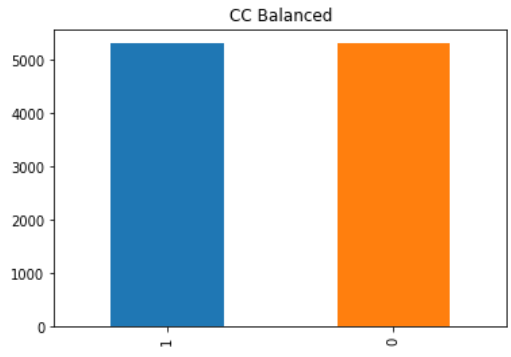
The CC data is the payment history, personal information, and whether they defaulted of clients for Taiwan banks in 2005. It contains 24,393 instances with 24 attributes each. The attributes include line of credit, gender, education, marital status, and age, as well as past delays, bills, and payments from April to September 2005 with each month and respective category being an attribute. The classes for CC were 1 (defaulted on payment) and 0 (did not default), making this dataset a binary classification problem. The FER data consists of 35890 different 48x48 pixel grayscale images of centered faces with different facial expressions classified as one of seven emotions: anger (0), disgust (1), fear (2), happiness (3), sadness (4), surprise (5), and just neutral (6). These categories were the labels, making this data a multiclass classification problem. The attributes were each individual pixel of the image, ordered from top to bottom and left to right, creating 2304 attributes for each of the over 35000 images.

These datasets are interesting because they do not concern themselves with trivial problems. Predicting who will default on a credit card payment based on payment history and identifying what emotion a face is making may appear to be relatively simple tasks for a human, but not for a machine that has to wade through many different attributes to decipher between the signal and the noise without any existing heuristic. The CC dataset has many attributes that may hold little significance, such as gender and months that have little significance. The FER dataset differs greatly from the CC dataset, since it suffers from the curse of dimensionality with almost as many instances as there are features and several labels for the supervised learning algorithms to classify with as opposed to just two. While it is already expected that the modeling will be more successful on the CC data as opposed to FER, the performance between different algorithms can still yield interesting results.

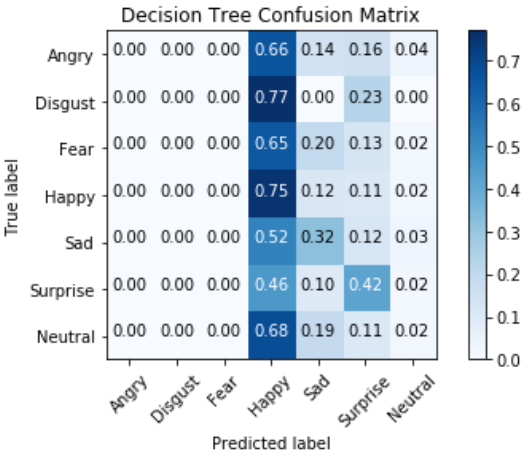
Preprocessing

Before performing any training, some preprocessing had to occur. Extraneous columns from the CC dataset were removed, as well as instances with missing values and undefined encodings for certain features, such as a ‘3’ for marriage or ‘7’ for education. This left 24,393 instances in the dataset. Because only 25% of the data had clients who defaulted, random undersampling was performed so that an equal amount of instances defaulted, as shown by the bar charts to the right. Oversampling was ruled out because it would require the defaulted instances to be tripled with repeated, averaged data from those instances, which can cause overfitting, which is possible when most of the labeled instances would have been essentially copies of the original set. Undersampling did not have this issue of overfitting, so only 25% of the non-defaults randomly chosen remained. Size of the set was also not a worry as there were still 10616 instances to be trained on, since only about 5000 were from class.

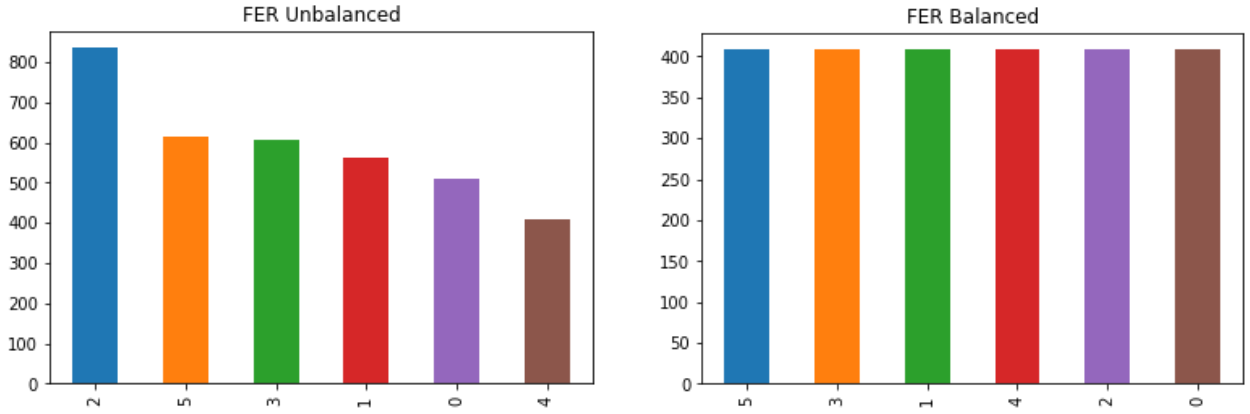




There may be some concerns that the training data reflects the real world without the original skew. However, it is hard to say what the reality reflects, since most clients are preselected for loans by the bank before being given a line of credit. This naturally skews the data, such that less clients will default. If this model is being used to replace the replace the preselection process, then using the skewed dataset might not be wise. However, one would be hard pressed to find data of people who default on loans that was not collected by a bank filtering out too risky of loans in the first place. In this case, the data was perfectly balanced so that the model would give no weight to the general probability of a credit card default being less likely to occur.



Preprocessing was also performed on the FER dataset due to even more severe imbalance. Because only 500 pictures were labeled disgust, which is relatively small compared to the thousands for every other label, those instances were removed, leaving 35340 instances with 6 labels. The confusion matrix below shows cross validation results of a decision tree trained on this imbalanced data. As one can see, the first three labels are simply ignored and happiness is hypothesized because it had more representation in the training data. So the model settled on this local maximum of guessing happiness when it isn't the case. In order to combat this, the lowest frequency class was removed. There is still imbalance with more happy faces than fearful or sad, so random undersampling was performed again, taking the least common label (4=sadness) and making all labels equal to that. This still allowed 24480 images to be used, so the relative tradeoff of information loss and balancing was not as much of an issue.



The major trade-off was efficiency vs performance. Attempting to process over 20,000 images with over 2,000 pixels in jupyter notebook using scikit libraries without any extra computing resources was futile. Therefore, a tenth of the dataset was sampled in order to train learning algorithms in a reasonable amount of time. So the final FER dataset consists of 2448 balanced images (as shown in the bar chart to the right) with 2403 pixels as features and 6 emotions as labels. For both datasets, instances were split using the default train_test_split function in the scikit-learn library such that 75% of the data was in the training set for the models to train and use cross validation, and the remaining was for testing at the end. Both datasets also had a scaled feature counterpart as implemented by StandardScaler() in sklearn.preprocessing, which was used to train on neural networks, support vector machines, and k-nearest neighbors algorithms.

