Machine Learning Project 1 Supervised Learning Report

February 10, 2019

0.0.1 Author: Neil Acharya

0.0.2 CS 4641

1 Dataset 1: The Pokémon Dataset

For my first classification problem, I chose to make a model that classifies a Pokémon based on its typing and its stat totals as to whether or not it can be considered "Legendary". For context, Pokémon is a videogame in which you capture, train, and battle imaginary creatures called Pocket Monster (Pokémon for short) and there are more than 700 unique types of pokémon you can find in the wild. These differences between species can not only be seen visually in their physical design, but also in intangables like their movesets, typing, Attack rating, and other statistics that play an important role when fighting, but can't be seen on the surface level game. So for example, one Pokémon may have a very high Attack stat but be weak when it comes to certain types of Pokémon, or another may have a very good Speed stat but have way to deal damage in their Attack stats. They can also be divided up by what game they were released it, often refered to as a generation by the community.

Even among these statistical differences, there is another group that Pokémon may be divided into called "Legendaries". Legendaries often have very high stat totals, or have access to a signature move that makes them stronger than others in a fight. The goal of my classifier is to categorize Pokémon into either Legendary or not based on the features provided in the dataset.

This dataset has 12 different features for each Pokémon entry. Specifically, each Pokémon has:

- **Number** The Pokédex number of that Pokémon. Can be considered the in-game indexing system.
- **Name** The Name of the Pokémon
- **Type 1** Type of that Pokémon
- **Type 2** Secondary type of the Pokémon, if applicable
- **Total** The sum of the Attack, Defense, Special Attack, Special Defense, Speed, and HP stats of the Pokémon
- **HP** Hit Points of that Pokémon
- **Attack** Attack stat of that Pokémon
- **Defense** Defense stat of that Pokémon

- **Special Attack** Special Attack of that Pokémon
- **Special Defense** Special Defense of that Pokémon
- **Speed** Speed of that Pokémon
- **Generation** Which generation it was released in
- **Legendary** A boolean entry that is true if the Pokémon is Legendary and False otherwise.

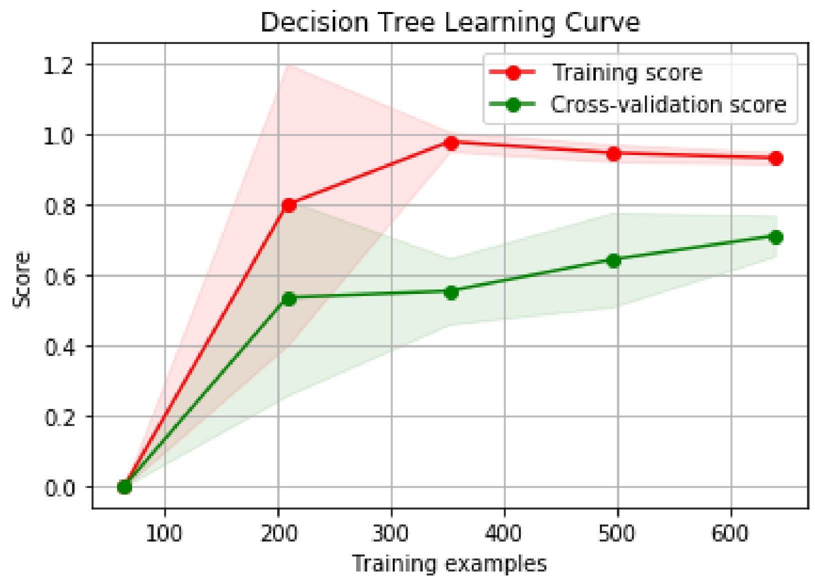
1.1 Data Cleaning

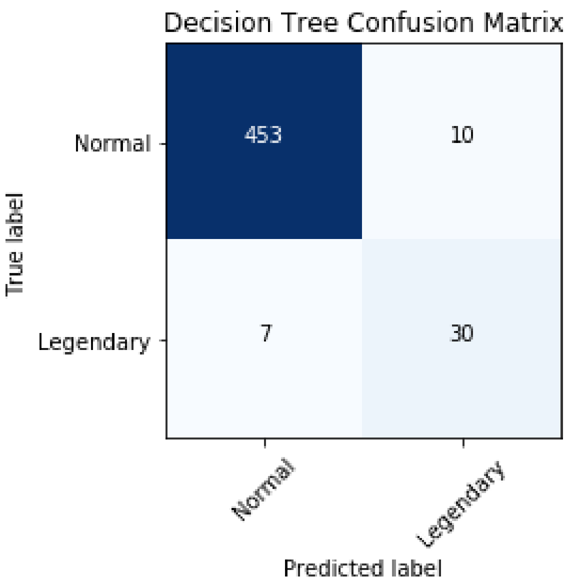
Part of the reason I chose this dataset is because almost all of the features in this are already formatted properly to be fed into a classifier. The one set of features I modified were the Type columns, where I concatenated the types into a single feature, then encoded those individual types so there were numeric values instead of the strings (i.e instead of "WaterFlying", it was encoded to 26). I also dropped "#", "Name", "Type 1", "Type 2", and "Generation" from the feature set we would be training on, since they weren't relevant features. Ultimately this meant I was training on 8 different features with 800 samples, split into training and testing at varying splits.

