MACHINE LEARNING – OMSCS7641
SPRING 2022
SUPERVISED LEARNING

INTRODUCTION

This paper examines five supervised learning techniques on two different datasets. It compares and contrasts the techniques, their hyperparameters on two different datasets. The supervised learning techniques implemented are Decision Tree, Neural Network, AdaBoost, Support Vector Classification (SVC), K-Nearest Neighbors (SVC). I used Scikit for implementations and I took advantage of my previous codes from the Introduction to Machine Learning Class from Udacity.

DATASETS

The first dataset is from the UCI database (1). It has six input features. Each feature contains 3-4 distinct values such as low, med, high. The output is ordinal, multiclass classification. The input and the output contain only discrete values. The input contains several features of a car such as number of doors or leg room. The output shows whether a **car** is unacceptable, ok, good or very good. There are 1727 rows.

The second dataset is from Kaggle (2). It shows whether the **water** sample is potable or not. There are nine input features. Each feature consists of continuous real values. Some of the features are pH, hardness, sulfate concentration etc. The output is binary (potable or not potable). There are 3276 rows.

Why they are interesting: I selected these datasets because they are not trivial, nor too hard. There are significant correlations between the input and output. Machine learning tools are able to predict to some degree. Each learner predicts at a different success level.

First Dataset – Cars

Data Preparation

The size of the data is 1727*7 including the input and output. The labels are converted from text to integers such as 0, 1, 2, 3 for each feature separately. The data did not contain any Nan or empty values. **Metrics:** The output data is unbalanced as shown in Figure 1. Most of the data contain only the unacceptable cars. Therefore, a high accuracy does not guaranty high precision or recall. For that reason, I used F-score as the evaluation metric which balances between precision and recall. High precision minimizes the false positives meaning a bad car wouldn't be identified as a good car. High recall minimizes the false negatives meaning a good car wouldn't be identified as a bad car. F-score would be balancing between these two metrics. I used a beta score of 1 which gives equal weight to precision and recall. I also used the macro option as the output is multilabel, not binary. Macro option of sklearn.metrics.fbeta_score calculates the f-score for each label and takes its mean without weighing. I used macro because I wanted to give equal importance to each label.

Training, Validation, Testing: Data is split into 80% training and 20% test. Training set is further split into 80% training and 20% validation sets. Each time data is randomly shuffled, and the random state is fixed for reproducibility. It is important to split the data into three as each as its own purpose. Training set is used to tune the hyperparameters. Validation set checks for things like overfitting, underfitting, ability of prediction outside the training set. Hyperparameters are tuned back and forth by fitting into training set and evaluating the validation set. Testing set is to see the final prediction once we are satisfied with the model. At this point, the model cannot be tuned anymore because that's cheating.

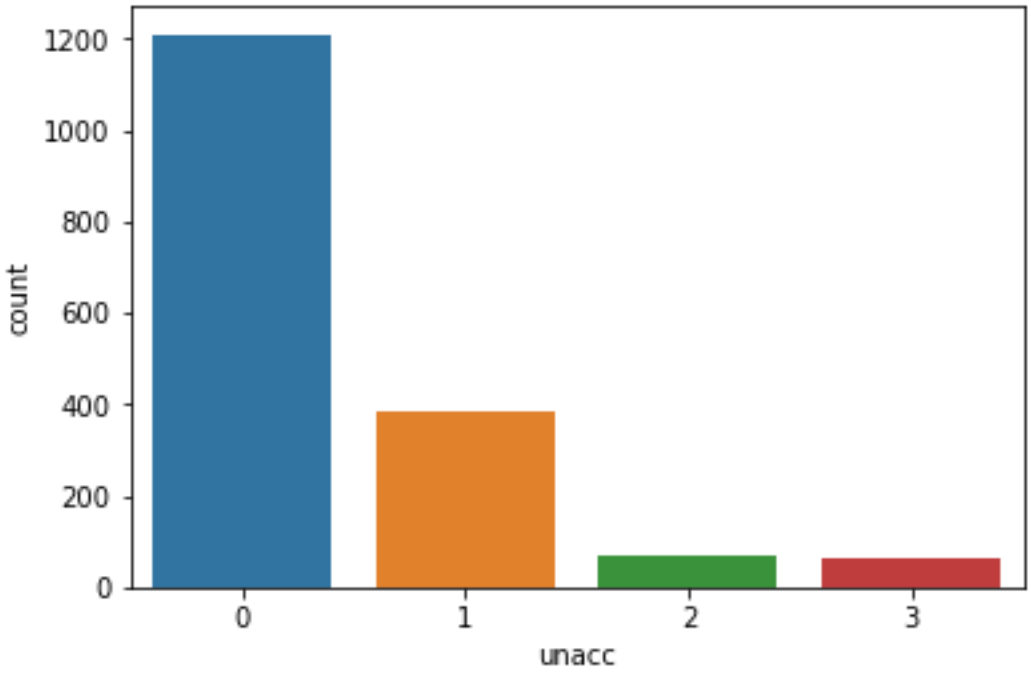

Figure 1. The count number of each output label from 0 to 3 i.e. unacceptable, acceptable, good and very good.

Scaling: I scaled the input data between 0 and 1 by using MinMaxScaler. It is easier to converge on scaled data and hence it is recommended. It can take much longer to converge on unscaled data. The scaling is done first on the training set. By using these scaling parameters, validation and training input features are scaled as well. I did not scale the output features. The output varies from 0 to 3 and learning predicted well.

Cross-Validation: Cross validation can be particularly useful if the data is small and we do not want to waste some of the data for validation. It can also show whether the data is represented well in k different minisets. In order to check whether cross-validation would be beneficial for my case, I did a 5-fold cross-validation on a Decision Tree classifier. The cross validation accuracy results are 0.88, 0.88, 0.85, 0.84, 0.85. The cross validation shows that the data is represented well and the accuracy result for each validation is similar. Therefore, I decided not to use cross-validation for this dataset.