Algorithms

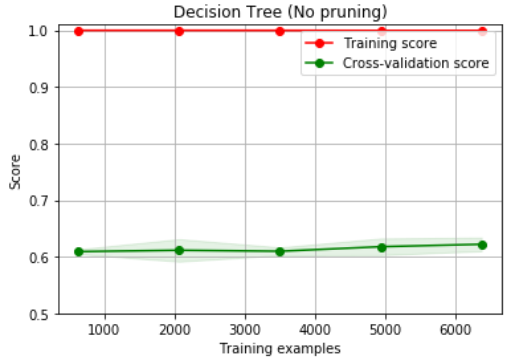
Five supervised learning algorithms (decision trees, neural networks, boosting, support vector machines, and k nearest neighbors) were used as models for each dataset. All were implemented in sci-kit learn libraries.

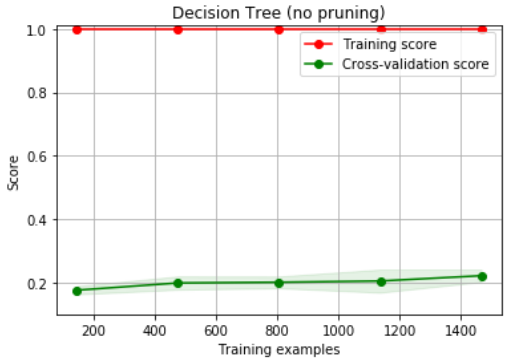
1. The default DecisionTreeClassifier is an optimization of the CART algorithm that stops splitting when no further information gain can be made.
2. For boosting decision trees, the AdaBoost ensemble classifier was implemented with decision stumps as its weak learners. Those decision stumps are the same decision tree classifiers as previously mentioned, except their maximum depth has been set to one.
3. The neural network used was the MLPClassifier, which is a multilayer perceptron that performs backpropagation. This classifier allowed for a lot of variation between layer architectures and optimizations which were tuned
4. The k nearest neighbors algorithm was implemented with KNeighborsClassifier, which automatically calculates distance by either BallTree, KDTree, or brute-force search algorithm based on fit.
5. svm.SVC() was the support vector classifier, which is a modified from libsvm.

Decision Tree Performance

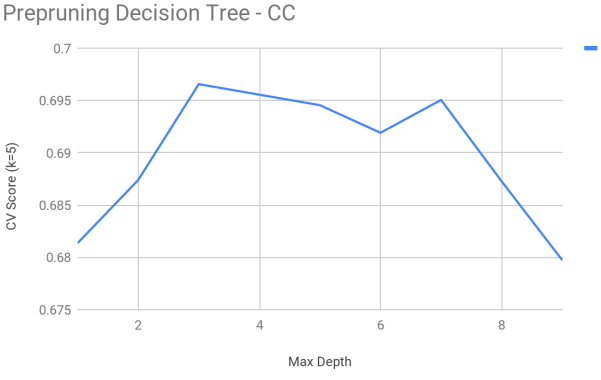
Training

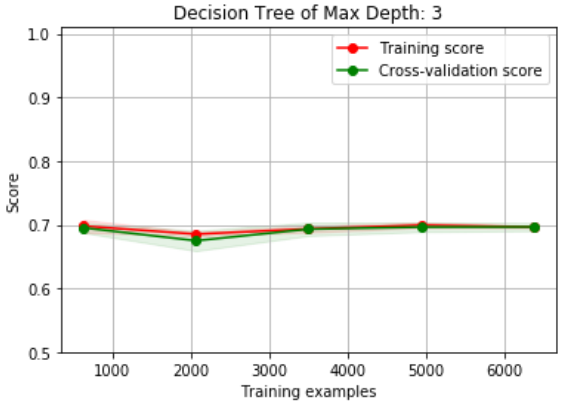
The initial decision tree was grown fully with no additional stopping criteria beyond no longer splitting and creating leaves when there is no more information gain. The decision tree overfitted to the training data, causing a low accuracy upon cross validation, as seen in both the CC dataset with 62.5% (on the left), and 20.7% for the FER dataset (to the right) below.

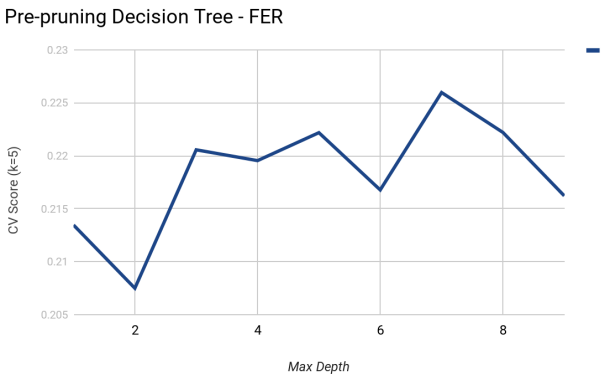


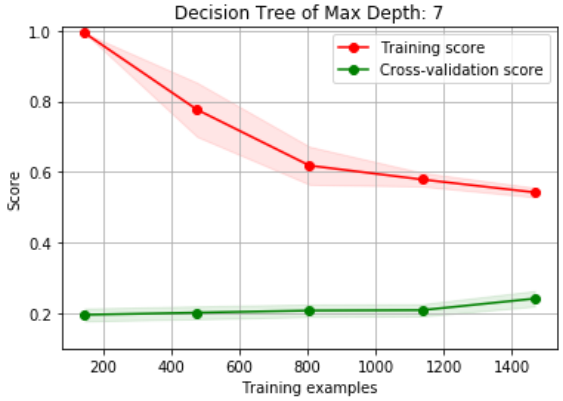


To improve cross validation accuracy, pre-pruning was performed by limiting the maximum depth of the decision tree and selecting one that had the best performance in cross validation based on the line graphs. The maximum depth of the tree regulates how many splits can be made based on the data. By limiting the depth, the decision tree can only make so many splits based on the training data until it no longer can. This allows for generalization because the tree can only perform splits on the features that hold most importance, as those features will be split first and have the highest gain. The less significant features which may just be noise are ignored since the decision tree has stopped training. This generalization allows for a more accurate tree that can be used on the test set, as shown in the cross validation results in the pre-pruned trees below.