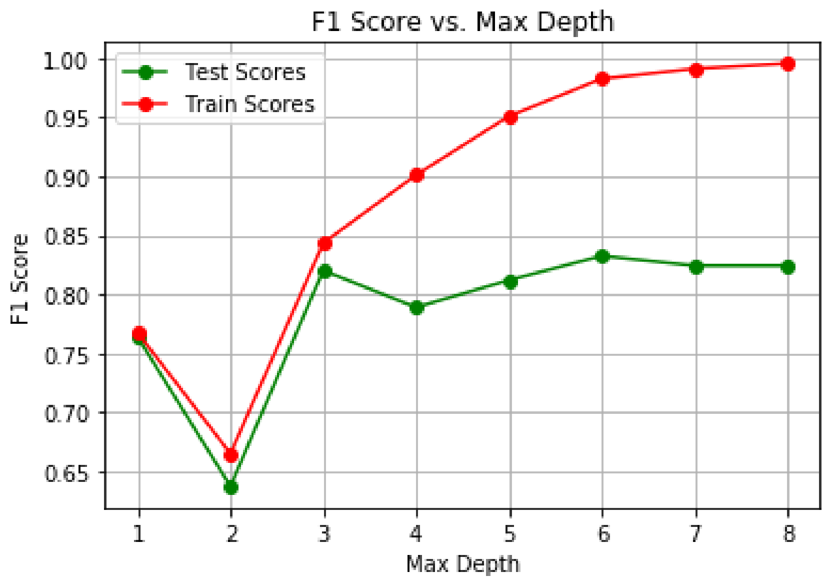
2 Results

For the results, each algorithm will be compared using their F1 score since there is an imbalance of positive and negative labels, making accuracy a not reliable metric. In this problem, the cost for False Positives and False Negatives are about equal; that is to say that whether we don't classify a legendary correctly, or classify a Pokémon that isn't a legendary as one are about equally bad. Therefore we use F1 since it takes both precision and recall into account.