Computation Time: Figure 2a shows the computational time to train each learner as a function of training subset size (1%, 10% and 100% of the training set). KNN and Decision-Tree are the fastest because these are not iterative models. KNN training time only relates to creating a hash table. Decision-tree is based on a recursive function and it uses a maximum depth of 5 and the gini function. As the tree is not very deep, its training time is close to 0. SVC is a method that gets slower as the data set gets bigger. Therefore, for 1% and 10%, training subset, SVC training time is close to 0. However, for the full training subset, it gets slower. Adaboost consists of 50 different Decision Trees run serially. Therefore, it takes more time to train. Neural network is the slowest for training as it consists of 3 layers each with 10 neurons, tanh activation function. The number of hyperparameters to tune and derivatives to calculate increase greatly with the number of layers, neurons and the dataset size. Figure 2d shows that once a NN is trained, it is very fast in prediction as it is only a set of one step forward matrix multiplications. Similarly, prediction time for Decision Tree is quite fast as gini function is not used here and the output is found as a result 5 consecutive yes/no questions. Adaboost also does the same as Decision Tree for prediction except there are 50 decision trees now. Therefore, it takes more time to predict. KNN is known to train fast but predict slower as it has to do a dictionary search and it needs to find the 5 nearest neighbors. For prediction, SVC is the slowest among all when the full-size training set is used. As I used the rbf

kernel for SVC and the training subset is larger, it is possible that the number of support vectors is much larger. This may cause larger prediction time. I need to note that I have found that SVC training time can vary largely based on the size of dataset, number of iterations required, kernel used and other hyperparameters such as C and Gamma. Please note that Figure 2d plots validation time as a function of training set size; and the validation set size doesn't vary. Still, KNN and SVC take much longer to compute than smaller training set size. The reason for SVC being slow is there are now many support vectors to compute that were obtained while training a large dataset. The reason KNN is slow that the dictionary that stores the training dataset is much larger even for the same validation set size.

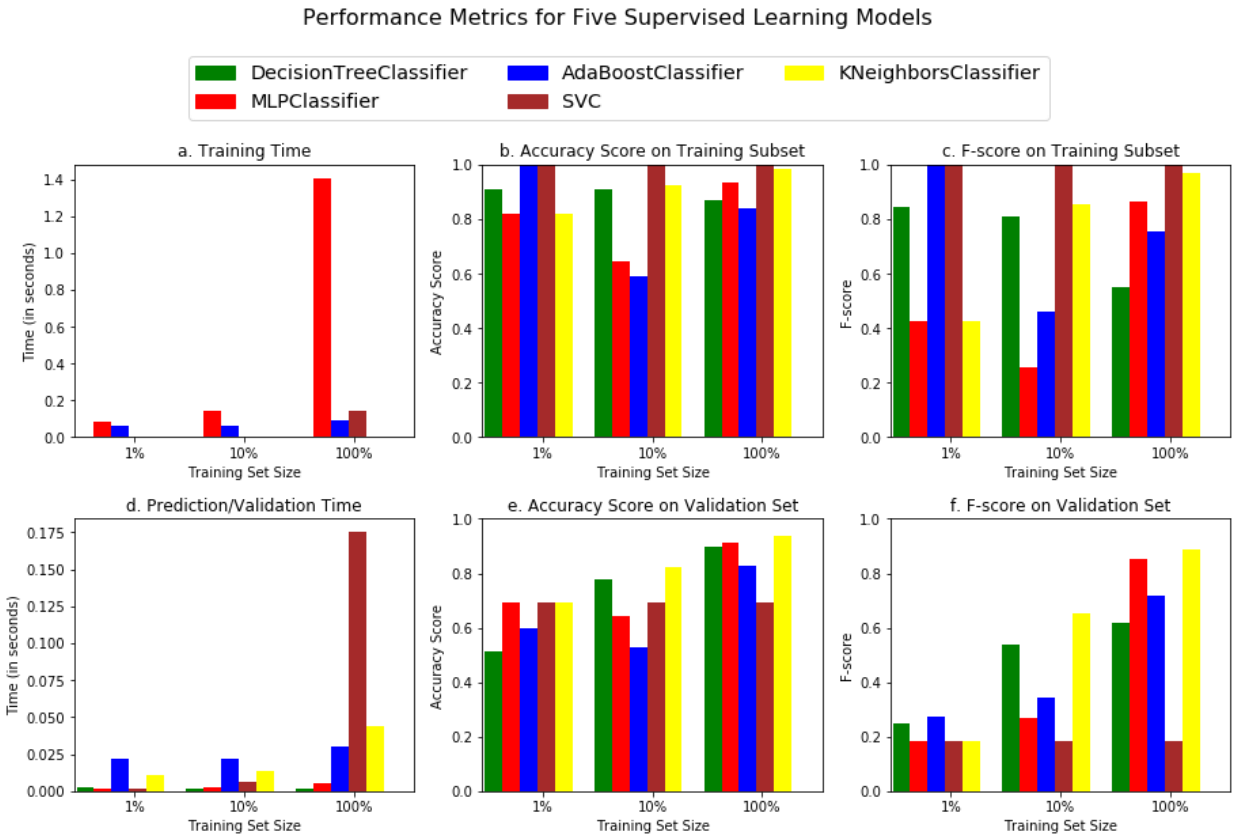

Figure 2: a) Training time for each algorithm, b) accuracy score on training set, c) f-score in training set d) validation time, e) accuracy score on validation set, f) f score on validation set.

Learning Curve: Learning curve plots evaluation metrics such as accuracy, precision, recall or f-score as a function of training set size. Figures 2b and 2c show the accuracy score, and f-score of the training set as a function of training subset size. Figures 2e and 2f show the accuracy score, and f-score of the validation set as a function of training subset size. It would be very easy to fit into smaller data, which would give low bias but high variance, if we have a complex enough model. We can see that SVC gives 100% accuracy and f-score for the training set for every training subset. That is because it is able to generate enough rbf support vectors to isolate every output class. However, this doesn't predict well as we can see its f-score is very small. This is an example of low bias and high variance. Decision tree is not complex enough to capture all the training set as we can see that its f-score is declining as the training subset grows in Figure 2c. However, Figures 2e and 2f show that its accuracy and f-scores are increasing with the training subset. We can see that training error grows and validation error declines with increasing training subset. We can see a similar behavior for Adaboost and Neural Network. For KNN, both the accuracy and f-score increase for both the training set and the validation set as the training subset increases. This is because as there are more neighbors showing similar properties, the accuracy

and f-score will increase. Overall, Figure 2f shows a positive correlation between f-score and the training subset size for the validation set. This means that as have more training data, we can predict more accurately.