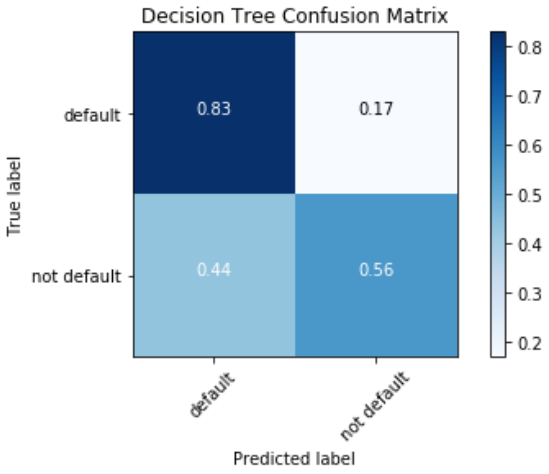


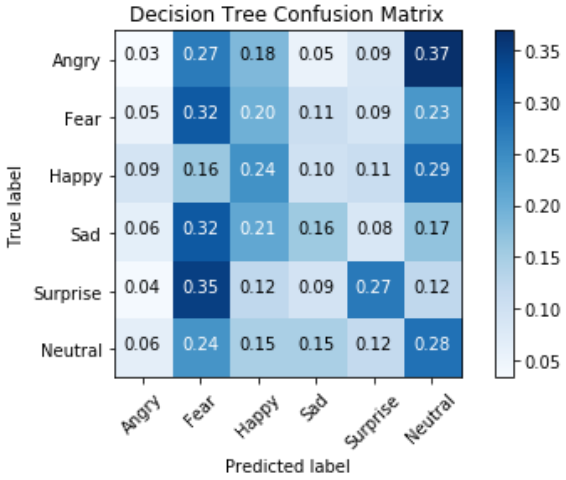






Results





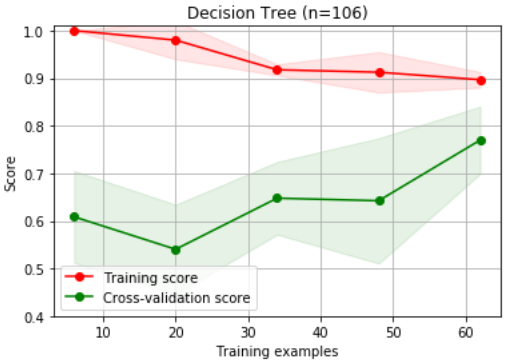
The CC Decision Tree (left) had an accuracy score of 0.69, labeling 69% of instances correctly in the test set. However, the confusion matrix shows a more complete picture of how the model functioned. The decision tree correctly predicted 83% of clients defaulting, and correctly predicted 56% of clients who did not default. So the model did reasonably well with predicting defaults but only did slightly better than random guessing for predicting who doesn't default.

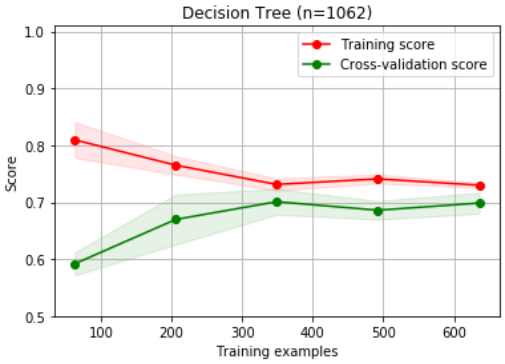
The FER Decision Tree (right) tells a similar story, with 23% accuracy, which is better than the 16% chance from random guessing since all 6 expressions are equally distributed. However, the model's predictions are not as equally distributed. The most common predictions are mistakes, with 37% of neutral predictions actually being fear and 35% of fear predictions actually being surprised. The most accurate predictions, fear and neutral, were popularly predicted on every other face. Only predicting surprise breaks this trend with most surprised predictions occurring on actually surprised faces. Sad and happy are also most popular predictions on their respective faces, but not to the same extent as surprise. Anger was neglected, with single digit predictions for each label. The decision tree appeared to arrive at a local minimum by over-classifying by a couple of labels and having the higher accuracies balance out the lower ones.

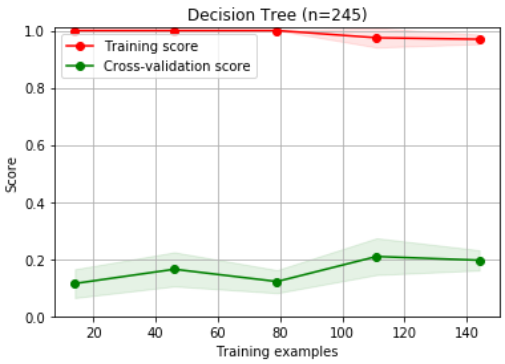
Sizes

For the CC dataset, sizes of the instances to 1/100th and 1/10th were modeled with cross validation. While having 106 instances is suboptimal, the medium dataset with about 1000 instances suggests that this is a sufficient amount to train on to yield similar results. It could be possible to balance the CC dataset and have around this many instances.

CC **FER**



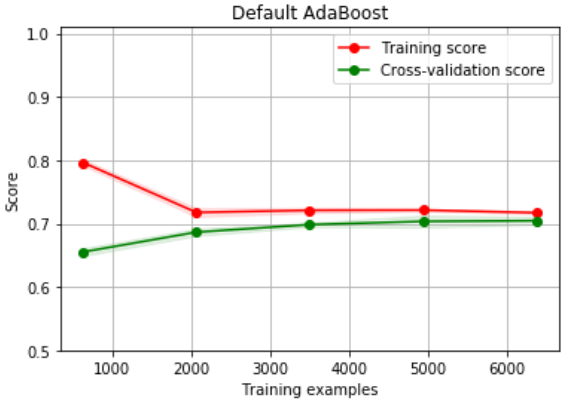


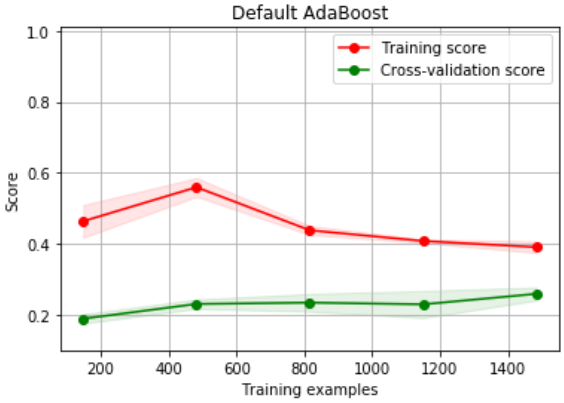


On the other hand, the FER dataset doesn't have this advantage, as observed above. At at tenth of the size of the used data, the cross validation does not bode well. Overfitting is clear even though the maximum depth has been limited just like the larger dataset. The decision tree needs more instances to generalize properly.