2.1 Decision Tree

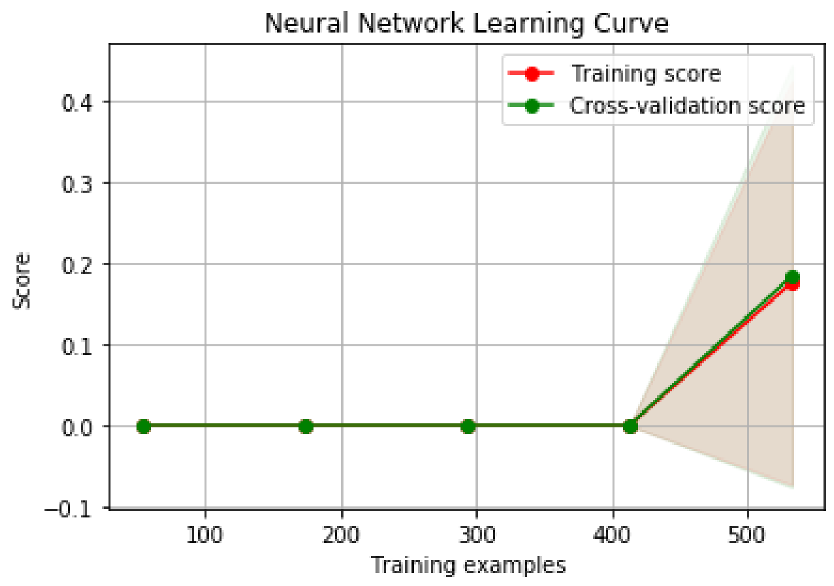


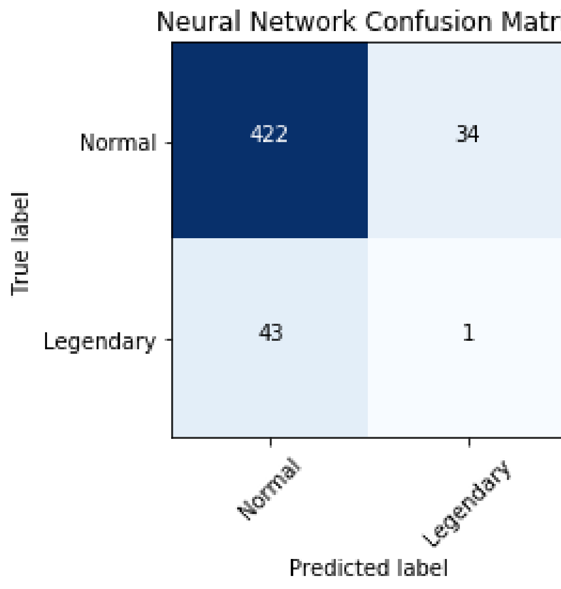




The average score of the pruned decision tree was **0.779**, which was the second best among all algorithms. The decision tree that performed the best was not a feature-length depth tree, but pruned using grid search cross-validation to be of **depth 6**. For the splitting, the default metric of "gini impurity" was used. The ideal training set size is around 300 examples which is approximately 3/8 of the full dataset. The confusion matrix indicates that the estimator classified most Legendaries correctly, but misclassified a small percentage of normal Pokémon as Legendaries.