I also compare the number of iterations for MLP as a function of training subset size. The number of iterations are 102, 142 and 401 for 1%, 10% and 100%, respectively. As the amount of data increases, it is expected to have larger number of iterations.

Optimization of Decision Tree Hyperparameters

I carried out a grid search of sklearn. I fixed the random state for reproducibility and used the gini function. I varied minimum samples allowed in a leaf, minimum samples to split and the maximum depth. I used fbeta score with beta 1 as evaluation metric. For the best model, f-score on the validation data is 0.88. This is much higher than 0.6 shown in Figure 2f. It finds the optimum parameters as minimum sample leaf size 2, minimum samples split size 2 and maximum depth 12. This optimization comes from the training metrics. Now, I would like to optimize based on the validation set by using valdiation curves. Figure 3 plots the f-score as a function of maximum depth and minimum leaf size. As the depth increases, the training set fits better and better because can ask more binary questions. However, validation f-score starts to decline after maximum depth eight. This is a sign of overfitting i.e. low bias high variance. After fixing the maximum depth, I plot the f-score as a function of minimum leaf size as shown on the right side of figure 3. 1 seems to be the optimal value for the minimum leaf size hyperparameter. This may be because there may be some specific subsets as small as one requiring a specific set of questions.

Figure 3. F-score for training and validations sets as a function of maximum depth (left), and minimum leaf size (right).

Finally, I decide to use a decision tree that has the following hyperparameters: minimum number of samples to split is 2, gini function is used to split, random state is fixed, maximum depth is 8 and minimum samples size in a leaf is 1. **This shows an f-score of 0.87 for the testing data**.

Optimization of Neural Network Hyperparameters

A grid search is performed through different activation functions (tanh, logistic, identity, relu), number of layers, number of neurons per layer and maximum iteration for convergence. The optimum values after optimizing to the training set based on fbeta score are tanh for activation function, 3 layers with 20 neurons each and 1000 maximum number of iterations. The solver is adam. I also carry out early stopping after 100 of iterations. Early stopping is very useful to avoid overfitting. When early stopping is on, 10% of the training set is used for validation and if the validation performance doesn't improve, the training stops to avoid overfitting. The

validation f-score is 0.82 which is very similar to what is shown in Figure 2f. So, there hasn't been any improvement.

Next, I plot the validation curves. Figure 4 shows the F score as a function of number of layers and number of neurons per layer. Both training set and validation sets show similar performance which means there is no overfitting. Also, as the F-score is very high for both the training and validation sets, we can say there is low bias and low variance. After 4 layers, validation set doesn't show any improvement. Therefore, I set the number of layers to 4. On the right side of Figure 4, I fix the number of layers to 4 and I vary the number of neurons per layer. After 20 neurons per layer, validation set doesn't show any improvement. Therefore, this becomes my final model. 369 iterations were required for convergence.

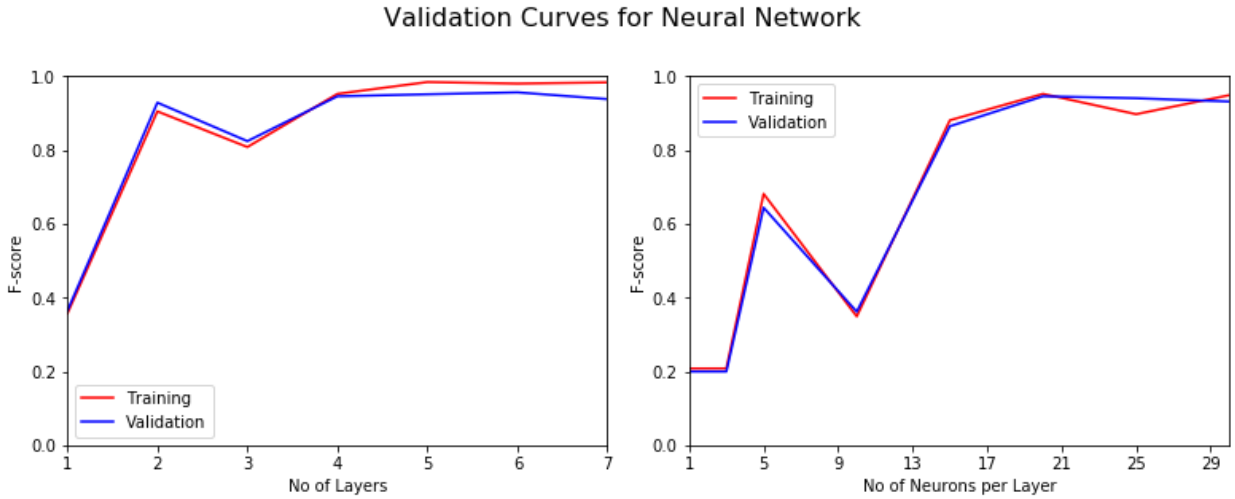

Figure 4: F-score for the training and validation sets as a function of number of layers (left) and number of neurons per layer.

Finally, I test the final model on the testing set and the **F-score is 0.84.**

Optimization of Boosting Hyperparameters

I use AdaBoost as a fixed random state for reproducibility. The base model for AdaBoost is a Decision Tree classifier with a maximum depth of 1 and with the gini function.. I do a grid search over number of estimators and the optimum number of estimators is 30, which gives a validation set F-score of 0.82.

Figure 5 shows the F-score for training and validation sets as a function of number of estimators (left) and the maximum depth for the Decision tree (right). We can see on Figure 5 (left) that training and validation scores are very similar. This means that validation set is well represented in the training set and the model is able to predict unseen data. Therefore, there is no overfitting. After about 26 estimators, validation score doesn't improve. Therefore, we start to change the maximum depth of weak learner. We can see on Figure 5 (right) that overfitting occurs after 5 maximum depth which creates high variance.

