**Supervised Learning**

**Abstract**

This assignment explores various supervised learning algorithms by comparing and contrasting their properties while training on two different datasets. The algorithms will first be analyzed according to different hyperparameters in order to choose the best performing settings for each dataset, then the algorithms will be aggregated together to compare the performance on training/testing a similar dataset. There will be two datasets used for this assignment as described as follows:

* **MNIST Handwritten Digits:** This dataset is one of the most well known in data science. It consists of 60K training and 10K test examples and was created by LeCun et al. The supervised learning task associated with the data is: given an image of a handwritten digit, determine which digit the image represents. For this assignment, the data was split into columns representing the individual pixel values and put into csv files for easier computation. The distribution of digits is approximately even, providing the algorithms with an evenly distributed dataset to classify digits. The high number of features combined with the high number of samples provides the algorithms with a problem that tests how well they perform on large data.
* **Wine Quality:** This dataset was obtained from the public repository maintained by University of California Irvine. The data consist of 6497 measurements of the physicochemical properties of both red and white Portuguese “Vinho Verde” wine. Each wine sample has an associated quality rating, which was determined by taking the median of at least 3 evaluations made by wine experts. The dataset presents two important distinctions from MNIST: the larger presence of noise in the labeling and the smaller sample size. Since the evaluations were made by taking the median of human opinion, there exist some variation on the true quality of a wine sample, so the relationship between wine characteristics and rating may not be 1 to 1. The smaller sample size will also contribute to significant differences in performance compared to MNIST, since there will be less data to make decisions on.

**Methods**

All algorithms were trained on two datasets. The sets were split into a training set and validation set; each training run used 5-fold validation to run the training. Each algorithm is tuned using two hyperparameters and trained on both of the aforementioned datasets. The final analysis is conducted with the algorithm tuned with the best performing hyperparameter and compared according to training set size.

**Decision Tree**

A Decision Tree utilizes a tree of decisions to make a final classification on a data point. The goal of each decision is to decrease the number of possible outcome classifications a given input could have; the tree is run on the entire dataset and generated by choosing features that decrease the overall entropy of the dataset.

**MNIST**

The above charts are of the Decision Tree trained on the MNIST dataset. According to the left chart, the max number of leaves seems to only matter up to a certain point, specifically around the 400 mark. An increase in the number of leaves continues to improve the training performance but does not improve the validation performance. A likely explanation for this trend could be the higher number of leaves cause overfitting on the data.

Similar performance can be observed on the right graph. As the tree depth grows larger, the decision tree performance on the validation set reaches a plateau at around 0.85 accuracy. Again, the cause of the increase in training performance but stagnating validation performance could be due to overfitting.

**Wine Quality**

The above charts are the Decision Trees trained on the wine quality dataset. Both charts show similar learning curves: the training and validation accuracy are similar at the beginning, but quickly diverge as the hyperparameter increases. Since the training accuracy approaches 1 for max depth and even reaches 1 for max leaves, but the validation accuracy remains relatively constant. A likely explanation for these two graphs is overfitting, since with higher training accuracy, a higher validation accuracy does not follow. With the small amount of data but high number of hyperparameters, the decision tree begins to learn the noise of the dataset and fails to generalize across the entire function space, producing the even higher variance seen in the graphs above. The small dataset size also contributed far more to overfitting in the wine dataset, since unlike MNIST dataset, the validation curves do not reach high accuracy at all.

**Final Hyperparameter Choice:** Max Leaves: 400 Max Depth: 10

**KNN**

KNN, or K-Nearest Neighbors, uses a distance metric among a cloud of points to classify a given data point. Each sample in a dataset is considered a point in N-dimensional space, where N is the number of features a sample has. Distance is then calculated to find the nearest K points and classified according to the points found.

**MNIST**

For the left graph, as K is adjusted, both the validation accuracy and the training accuracy decrease. The training accuracy begins at 1, which is expected since the first nearest neighbor among the training set is always the sample itself. As more K is calculated, the training error increases, since the more samples that are taken into account, the more likely the sample itself will be misclassified into another class. Since the clusters produced by the feature space can include overlapping points, having higher K value can lead to misclassification since adjacent points are not necessarily of the same class. In addition, the high performance of the algorithm suggests the noise floor may be reached, as the accuracy actually decreases with increasing K, as explained above.

The different distance metrics perform similarly across the training and validation sets, with the exception of the Chebyshev distance, which is approximately 0.2 less accurate compared to the other datasets. The Euclidean, Manhattan, and Minkowski distances are similar to each other, describing the distance in a traditional Cartesian sense between two vectors, while the Chebyshev distance is the maximum difference in any given dimension for two vectors, so even though the Euclidean distance of a point may be closer to a vector, the classification may be different. With the high number of training samples, the Chebyshev distance misclassifies some vectors since slight distance differences are amplified with the large training set.

**Wine Quality**

For the left graph, as K is increased, the training accuracy decreases while the validation accuracy remains roughly the same. This trend on the Wine dataset is similar to that of the MNIST dataset; the higher K values increase the likelihood that vectors are misclassified, since the presence of overlapping vector clusters can cause misclassification since nearby vectors are of another class. An important distinction between the above graph and the one for MNIST, however, is that the error levels out earlier. Since the wine dataset contains far fewer dimensions and samples than MNIST, bias is much more likely to be present. As demonstrated in the graph, the accuracy values begin to converge, indicating presence of bias.