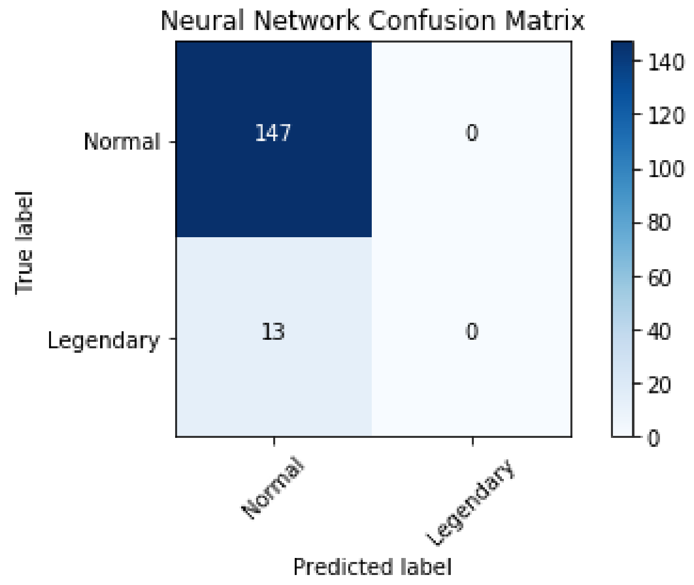
2.2 Neural Network



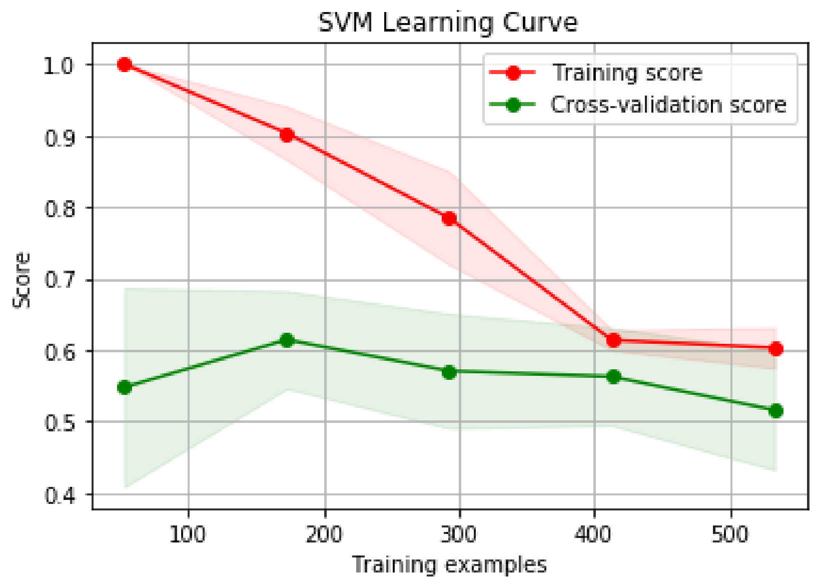


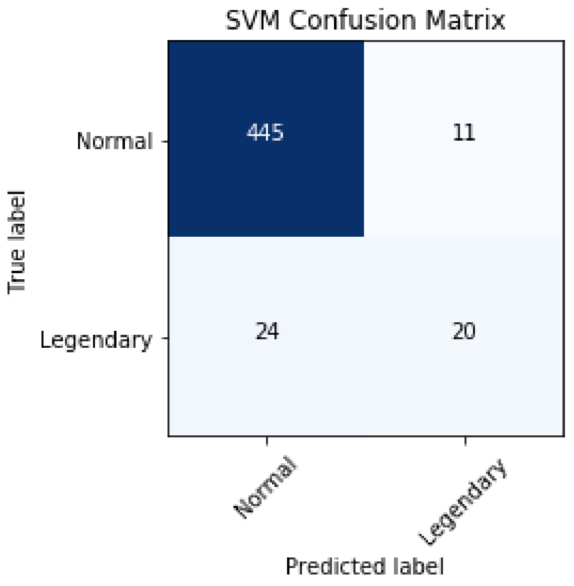
The average score of the initial neural network was **0.02531**, which was one of the worst among all algorithms. Using grid search to find the optimal hyperparameters again, this neural net had a **hidden layer size of 20 with 2 layers**, had a maximum iteration count of 800, and used the lbfgs solver. This estimator only classified one of the Legendaries correctly, while still having a similar number of false positives as the Decision Tree, ultimately hurting the f1 score a lot.

Based on the learning curve, we can see that overall, the low convergance of the training and validation scores mean the neural network needs a lot more sample data to improve than what is available; it only starts showing signs of improvement after 400 samples. Therefore, after looking at this learning curve, another model was made using a greater percentage of the data in the training split rather than the test split (80/20 split). The results from this new model are ultimately worse than before, as it generalizes all samples as "normal", as shown by this confusion matrix:



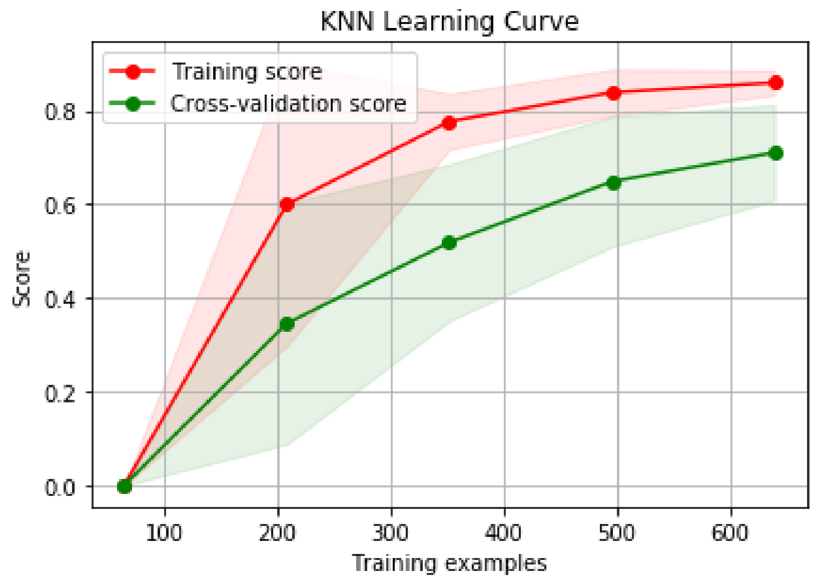
2.3 Support Vector Machine

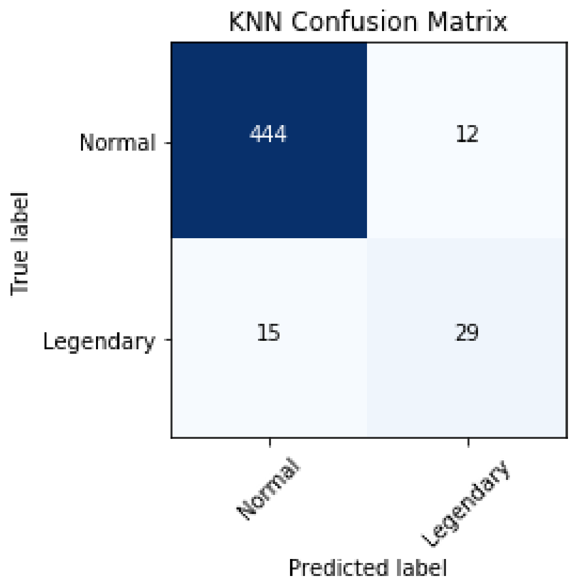




The average score of the SVM was **0.5333** which was the second worst among the algorithms. Using grid search, the optimal kernal was found to be the linear kernel, with the rbf kernel performing just as badly as the modified neural network, classifiying everything as "Normal". The learning curve of the SVM shows that with more examples, the training score and testing scores have a very high variance that slowly come together, indicating that more samples would likely help this learner. The confusion matrix shows approximately the same number of false positives as the decision tree, while it only classifies half of the true positives correctly.