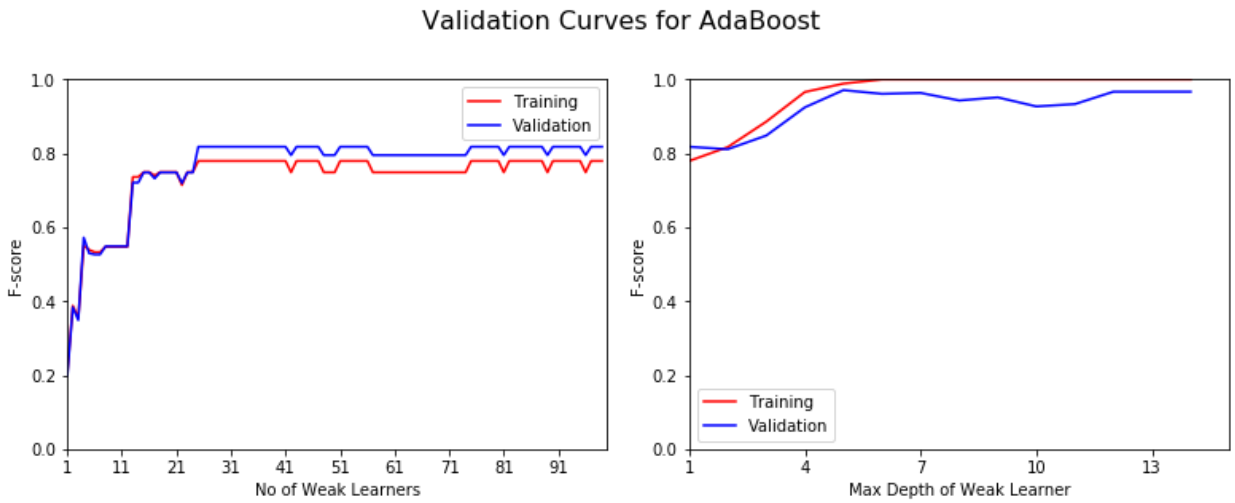

Figure 5: F-score for training and valdiation sets as a function of number of weak learners (left) and maximum depth of weak learner (right).

The finaly hyperparameters of Adaboost are maximum depth of weak learner is as 5 and number of estimators as 26 along with other aforementioned parameters. **Final F-score on the testing data is 0.92.**

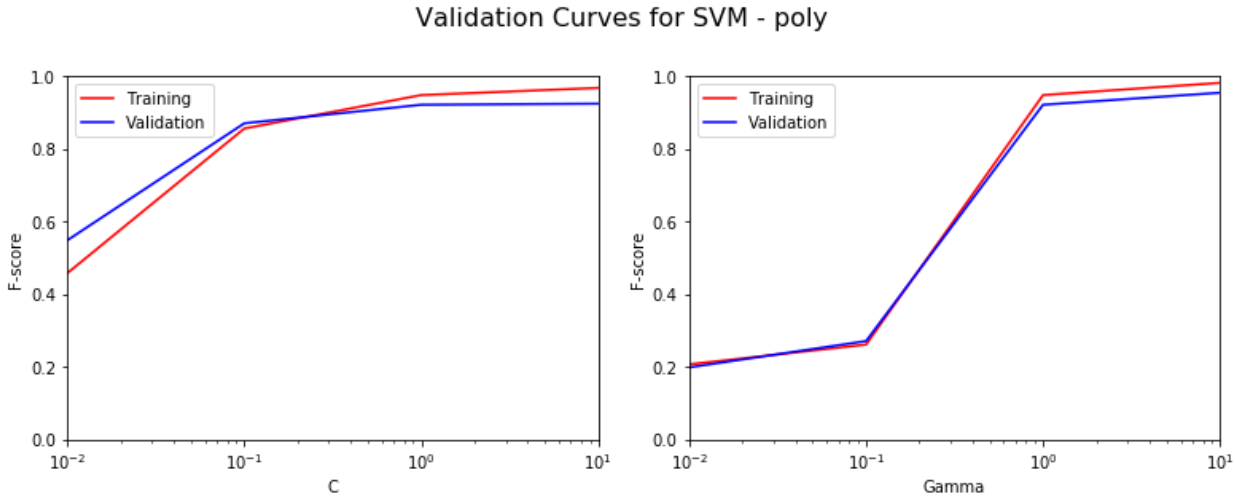

Figure 6: F-score for training and validations sets as a function of C (left) and Gamma (right).

Optimization of Support Vector Classifier (SVC) Hyperparameters

Initially, I fix the random state for reproducibility. I choose a polynomial kernel, C as 1 and gamma as 1. I vary C between 0.01 and 10. Figure 6 (left) shows that training set shows lower bias at higher C. However, we see no improvement in the validation set after 1 i.e. overfitting. Therefore, C is taken as 1. now, we vary Gamma between 0.01 and 10 as shown in Figure 6 (right). Both training and validation sets perform better at higher gamma (low bias, low variance).

Now, I change the kernel function to rbf to see if I can get something better. Again, I fix gamma to 1 and vary C between 0.01 and 100 as shown in Figure 7 (left). There is no overfitting because both sets perform similar. The performance is best at C 100. Now, I fix C to 100 and vary Gamma between 0.01 and 100 as shown in Figure 7 (right). Now, the performance is best at Gamma 1. Again, there is no overfitting as both sets show same performance. However, I bias starts to increase after Gamma 1.