Boosting Performance

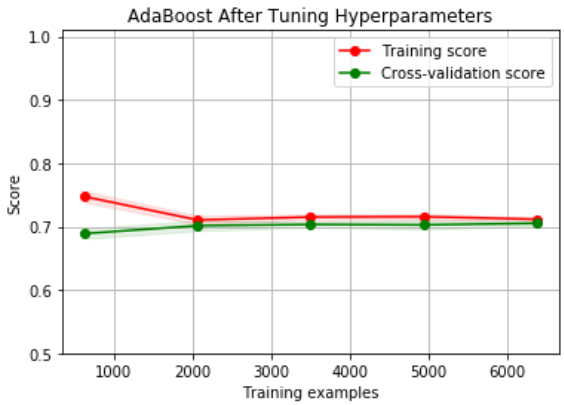
Training with Default AdaBoost Ensemble

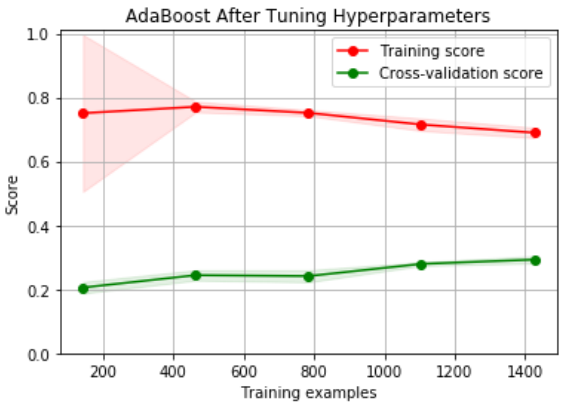




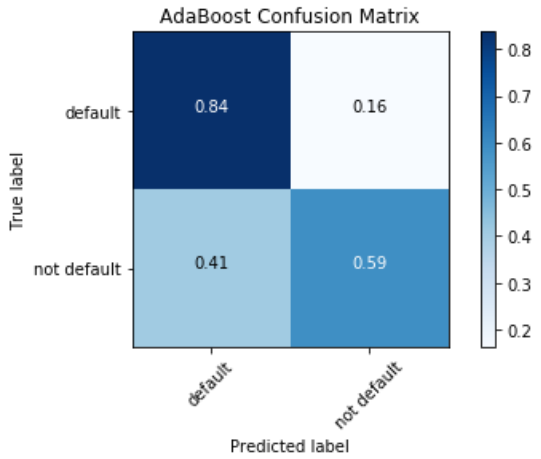
AdaBoostClassifier used 50 decision stumps with a learning rate of 1. This ensemble straightaway had a cross validation score of 70% on the CC dataset without any tuning (left). For the FER dataset (right), the validation score was 26%, similar to the cross validation for the pre-pruned decision tree**.**

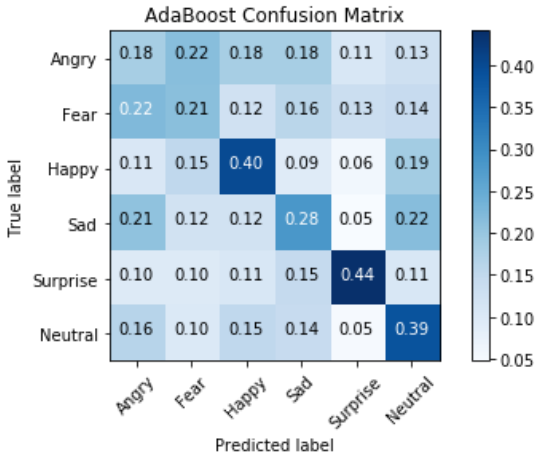
Hyperparameter Tuning The number of base estimators and learning rate was adjusted after applying a grid search for different magnitudes of learning rate and a range of estimators from 50 to 500. AdaBoost did not improve much on the CC dataset. Tuning the learning rate to 0.063, which was similar to the default, and using 310 estimators only caused the cross validation score to increase to 70.5%. It appears that increasing the decision stumps over 50 when there were only 24 features at most to base a decision had a minimal effect. For the FER dataset, the ideal learning rate was a whole order of magnitude higher at 0.316, and the number of decision stumps doubled to 710. This tuned classifier had a cross validation score to jump to 29.8%





Scores



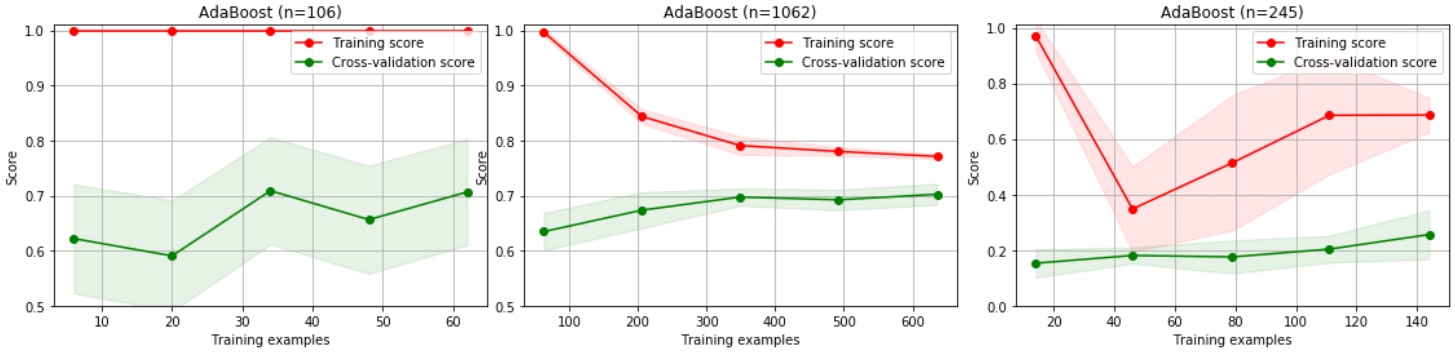


CC AdaBoost had an accuracy score of 71.5%, making it the most accurate supervised learning algorithm. The confusion matrix also demonstrates its predictive power, although it is a little less balanced than the neural network. The model tends to predict default labels more often, with 41% of non-defaults being classified as defaults.

FER AdaBoost was also relatively accurate having an accuracy score of 31.5%, but was also more balanced compared to other algorithms. Four of the six emotions had their most popular classifications be correct without dominating other emotions' incorrect classifications. Anger and fear were the exception, but still, their classifications were above random chance. Considering the features outnumbered the weak learners that could only make one split at a time, these results were impressive. The images had to have key pixels for the boosted trees to classify to this accuracy.

Sizes

CC **FER**



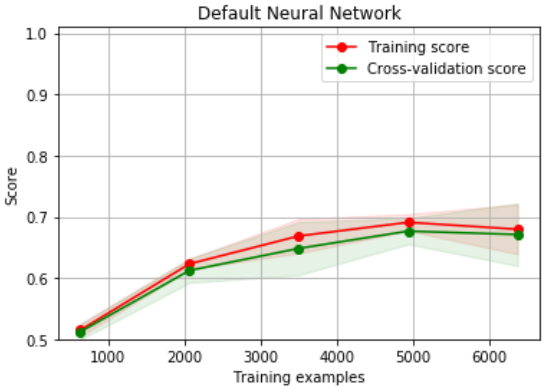
Like other classifiers, AdaBoost seems to perform similarly with over a thousand instances as opposed to just a couple hundred. With CC, the smaller dataset seems to overfit, which makes sense when there are more estimators than features or instances. With FER, overfitting does not seem to be an issue, as there is a similar performance to the larger dataset, as there are substantially more features than estimators.