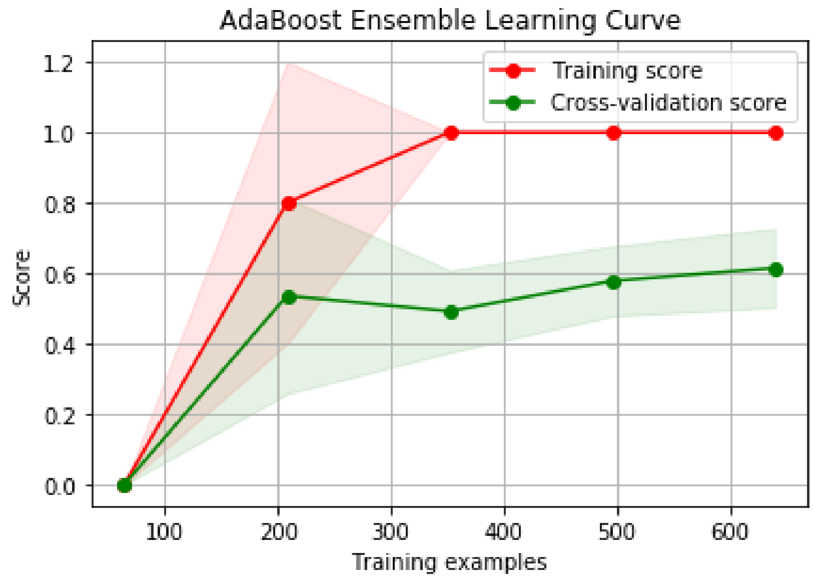
2.4 K-Nearest Neighbors

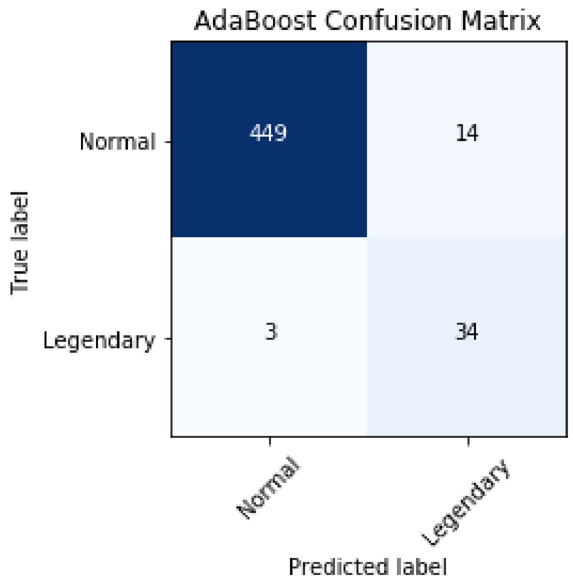




The average score of the k-NN was **0.638** which was the third best among the algorithms. Grid search found the best hyperparameters to be **k=3** and **weighting to be uniform**. The confusion matrix shows that the number of false positives is about the same as the decision tree, while it gets more false negatives, although not as much as the SVM. The learning curve most closely resembles the decision tree, with both achieving higher convergance than the rest.

2.5 AdaBoost Ensemble Learner





The average score of the AdaBoost Ensemble Learner was **0.80**, which was the best score among all the algorithms. As an ensemble learner, to get the best performance, a decision tree was used as the base estimator so this particular model can be thought of as a random forest. A decision tree was used as the base because it performed the best from the rest of the algorithms used. Grid search was used to find the optimal number of trees in this random forest, and was found to have **11 estimators**. The confusion matrix shows us that it only misclassifies 17 total, while getting almost all of the Legendary labels correct.

3 Analysis

Overall, this dataset proved to be interesting in comparing these different algorithms and shows the strengths and weaknesses of each of them respectively. Starting off, the fact that Decision Trees and Boosting worked the best highlight the power of simplicity and showcases Occum's Razor at work. With a feature set of only 8 different statistics, and only 800 data points to work with, the neural network and svm just couldn't compete with other learners that either rely less on the density of data. Since this isn't the type of problem where data augmentation is an option (only the creators of Pokemon can create more Pokemon), slow eager learners just don't work here.

I expected learners that extrapolate from their locality to perform well in this problem, since there tends to be a strong correlation between stat total and legendary status, and as expected, Boosting and K-nn both performed admirably. Interestingly enough, among all the confusion matrices, the number of false positives, where a Pokémon is label legendary but isn't, is astonishingly similar. This might be due to a mechanic introduced in later generations where "normal" Pokémon can gain access to a change in stats that raises their totals while still not being listed as legendary. This may be considered noise on part of the data, or a lacking classification class, but definitely attributed to the lower F1 score among all models. As such, in the future I would probably want to wither make this a multi-class classification problem, or change the metric from F1 to Recall, since False Negatives probably have more significance in hindsight.