The distance metric graph is similar to MNIST, but Chebyshev distance performs equally well to the other distance metrics. Even though Chebyshev calculates distance significantly differently than the other metrics, the comparatively low number of training samples means that slight differences in distance calculations are negligible. Therefore, for small datasets the distance metric does not influence the performance significantly. For this dataset the choice is still the Euclidean metric, since it slightly outperforms the other hyperparameters.

**Final Hyperparameter Choice:** K – Neighbors: 1 Distance Metric: Euclidean

**SVM**

SVM, or Support Vector Machine, draws support vectors on a dataset based on the presence of clusters of points in the feature space. The goal of the algorithm is to find the set of support vectors that misclassifies the least number of points, and the algorithm iteratively finds the best way to assign the vectors using the kernel method.

**MNIST**

In SVMs, the C value is a regularization parameter that determines the penalty for the support vector margin constraints. The higher a C value is, the higher the penalty for point misclassification, thus tighter vectors will be created. Lower C values allow more misclassifications and draw simpler support vectors. We can see that for very low values of C, the SVM does not describe the data well, and hence has low performance. Similarly, because of the high dimensionality of the data, for high values of C, the SVM over-penalizes incorrect values, leading to a drop in performance. The value that produces the best results is at 0.1.

The worst performing kernel is the polynomial kernel. The high dimensionality of the data may explain this discrepancy. Using a polynomial data to fit to data with 784 dimensions is difficult, as the support vectors generated by the polynomial is likely to cause many samples to be misclassified into other categories than intended; the low validation accuracy then follows from the low training accuracy. The other kernels show similar performance to each other, with linear slightly beating the others. Even with such a high dimensionality of data, it seems that sometimes simpler is better, since the handwritten digits are different enough to be linearly separable even on a high dimensional space. Indeed, as evidenced by the graph there seems to be linear, smooth properties associated with the dataset.

**Wine Quality**

We see that very low values of C are detrimental to performance, while higher values of C do not produce much improvement. Unlike with the MNIST graph, the C value curve levels off, probably due to the lower dimensionality and size of the dataset. Low values of C do not penalize misclassifications enough to be considered accurate, while higher penalizations produce support vectors which are not much different from slightly smaller values. Therefore, the C value that will be chosen is 0.1, since it produces the most consistent accuracy across both datasets.

While the different kernels had lower accuracy compared to the MNIST dataset, the results were similar in that the linear kernel produced the best accuracy and the poly kernel performed the worst. The difference in performance is not as drastic as for the MNIST dataset, but the poly kernel displays its weakness nonetheless by producing lower accuracy. Despite the lower dimensionality of the dataset, there are still enough dimensions to make the classification problem for poly kernel difficult. The data also happens to have a linear, smooth structure, as evidenced by the higher performance for the linear, rbf, and sigmoid kernels. The linear kernel only slightly edges out the other kernels, and we therefore pick the linear kernel as the best hyperparameter.

**Final Hyperparameter Choice:** C Value: 0.1 Kernel: Linear

**Boosting**

Boosting is an iterative variant on Decision Trees in which a weak classifier is fit on a dataset, and then subsequent copies of the classifier are iteratively fit with each iteration. Weights are used to focus on the difficult samples, with higher weights being placed on incorrect classification versus lower weights being placed on correct classification. This assignment used the Adaboost.SAMME.R variation of the algorithm, which supports multiclass classification.

**MNIST**

The curves for validation and training closely follow each other for both hyperparameters. The number of estimators seems to show significant improvement for the first 30 estimators, but as more estimators are added, the accuracy does not improve at all. Overfitting is unlikely to be present in this case, since the training accuracy closely followed validation accuracy. We can conclude that the algorithm can generalize this dataset well, since the training accuracy is almost the same as the validation accuracy. However, the lower training accuracy compared to the other algorithms displays the weakness of the weak learners. Successive iterations will not significantly improve the algorithm since the weak learner can only learn a certain amount from the dataset.

Notice that the maximum corresponds to the maximum accuracy obtained by the number of estimators. For boosting, there is a tradeoff between the number of estimators and learning rate, since the learning rate controls how much the algorithm adjusts for each successive estimator; therefore, each curve for different datasets will be different, but generally lower learning rates perform better since the algorithm will slowly approach a local minimum. Larger learning rates produce variance, since the collection of the weak classifiers will produce many different results on the dataset; placing lower learning rate values on the estimators will allow the algorithm to discount noise, but too low of a learning rate will not allow the successive classifiers to update the algorithm.

**Wine Quality**

We see again that the curves for the Adaboost hyperparameters closely follow each other, albeit at a much lower accuracy. For this dataset, the number of estimators does not have a significant impact at all on the accuracy of the predictions. The only noticeable effect is a slightly higher accuracy rate for 1 estimator, but for all other number of estimators the accuracy hovers around 0.39. For such a low feature space and low sample count, large number of estimators do not significantly impact the accuracy, since weak learners can only estimate so much out of the dataset.