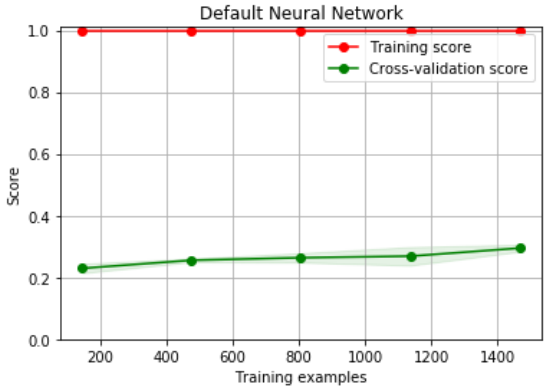
Neural Network Performance

Training with Default MLPClassifier

The initial neural network to test on the CC training data was an adam solver that had 12 nodes in one hidden layer, and a constant initial learning rate of 0.001 with 0.0001 for regularization. This network had a cross validation score of 66.2% as seen on the left, which was not bad but could be improved by tweaking the architecture of the neural network. The neural network for the FER training data was similar

to the CC model, except it was an lbfgs solver with 100 nodes in one hidden layer, which had a 29.6% score.

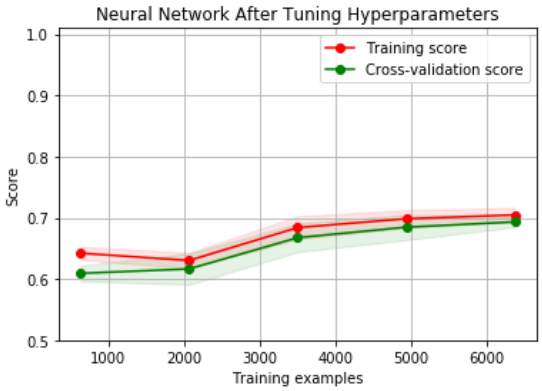




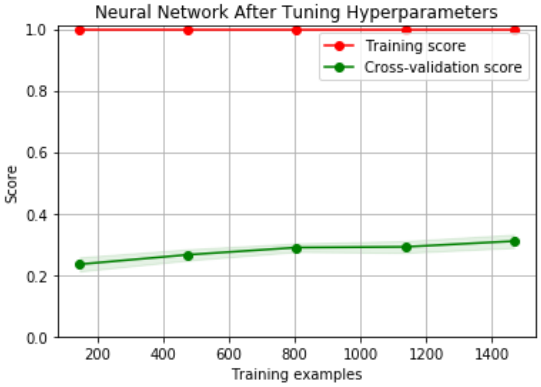
Hyperparameter Tuning

For the CC and FER dataset, three hyperparameters were tuned using an exhaustive grid search with 5 fold cross validation for each candidate. Three different weight optimization algorithms (Newtonian and stochastic gradient descent) were tested. Constant and adaptive learning rates were also tested, although that pertained more to stochastic gradient descent algorithms. Early stopping was employed so the neural network would stop iterating when the validation score did not increase. The alpha regularization parameters were tested from .1 to 1e-7 as recommended by scikit-learn libraries.

For the CC dataset, the number of neurons in the one hidden layer ranged from 12 to 62 in step sizes of 10. 12 was the mean of neuron units between the input and output layers, and 62 is a little over twice the size of the input layer. These numbers were used as they are rules of thumb for network architecture in Heaton's *Introduction to Neural Networks in Java.* Specifically, three of them were: use the mean between input and output, use ⅔ neurons of total between the outer two layers, and use less neurons than twice the size of the input layer. Establishing a range between these heuristics created the above hidden layer size to be tested. After 17 minutes of evaluating over 1000 fits (200 candidates with 5 folds of cross validation each), the best network was an Adam solver with a constant learning rate, 52 neurons in the hidden layer, and a higher regularization with an alpha of 0.01. This improved the cross validation score to 69.7% as shown on the right.

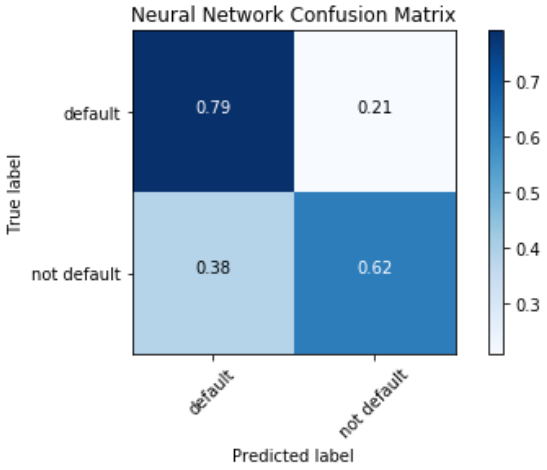


With the image dataset, a different strategy was employed to be more efficient. Hidden layers were optimized after the first three parameters were tested in an exhaustive grid search with 3 folds for cross validation. An lbfgs solver with a regularization of 0.1 scored highest. With these parameters, another grid search among 1 hidden-layer and 2 hidden-layer architectures was performed. Using Heaton's heuristics, ranges were searched from 200 to 4200 neurons for the first layer and 100 neurons for the second layer when applicable. After evaluating 30 combinations of the above for over 2 hours, the best architecture with a cross validation accuracy of 31.4% was surprisingly one hidden layer with 200 neurons. It is possible that 2 hidden layers was not enough to glean any complex information for the network to learn and the extra neurons just added noise.

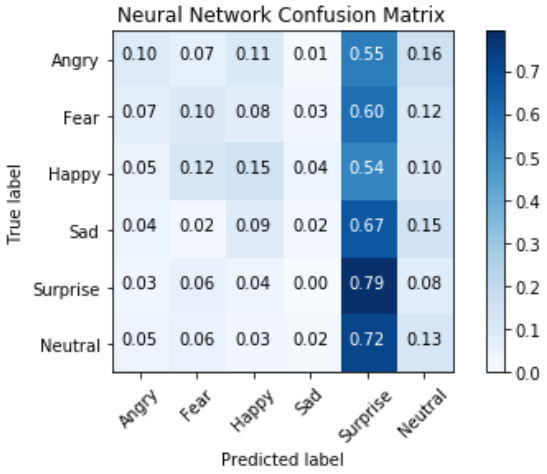


Scores

CC



FER

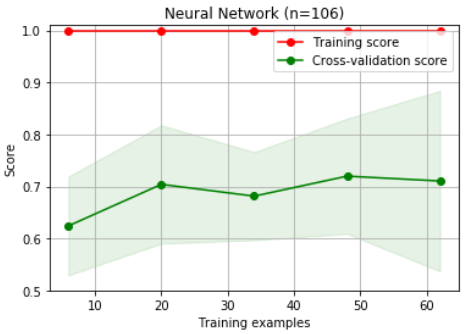


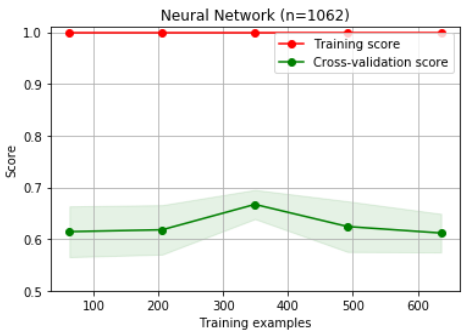
The CC Neural Network also had an accuracy score of 0.69, labeling 69% of instances correctly in the test set. On the surface this does not seem interesting, as it labeled the same amount of instances as the decision tree. However, the confusion matrix shows how the neural network performed better. The neural network correctly predicted 79% of clients defaulting, and correctly predicted 62% of clients who did not default. So the model did reasonably well with classifying defaults and better than the decision tree in classifying clients who did not default. Therefore, this neural network has more predictive power than the decision tree.