Also, unquestionable, AdaBoost performed the best as per my definition of classifying the most legendaries correctly, and not misclassifying as many normal Pokémon.

4 Dataset 2: Handwritten Letter Recognition

Similar to MNIST, this dataset contains a bunch of metadata on images of handwritten letters for classification. Instead of actually having the pixel matrix, however, we are just given 16 different measurements to base the classification on. Dataset found here.

This problem differs from the Pokemon problem in the quantity of data available (20000 samples) and the number of classes to classify to (26 classes, A - Z). I predict that the slower learners will do a better job of learning with this dataset because of that difference in quantity.

This dataset has 17 columns, 1 column with the classification label, and 16 different features:

- **lettr** capital letter (26 values from A to Z)
- **x-box** horizontal position of box (integer)
- **y-box** vertical position of box (integer)
- **width** width of box (integer)
- **high** height of box (integer)
- **onpix** total # on pixels (integer)
- **x-bar** mean x of on pixels in box (integer)
- **y-bar** mean y of on pixels in box (integer)
- **x2bar** mean x variance (integer)
- **y2bar** mean y variance (integer)
- **xybar** mean x y correlation (integer)
- **x2ybr** mean of x * x * y (integer)
- **xy2br** mean of x * y * y (integer)
- **x-ege** mean edge count left to right (integer)
- **xegvy** correlation of x-ege with y (integer)
- **y-ege** mean edge count bottom to top (integer)
- **yegvx** correlation of y-ege with x (integer)

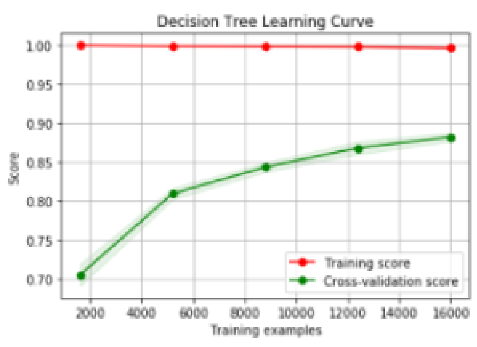
4.1 Data Cleaning

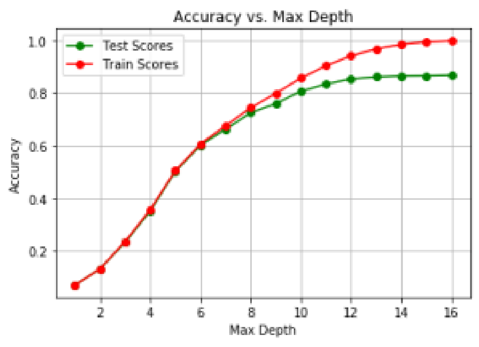
As with the previous problem, this dataset is in a structurally ideal format to be trained on as all the training feilds are continuous integer values. Similar to the Type encoding from before, the letters will be encoded to integer values so that the SVM can run classification. As a standard, the source documentation states that the data should be split 16000:400 into train and testing sets, so those numbers were used across all algorithms.

5 Results

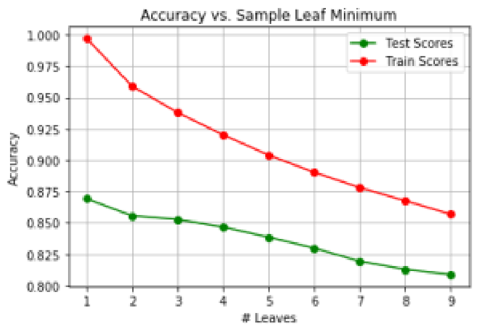
This time overall balanced accuracy was used as a metric as opposed to F1. Since this data set is relatively balanced in its classification frequency, a balanced or equally weighted metric is ideal since there is no preference for one class over another. False negatives and false positives don't matter as much either, meaning F1 score, precision, and recall all aren't necessary. Also, since there are 26 classes in the label set, confusion matrices will be left out of the figures since they would take up too much space. Instead, the analysis will cover the effects of tuning the different hyperparameters in each algorithm more closely.