The learning rate hyperparameter for this dataset achieves a maximum accuracy at around 0.25, much like the MNIST dataset. While the curve is different regarding the overall shape, the maximum observed corresponds to the converged accuracy when comparing the number of estimators. For this dataset lower values of learning rate do not significantly impact the accuracy, although it is noticeable that the accuracy drops. The lower dimensionality and lower sample size mean that the algorithm learns the features earlier, so successive iterations improve the accuracy only slightly. With higher learning rates, the lower sample size means that the successive weak learners will have a higher chance of learning the noise in the dataset, thereby decreasing overall accuracy. We select 0.25 to achieve a balance between both the datasets tested.

**Final Hyperparameter Choice:** Number of Estimators: 50 Learning Rate: 0.25

**Neural Network**

Neural Networks, in this case MLP, or multilayer perceptron, uses a network of functions that takes in inputs and changes weights iteratively. This style of supervised learning algorithm is extensively used in the industry due to its ability to learn complex functions on a variety of datasets. In fact, it can be proven that a multi-layer neural network can learn any arbitrary function. This expressiveness comes at a cost; this algorithm requires many resources to train on the dataset.

**MNIST**

The max iteration of the neural network shows little improvement over time. The accuracy is quite high for both the validation and the training set, indicating accurate learning from the dataset. Here we see the effect of having a very large dataset and a good algorithm: while the error is not quite at the noise floor, it is close enough to the best performing algorithms that the max iterations produces negligible performance differences.

The number of hidden layers improves the accuracy of both the validation and training sets. While it may seem like the dataset is overfitting with high variance, the curves seem to indicate that the algorithm is converging towards the noise floor, as the validation accuracy is approaching a similar number as the value for KNN where k equals 1 neighbor did. The combination of a high number of training samples and a complex model allows the algorithm to perform well and avoid bias and variance. Since higher complexity did not yield better improvement, 6 hidden layers seems to be the point at which one reaches diminishing returns.

**Wine Quality**

The accuracy of the max iterations hyperparameter is smaller than that of the MNIST dataset. In this graph we can see the effect of having a smaller dataset on the results; while the trend for each successive iteration is positive, the improvement for each successive increase in iteration is minimal. This is due to the fixed parameters of the neural network, with a fixed network configuration, it can only learn so much for every iteration before reaching the limits of the network and dataset. We therefore set the hyperparameter to 200 max iterations, since the difference between 200 and 400 is only slight, with 200 producing results much more quickly.

Again, we can see the power of the neural network come in play. A similar observation can be made for this dataset as for the previous MNIST dataset: the hidden layers converge to near the noise floor, and the number of hidden layers reaches a point of diminishing returns after about the 6th hidden layer. While the number of training samples is lower than that of MNIST, there seems to be enough of a combination of training set size and model complexity to learn the features quite well.

**Final Hyperparameter Choice:** Max Iterations: 200 Hidden Layers: 6

**Analysis**

Below is a plot of the aggregate performance on a test dataset. The test dataset was created from a subset of the initial starting set and set aside for the final testing. From the perspective of the algorithms this dataset has not been seen before, so the results from this analysis will give a good idea on the generalizability of each algorithm.

**Aggregate Learning Curve Graph**

**MNIST**

The graph shows all algorithms converging to a stable point as training size increases. The stability indicates that additional training data will not improve the performance of the algorithms; instead, what appears to influence the performance the most is the algorithm and hyperparameters themselves. We can therefore conclude that the dataset suffers from neither high bias nor high variance. While certain algorithms, such as decision tree, show signs of converging to a central point and displaying bias, the overall performance of the algorithms demonstrate the powerful effect a large, feature rich dataset has with a well-tuned algorithm.

With the current dataset, we can see that neural network performs the best. Excluding the testing performance of KNN, which is 1 since k neighbors is 1, we can see that the neural network algorithm performs the best out of any other algorithm.

**Wine**

While there are a few contradictory examples, the overall graph seems to be trending towards a single error level. The discrepancy between the overall training accuracy and the testing accuracy indicates that the dataset suffers from high variation. More data would likely rectify the problem, as most of the algorithms seem to be learning the noise rather than the signal, thereby overfitting. The most drastic non-trivial example is the decision tree: while the training accuracy is relatively high, the low performance of the testing set indicates overfitting on the data. KNN training accuracy remained at 1 due to the choice of 1 nearest neighbor and we can see that the testing accuracy is among the highest of all the other algorithms.

The best algorithm for this dataset would be KNN. Despite the performance of the neural network, which seems to approach the bias convergence point much more quickly than the other algorithms, KNNs testing accuracy is slightly ahead of the neural network’s.

**Final Thoughts**

This analysis yielded interesting insights about the 5 different supervised learning algorithms analyzed. Training dataset size, hyperparameter choice, and algorithm choice all influence the final performance of the algorithms significantly. Noise is perhaps the most important factor when designing a machine learning problem, since knowing the level of noise in a dataset determines whether it is better to collect more data, which is expensive, versus simply tuning hyperparameters.