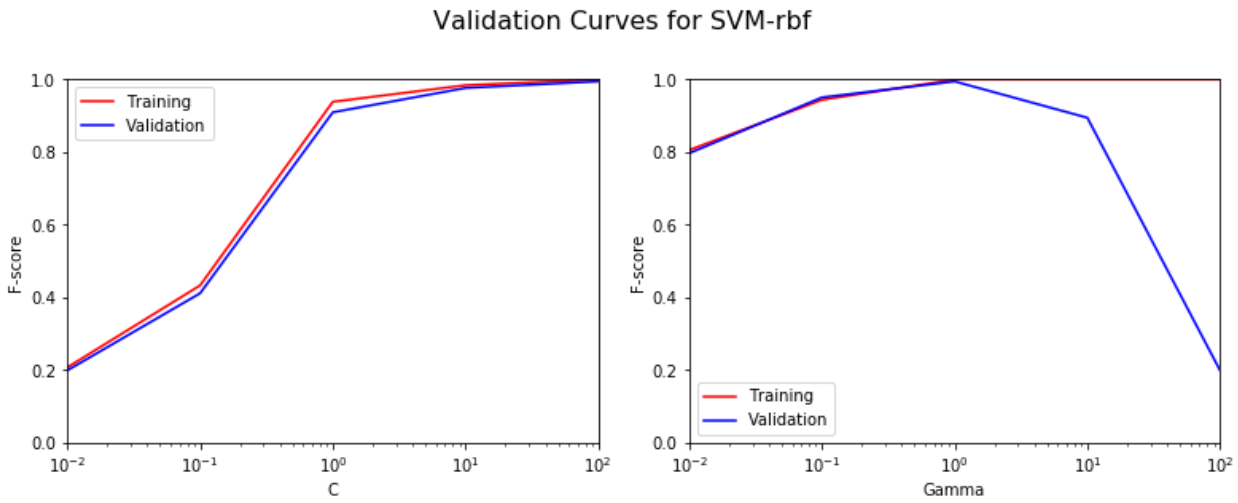

Figure 7: F-score for training and validation sets as a function of C (left) and Gamma (right).

Finally, I take the best rbf and polynomial-based models that I have and compare them as shown in Figure 8. Figure 8 (left) shows that rbf shows the best performance. Polynomials also fits well at higher degrees. However, rbf is better.

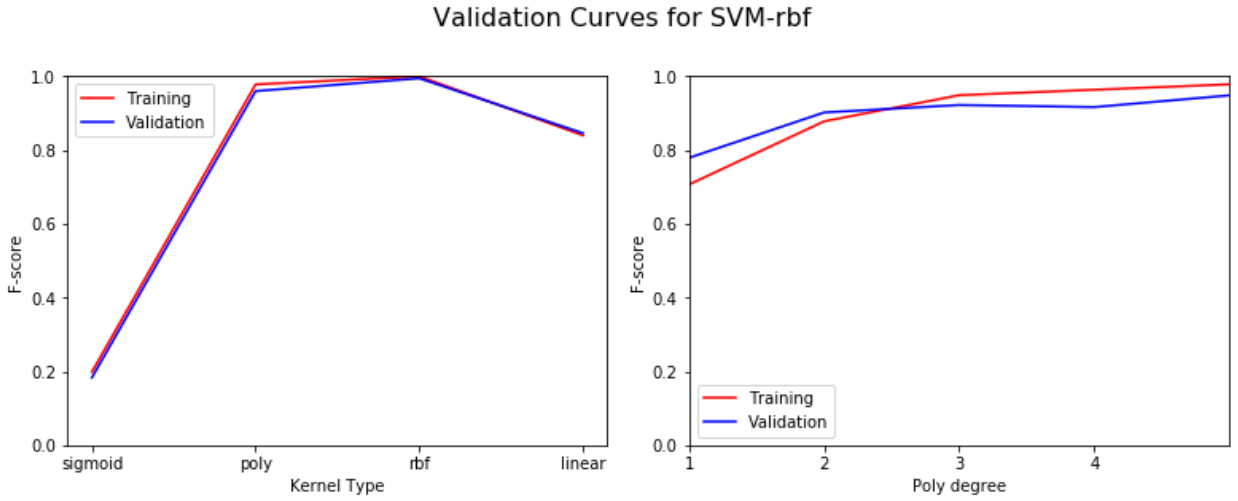

Figure 8: F-score as a function of kernel type and polynomial degree.

The final model is rbf with C 100 and Gamma 1. **The F-score for the testing set is 0.9978**. This means that all of the testing set is essentially predicted correctly.

Optimization of KNN Hyperparameters

For KNN, I vary the number of neighbors (the K value) between 1 and 100 as shown in Figure 9 (left). Only one neighbor estimates the best. This means that data is very smooth and there is no noise. At 1 k-neighbor, bias is close to 0 and variance is small as well. I fix the k-neighbors to 1 and vary the weighting schedule and both training and validation are on top of each other and uniform and distance-based calculation are equal. This is because as we copy the closest neighbor, its weight will always be 1 and hence identical results. It also means that training set represents the validation set and it is able to predict outside the trained data. **The final KNN model uses 1 nearest neighbor and the testing set f-score is 0.9978.**

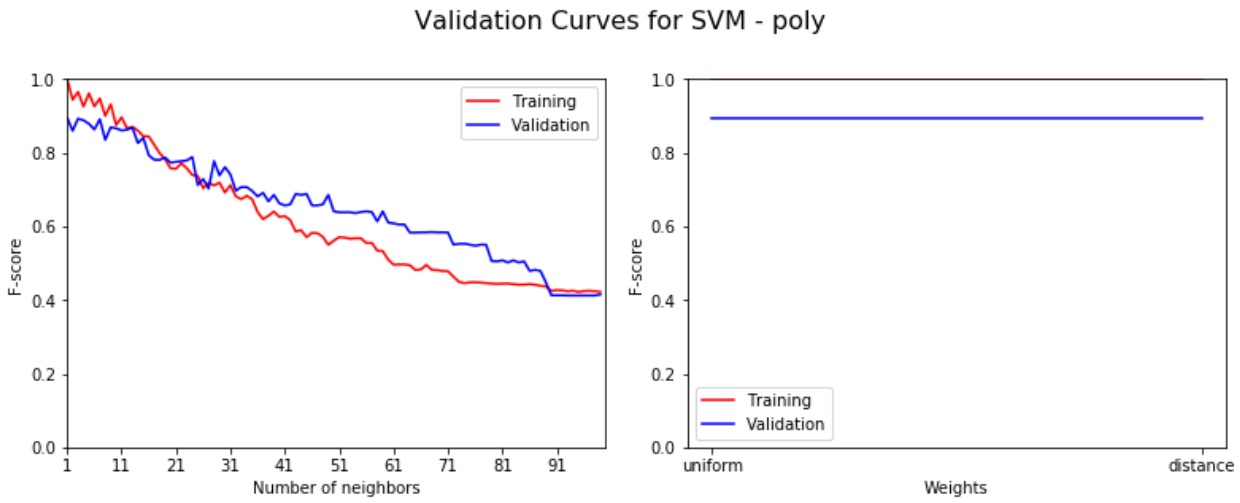

Figure 9: F-score as a function of number of neighbors and the weighting scheme used.