5.1 Decision Tree

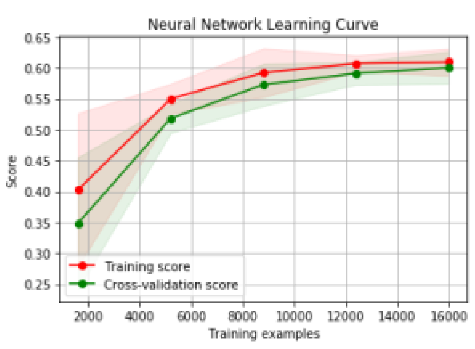


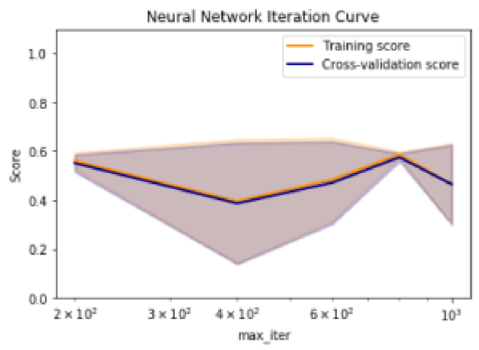


This time, the decision tree performed the second worst among all algorithms, with an accuracy of **0.87075**. This tree had a **max depth of 16** while using **entropy**, or information loss as its criterion for splitting. From the learning curve, we can see that even with 16000 samples, there is still room for the test set score to increase, meaning the decision tree may improve given more data, which actually mirrors the decision tree curve from the Pokemon problem. Where things differ however is in the max depth curve, in which this problem still has room to improve the test score with more depth when the Pokemon problem highlighted overfitting after depth level 6. Because of this, the decision tree could not be pruned based on depth. Instead, pruning was attempted based minimum samples needed to be considered a leaf node, but that also failed since this didn't converge either, as shown in this graph:



5.2 Neural Network

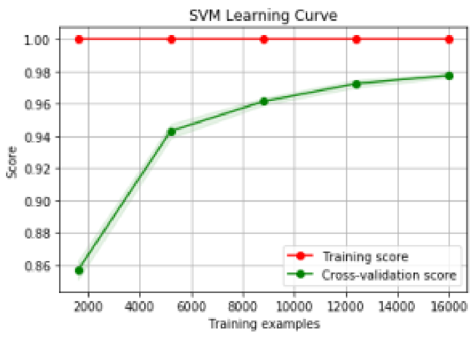


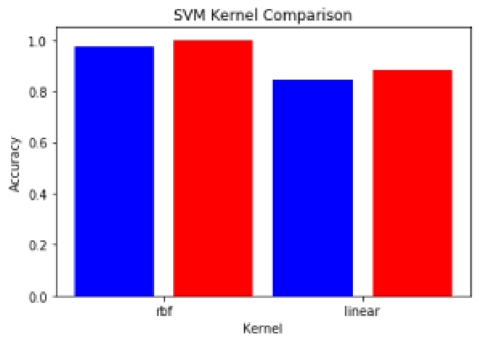


The neural network still performed the worst, with an accuracy of **0.555**. Grid search found the optimal parameters to be **"relu" activation function**, **hidden layer dimensions of (50,2)**, and the maximum iteration count to be **800**. Although these results are still the worst among the algorithms, the neural network actually demonstrates a good learning curve that seems to just approach its asymptotic training set behaviour will still avoiding overfitting. This means more data likely wouldn't improve its functionality.

The next chart graphs accuracy against the number of iterations the neural network runs. This chart is probably the highest variance graph of the dataset and demonstrates nothing of value, other than iteration count doesn't seem to impact performance, or that the numbers chosen weren't suitable in the scope of this problem.