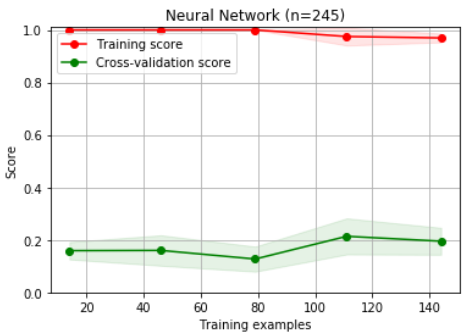
The FER Neural Network, on the other hand, does not have more predictive power than the decision tree. The MLPClassifier had an accuracy score of 0.27, which is technically better than the decision tree, but the confusion matrix shows that the classifier settled on some local minimum of predicting sadness for every single label. This neural network was very good at predicting sad for actually sad faces, with an accuracy of 79%, but it predicted sad on every other label to the same frequency. It appears that the cross validation scores from the hyperparameter search space may have been the wrong metric to use, as they do not take into account mislabeling or false positives like f1 scoring does. However, all of the labeled instances are the same in frequency, so there is no apparent reason for any particular label to be predicted more than the other.

Sizes

The neural network seems to perform better on a small set of data when looking at cross validation accuracy, but the overfitting is apparent with so few instances. The data needed to properly train a neural network for generalization requires more than a few hundred instances, especially with a lot of higher dimensions.







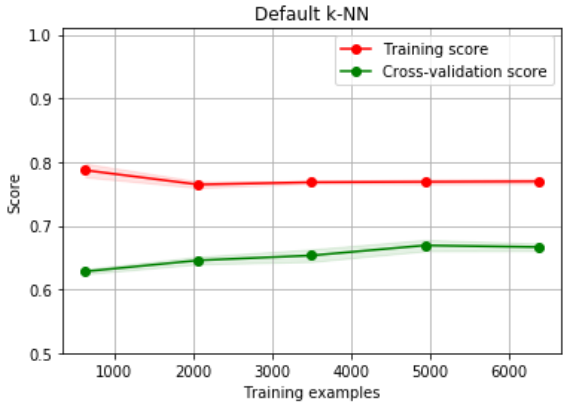
CC **FER**

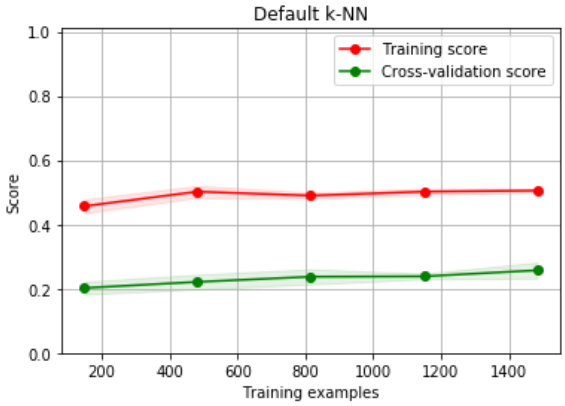
k-NN Performance

Training

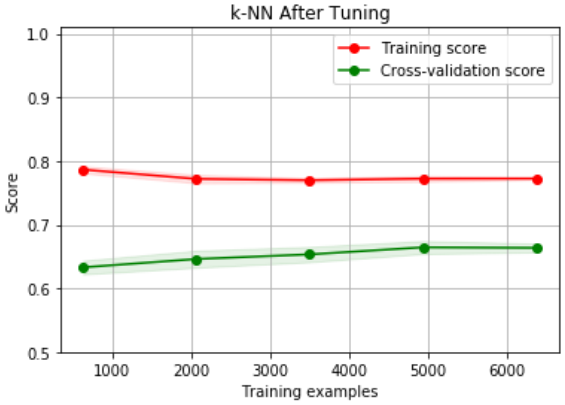
The default classifier uses 5 of the nearest neighbors calculated by Euclidean distance. Cross validation accuracies were 0.66 for CC (left), and 0.25 for FER (right).

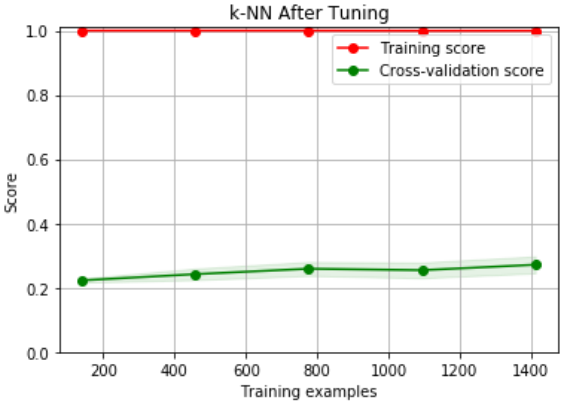
Hyperparameter Tuning



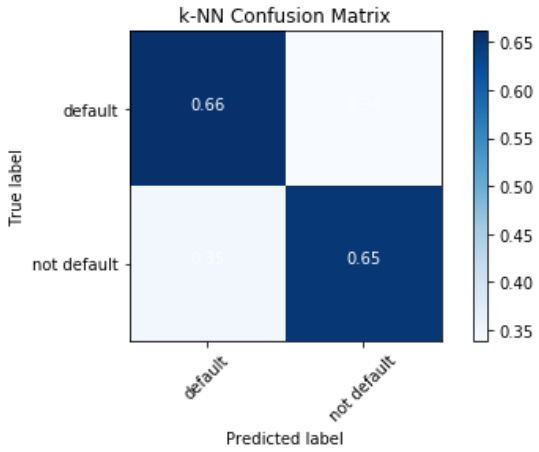


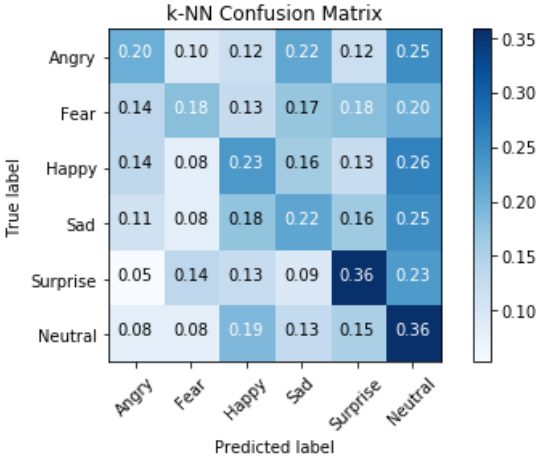
Number of neighbors, weighting of distances (uniform or by distance) and the distance metric (Manhattan and Euclidean) were grid searched for hyperparameter tuning. After seeing the results with the neural network, using a more balanced scoring metric was a point of interest. With that in mind, f1-score became the metric to use to evaluate the best estimator instead of cross validation accuracy. For the CC dataset, the highest f1 score ended up being a small variation of the default k-NN, with 5 neighbors, uniform weighting, and a Manhattan distance metric. For FER, 11 neighbors were considered by weighted Manhattan distance. Lower amounts of neighbors for both datasets resulted in higher accuracy, which makes sense considering that as the number of neighbors goes up, the likelihood of other classes being in the same radius of considered neighbors increases.