Comparison of Learning Techniques

We have optimized the hyperparameters of each learner and they predict much better than original (figure 2). But, how do they compare to each other? The F-score for the testing set for each learner is as follows. Decision tree gives 0.87, neural network gives 0.84, Adaboost gives 0.92, SVC gives 0.9978 and KNN gives 0.9978. Out of all the learners, KNN gives the best results. It is interesting because it was the simplest model. When I was optimizing decision tree and neural network, I may have skipped simpler models. AdaBoost improves the decision tree by focusing on the previous incorrect predictions. AdaBoost also simplifies the model by using a lower maximum depth. It is also interesting that an SVM-rbf model with high C gives very accurate results considering the risk of overfitting with high C values. Finally, each learner has its own mechanism and approach to predict. It seems SVC and KNN mechanisms estimate the best whereas the other learners are not able to capture the relation between the input and output as well as the first two.

Second Dataset -Water

Some of the explanations are skipped here as they are the same as the previous dataset.

Data Preparation: The input features are continuous, and the output is binary. There existed some empty values which I filled with the mean of its corresponding feature. First, I implemented MinMaxScaling between 0 and 1. However, iterative methods such as neural network and SVC were having convergence issues. Therefore, I finally decided to use Standard Scaling for this case.

Metrics: The data is balanced well but I decided to use F-score to both optimize precision and recall with equal weight. I used beta as 1.

Training, Testing, Validation: First, I divided the set into 80% training and 20% testing and then I divided the training set as 80% training and 20% valdiation. I ended up having 2096, 524 and 656 samples for training, valdiation and testing, respectively.

Cross-Validation: I do cross validation (5-fold) for an example Decision Tree and the accuracy scores are 0.6 3, 0.62, 0.61, 0.61, 0.67 for each validation set. As the results are very similar, I decide to skip cross validation and use one training set and one validation set as separate sets.

Learning Curves: Figures 10a and 10d show the computation time for the training and validation sets, respecti vely. The results are very similar to the previous dataset as shown in Figure 2. Therefore, I won't focus on here too much. For example, MLP classifier is very slow to train for large data but very fast to predict. KNN is fast to train but slow to predict for large data. SVC also predicts very slowly for large data probably because of the

number of support vectors that were created while training the large dataset. Unlike the car dataset, the second dataset doesn't show higher F-score at larger training subset which is the opposite of what I would expect. The number of epoch iterations for the MLP are 113, 139 and 392 for the 1%, 10% and 100% training subset size, r espectively. It is expected to have large number of iterations for larger dataset.

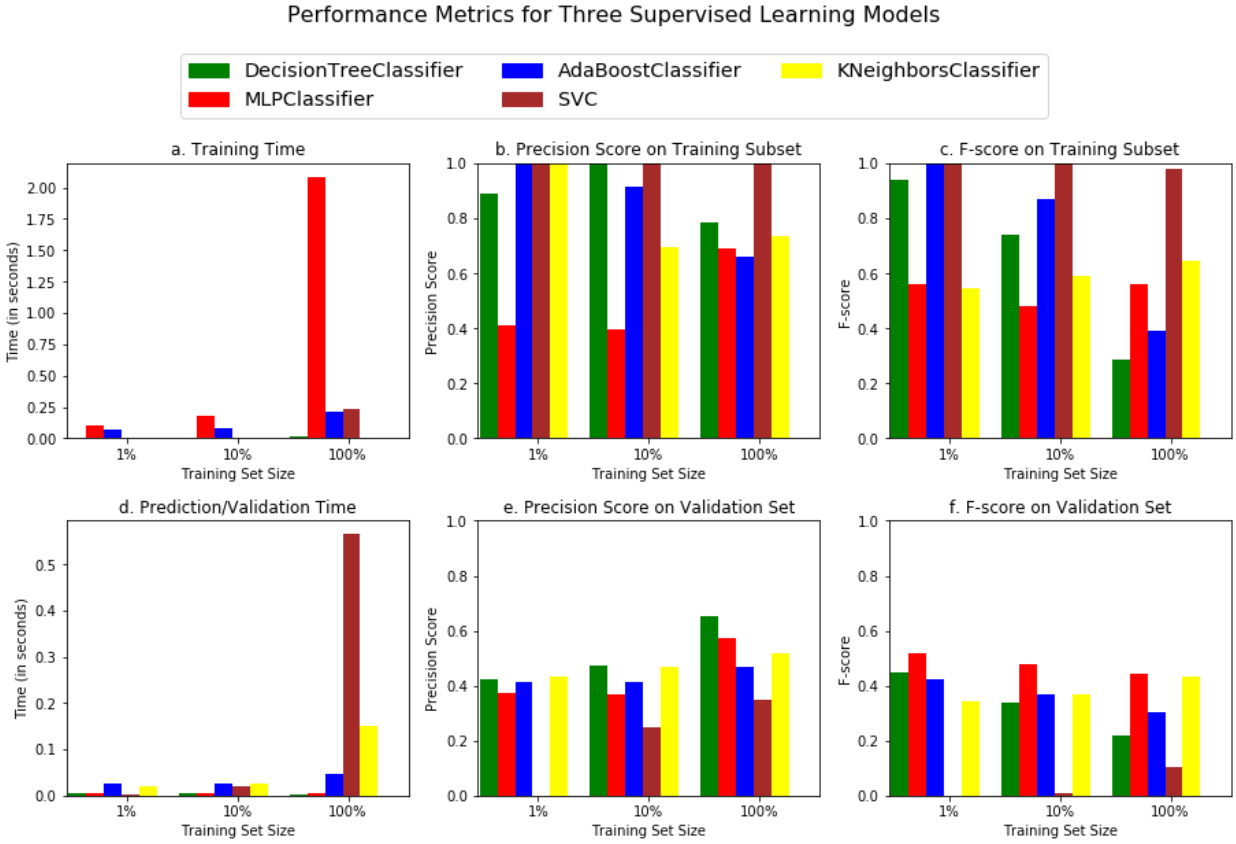

Figure 10: a) Training time for each algorithm, b) accuracy score on training set, c) f-score in training set d) va lidation time, e) accuracy score on validation set, f) f score on validation set.