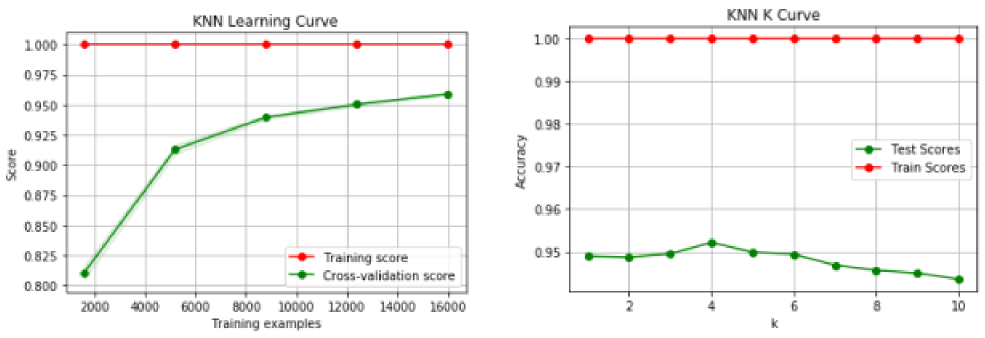
5.3 Suport Vector Machine





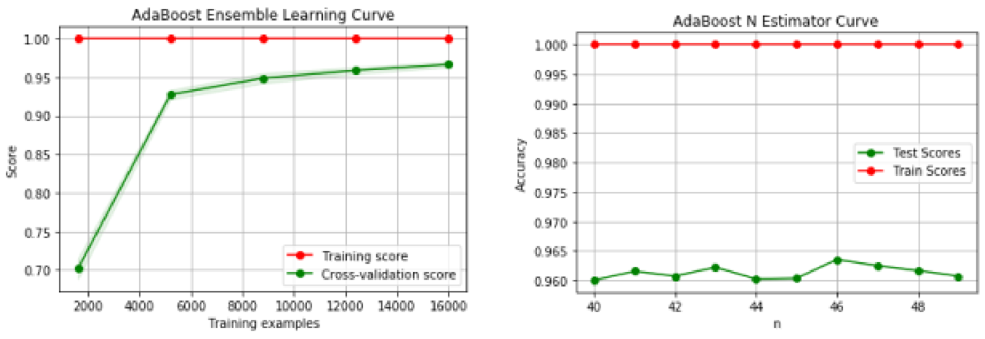
The SVM this time performed the best with an accuracy of **0.97425**, using **C=100** and the **rbf** kernel. The learning curve demonstrated relatively good convergence to the training error, and shows just how powerful SVMs can be given the proper support vectors to marginalize. The next chart compares the rbf kernel to the linear kernel and shows that although both kernels perform well, rbf is slightly better in both training score (red), and testing score(blue).

5.4 K Nearest Neighbors



The k-NN learner performed admirably, with an accuracy of **0.96075** on the test set. For this model, **k = 4** and the weights of each neigbor was based on **distance**. The learning curve also shows good convergence, similar to the SVM in that it reached a point of optimal performance before likely overfitting. This next graph shows how varying k affected the score of the model, where 4 was clearly the global maximum, and any more than 4 led to decreased accuracy. This hints at certain pieces of domain knowledge that we don't have as to why the 4 closest points tend to cluster.

5.5 AdaBoost Ensemble Learner



The average score of the AdaBoost Ensemble Learner was **0.9615**. Once again, we used a **decision tree as the base estimator**, each one with **depth 13**. Ultimately, **46 estimators** were used in this forest. The learning curve is another example of good convergence and demonstrates strong learning and good generalization without overfitting. The next graph compares different values of n for the number of parallel trees in the ensemble, which shows a soft peak on 46. It doesn't show the full range of n values from 1 to 50 due to the run time cost being so high for this algorithm.

6 Analysis

Compared to the first problem, this dataset provided much stronger accuracy scores due to the greater density of data, and and equal distribution of classes in training. I feel like this demonstrates the Curse of dimensionality to a certain degree, since the Pokémon dataset only had 8 features that I trained on with 800 examples. The letter dataset had 25 times as many samples and more balanced class distribution, which means there were almost as many samples per letter as there were in the entire Pokémon dataset. This also attributed to how different algorithms did better on these different problems. While the Pokémon dataset did better with Decision Trees due to the simpler nature of the problem, the letters did the best with an SVM which used a one-vs-one scheme to divide up the samples linearly in a more complex manner. We also saw the effect that having more samples had on the learning curves, as they were all unanimously "better" curves for the Letter problem, approaching good fit for all algorithms, where as Pokémon had some seemingly random or noisy curves that weren't easily interpretable.

Directly comparing each algorithm to each other raises some interesting observations and insights about each. One of the clearest details I learned through this experiment is that neural networks need a lot of parameter optimization compared to all other algorithms, which probably comes in the form of extensive domain knowledge. Where for the rest of the algorithms I could exhaustively search for the best hyperparameters, for neural networks, not only was that expensive in compute and time, but it also didn't always lead to optimal performance. For both of my problem sets, neural networks were not the proper tool to leverage the data available. Even with the significantly bigger set size of 20000, I believe neural networks need a significant amount of more data, and deeper domain knowledge to be useful.

A positive oberservation for these algorithms is the power of local learners and their ability to accurately generalize classification problems. Both KNN and AdaBoost can be considered local learners in that they both consider datapoints that surround a test sample moreso than learning from each individual sample to classify. Both of these algorithms tended to either outperform or match the best algorithms in both problems.

7 Conclusion

Each learning algorithm has its strength and weakness given the context of the problem (domain knowledge), the actual estimator the algorithms choose to design (their preference bias), and how "good" the data is for that problem. For example, my Pokémon dataset had a very limited sample space, and therefore did not perform well with Neural Nets, but because of the problem's simplicity, Decision Trees proved to be sufficient at solving it. Comparing it to the letter problem, where I had a lot more data taht was more balanced, the "local" learners tended to perform well. Overall, I can say that I am more accustomed to utilizing these algorithms as tools and am more familiar with when to apply each of them.