Scores





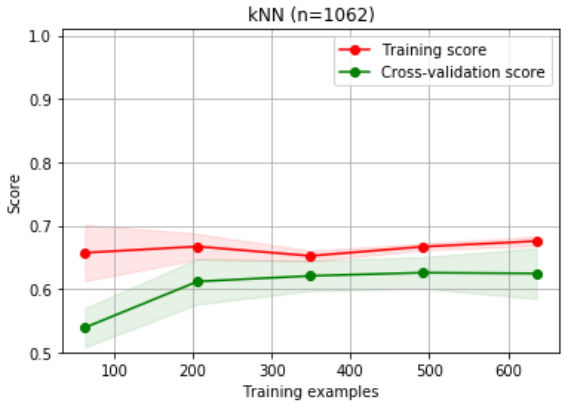
The use of f1-scoring when adjusting hyperparameters appeared to have a significant effect on the results of the model for both datasets. For CC dataset, the more balanced classifier compared to the other models had emerged, with both defaulting and not defaulting being classified to a similar accuracy of 66%. This resulted in a lower overall accuracy score compared to other models, but is the best one for classifying clients who do not default. For FER, which had an accuracy of 25%, each emotion has the most popular predicted label being itself. Neutral labels are over-classified incorrectly though, which may possibly be due to neutral faces having pixels similar to every other faces's pixels in certain parts of the image, such that those images are neighbors in an n-dimensional space.

Whether the results are indicative of the scoring metric or the actual model itself are muddled. To repeat this training and testing with this same set of data but with a different scoring metric would be cheating at this point, since we've already looked at the results on the test data. In the SVM model, cross validation accuracy will be resumed to evaluate and compare performance.

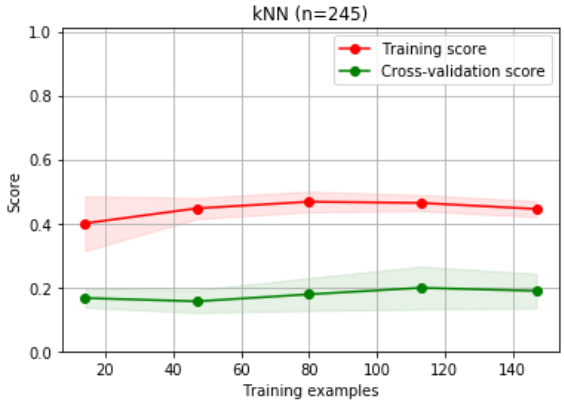
Sizes

k-NN appears to perform without overfitting at smaller sizes for both datasets, as their are still enough neighbors in the set for generalizability. The models still performed worse than the larger datasets these classifiers were originally fitted to.

CC



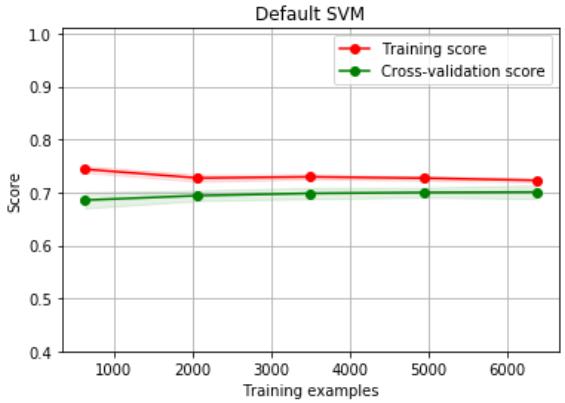
FER

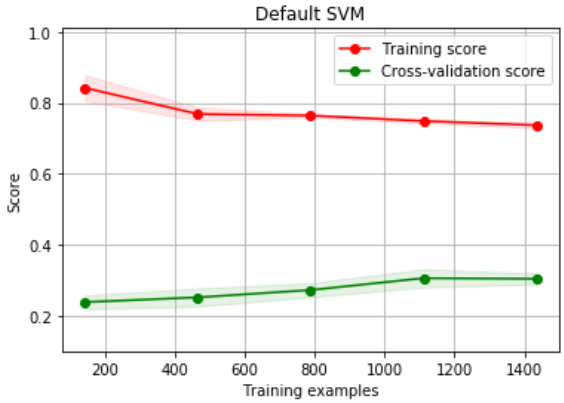


SVM Performance

Training with Default Support Vector Classifier

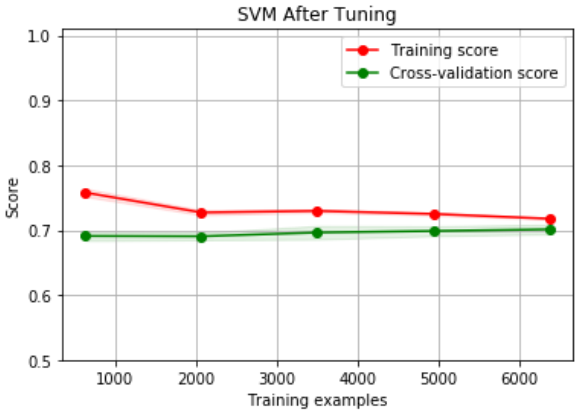
The default classifier used an rbf (radial based function) kernel with a C of 1 and a gamma of the inverse of the number of features, so 1/24 (approximately 0.04) for CC and 1/2403 (approximately 0.0004). With these values and this particular kernel, both models performed relatively well without tuning. The CC and FER datasets had a cross validation score of 70.2% and 30.6% respectively.