Optimization of Decision Tree Hyperparameters

A grid search is carried out similar to the car dataset. Validation F-score increases to 0.56 which is much better than original 0.22 (shown in Figure 10f). The optimum parameters have a maximum depth of 20, minimum samples in a leaf of 4, and minimum samples required to split as 2. Again, the gini function is used.

As shown in Figure 11, first I vary the maximum depth between 2 and 22 and I find that maximum depth of 16 is optimal because training and validation scores are diverging after 16 which is causing overfitting and low bias and high variance. Then, I fix the maximum depth to 16 and vary the minimum samples size in a leaf. Bias is increasing with the increasing sample size. However, variance is not declining. Finally, I decide to use maximum depth 16 and minimum leaf size of 3 which gives an **F-score of 0.45** on testing set.

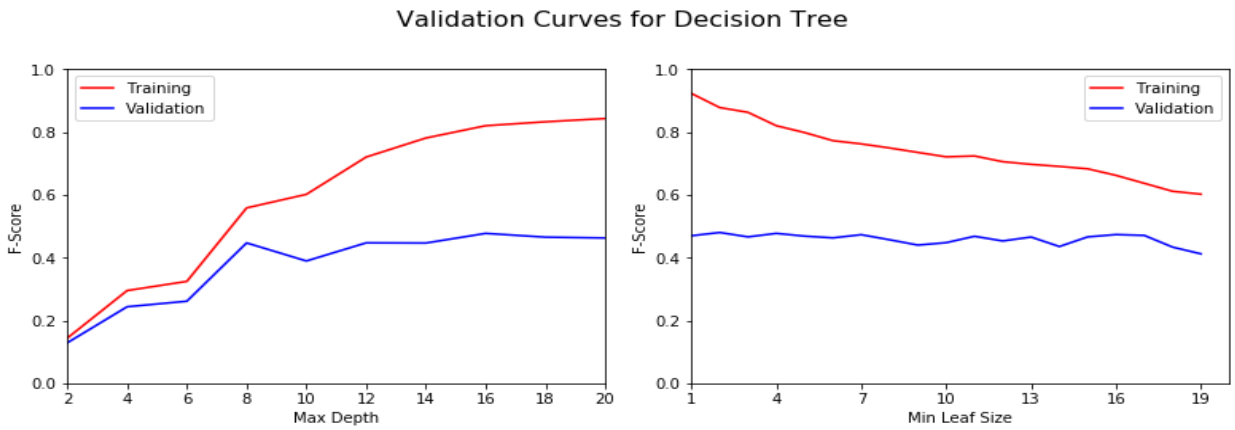

Figure 11. Training and Validation F-scores as a function of maximum depth and minimum leaf size.

Optimization of Neural Network Hyperparameters

I had a lot of difficulty in converging the ANN model. Initially, some of the hyperparameter values caused the failure in training of the model and the model only gave 0 as output. For that reason, I used Standard Scaling instead of MinMaxScaling, lbfgs solver instead of adam and I increased the maximum number of iterations to 100000. These changes resolved my problem. lbfgs is recommended for smaller datasets but it is said to be more robust. I noticed that it requires more iterations to converge. Like in dataset 1, I also used early stopping and number of iterations required for early stopping is 100. Random state is fixed for reproducibility. Activation function is tanh.

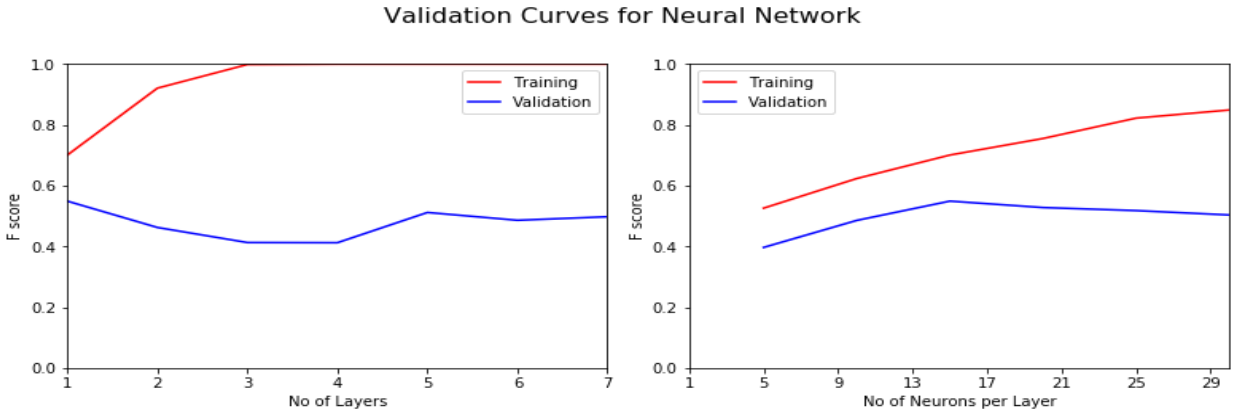

Figure 12: Training and Validation F-scores as a function of number of layers and number of neurons per layer.

