**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 even dataset to classify digits, but the high number of features means that overfitting is a concern, hence the focus on graphing multiple hyperparameters with respect to the validation/training performance.
* **Credit Card Fraud:** This dataset was obtained from the website Kaggle.com, a repository for various data competitions. The data contains transactions made by credit cards in September 2013 by European cardholders, and

**Methods**

All algorithms were trained on two datasets. The sets were split into a training set and validation set. Each training run used 6-fold validation to run the training, in which the training set was split into 6 different sections, for 6 different iterations, the algorithm was trained on 5 of the 6 sets, with the 6th set used for validation. 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.

**Hyperparameters**

The decision tree is limited by the depth of the tree and the number of leaves it may have.

**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.

**KNN**

KNN, or K-Nearest Neighbors, uses a distance metric between a cloud of points to classify a given data point.

**Hyperparameters**

The algorithm is trained with hyperparameters as follows

**MNIST**

**SVM**

SVM, or Support Vector Machine, draws hyperplanes on a dataset and divides the data.

**Hyperparameters**

The algorithm is trained with hyperparameters as follows

**Boosting**

Boosting is a variant on Decision Trees in which the training places weights on whether the data is classified correctly or not.

**Hyperparameters**

The algorithm is adjusted by the learning rate and maximum number of estimators.

**MNIST**

The curves for validation and training followed closely with each other.

**Neural Network**

Neural Networks, in this case MLP, or multilayer perceptron, uses a network of functions that takes in inputs and changes weights iteratively.

**Hyperparameters**

The network is trained with hyperparameters: Alpha Value and Number of Hidden Layers

**MNIST**

**Analysis**

Below is a plot of the aggregate performance on a test dataset.

**Aggregate Learning Curve Graph**