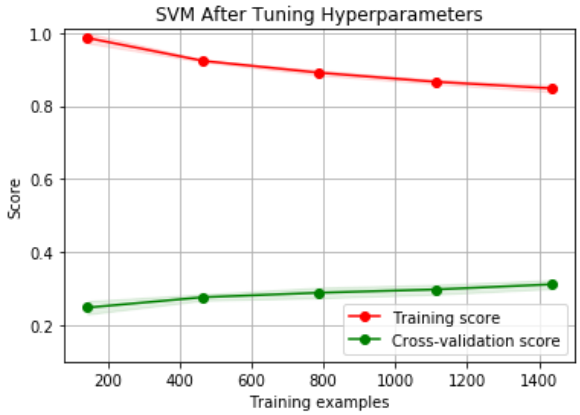




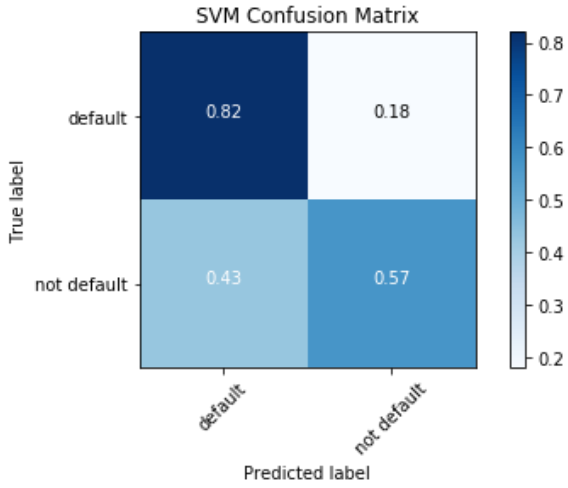
Hyperparameter Tuning

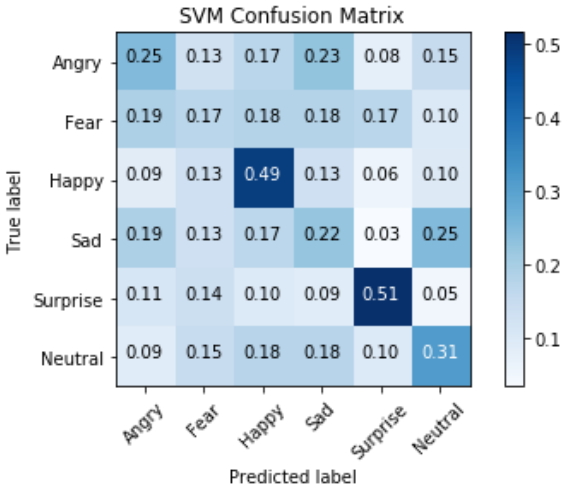
For the FC and gamma rates were adjusted for two different kernels: rbf and linear. The kernel rbf was best for both, so both datasets turned out not to be linearly separable. The initial search space around the C and gamma parameters varied by magnitudes, but not much improvement was realized as CC SVM's best estimator did not do better than the default and the improved FER scored at 31% with C=10 and gamma=.0001, which is similar to the original gamma.





Scores





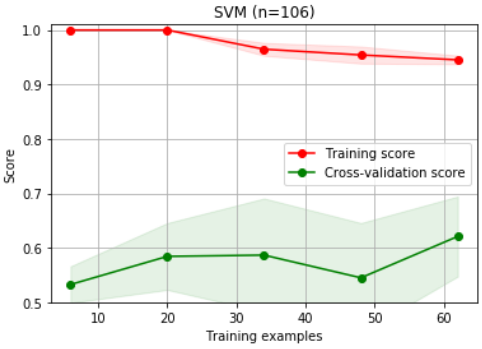
The CC SVM did not appear to perform any differently from the CC Decision Tree, with a score of 69%. Interestingly enough, the confusion matrix splits are almost identical, give or take a percentage point, despite taking much longer to train and attempt to optimize hyperparameters. This may be due to many of the features possibly being noise, which the decision tree never considered as its maximum depth was set to 3. Higher dimensional spaces are being considered by the SVM when only a couple of dimensions matter.

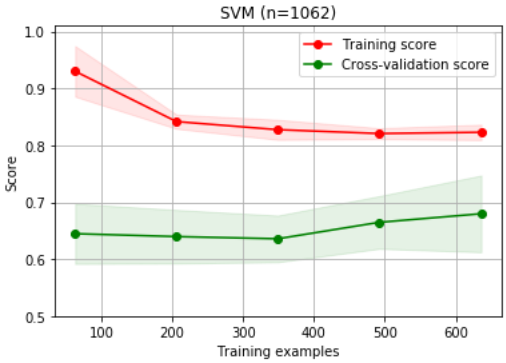
The FER SVM did not suffer from a similar curse of dimensionality, having a cross validation accuracy of 32%, making it one of the best and most balanced classifiers among the five algorithms. It performed best with three particular labels, correctly classifying 51% of surprised faces, 49% of happy faces, and 31% of neutral faces without dominating misclassification . It did not ignore any particular label, with each label being classified by at least random chance in classification. SVM appeared to have generalized well compared to other models thanks to higher dimensions.

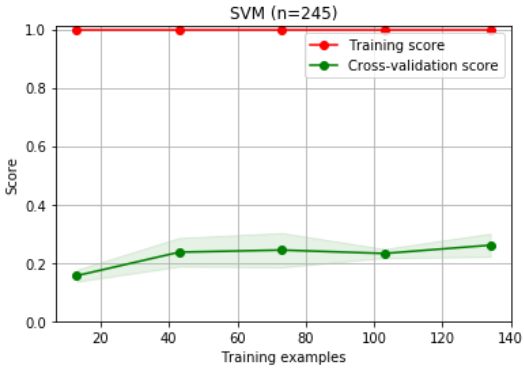
Sizes

The effect of size once again proves to be important with training classifiers. For FER, overfitting occurs when only a tenth of the original dataset is used. Although SVM performs better with higher dimensional feature spaces, this does not appear to be the case when the dimensions outnumber the instances by a magnitude of 10. With CC, this issue is seen to a lesser degree, as the dimension to instance ratio is nowhere near as high. In fact, CC SVM performs to a similar degree with a thousand instances and the smallest CC set has less overfitting compared to FER.

CC **FER**







Conclusion

The supervised learning algorithms were not perfect on either dataset. While they performed better than blind guessing, their accuracy in classifying credit card defaults or expressions was not particularly robust or inspiring. But the results were interesting as they revealed to some extent how these models learned from the data. Not everything under the hood comes to light from test data, but there were some key takeaways. A general trend, based on running the final models on smaller samples of the datasets, was that the more data the models had, the better they would do. With more instances to learn to decipher the signal and the noise from, the model improves in its classification accuracy.

Other pertinent points revolve around the optimization and scoring of the models and not the actual learning itself. For example, cross validation accuracy may not have been the best metric to measure how well the model actually classified. However, it is important to stay consistent with metrics when evaluating and tuning models, because the results of the test data could become compromised, as with k-NN. It becomes difficult to say with any confidence what the model did well and did not compared to other algorithms if they were not tested the same way. On another note, an exhaustive grid search that takes a long time does not guarantee that an optimal or even better tuning will be found. Using randomized search over smaller spaces may have been more efficient.

More specific takeaways about the learning algorithms are that a more complex model does not mean better results if the data is not similarly complex. A small decision tree can be just as accurate as a support vector machine, and due to Occam's razor, can probably tell you more about the significance of certain features in the data, as we saw with the credit card data. A complex architecture with hundreds of neurons does not guarantee that the neural network will converge on a model with any predictive power. It may just converge on a local minimum, as displayed by the performance of the MLPClassifier on the FER dataset. On a more positive note, support vector machines can generalize data with high dimensional feature spaces, and so can adaptive boosting, although it can also work with relatively simpler datasets. Overall, by applying these supervised learning algorithms on unique and different datasets highlighted many key differences in how these models learned. With further testing with different optimizations and algorithms, even more key insights can be gleaned in the future.