Then, I vary the number of layers by fixing the number of neurons in each layer to 15. Figure 12 (left) shows that validation F-score performs the best at 1 layer. After that, there is overfitting, low bias and high variance. Therefore, I fix the number of layers to 1 and I vary the number of neurons. The number of neurons is optimal at 15. Before 15, there is high bias and above 15 there is high variance which causes overfitting. Therefore, I finally pick 1 layer with 15 neurons. 2334 iterations were required for convergence. **Testing set F-score for the final model is 0.49.**

Optimization of AdaBoost Hyperparameters

A grid search is implemented over an Adaboost model based on Decision tree classifier with maximum depth of 1. Number of estimators are varied between 1 and 100. Optimal number of estimators is 65. Figure 13 (left) shows that 98 weak learners is the optimal number. Basically, there is no significant overfitting at that moment even though it performs worse than the training set. Then, I fix the number of estimators to 98 and vary the

maximum depth for each estimator. I see very low bias and very high variance as the maximum depth of the weak learner increases. Validation score doesn't improve after 3 maximum depth. Therefore, I finally use 98 estimators each with a maximum depth of 3. This gives me a testing **F-score of 0.39.**

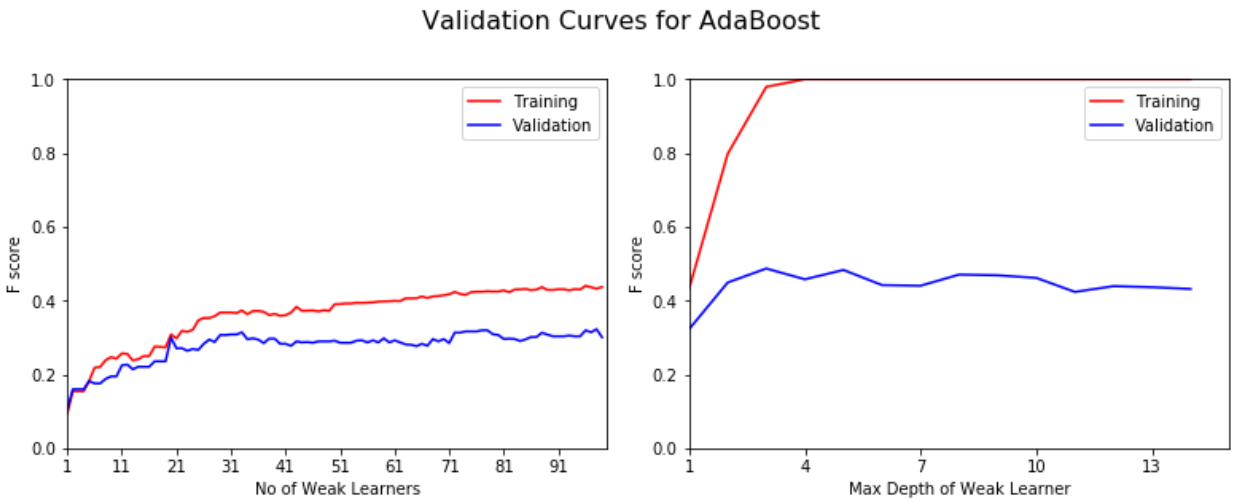

Figure 13: Training and Validation F-scores as a function of weak learner and the maximum depth of each weak learner.

Optimization of Support Vector Classifier Hyperparameters

Similary to the SVC in the first dataset, I tried different optimizations on different hyperparameters particularly for the polynomial and the rbf kernel as they were the most promising ones. I only show two validation curves here as I do not have more space left. Finally, I decided to pick an SVC with fixed random state, rbf kernel, 10 C and 0.1 Gamma as this was showing the best validation results after a series of validation curve studies. Figure 14 also compares several kernel functions where rbf shows the best performance. I can see that there some variance here. However, it is still better than the other kernels. Also, I can see that the polynomial kernel performs the best at 4th degree based on the validation score. At 5th degree, we start to see extreme overfitting, low bias and high variance. **The final SVC model gives 0.46 F-score on the testing dataset.**

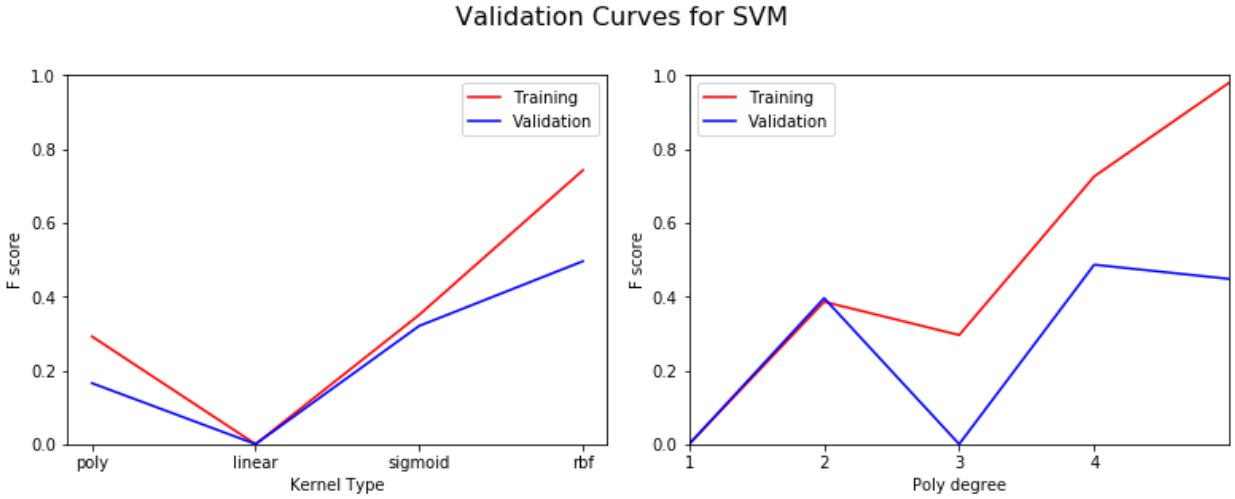

Figure 14: F-score for training and validation sets as a function of kernel type and the polynomial degree.

Optimization of KNN Hyperparameters

My hyperparameters for KNN are the same as the first dataset. Please see the first dataset for further explanations. I decided to use 1 nearest neighbor only and it gives a final **testing F-score of 0.46.**

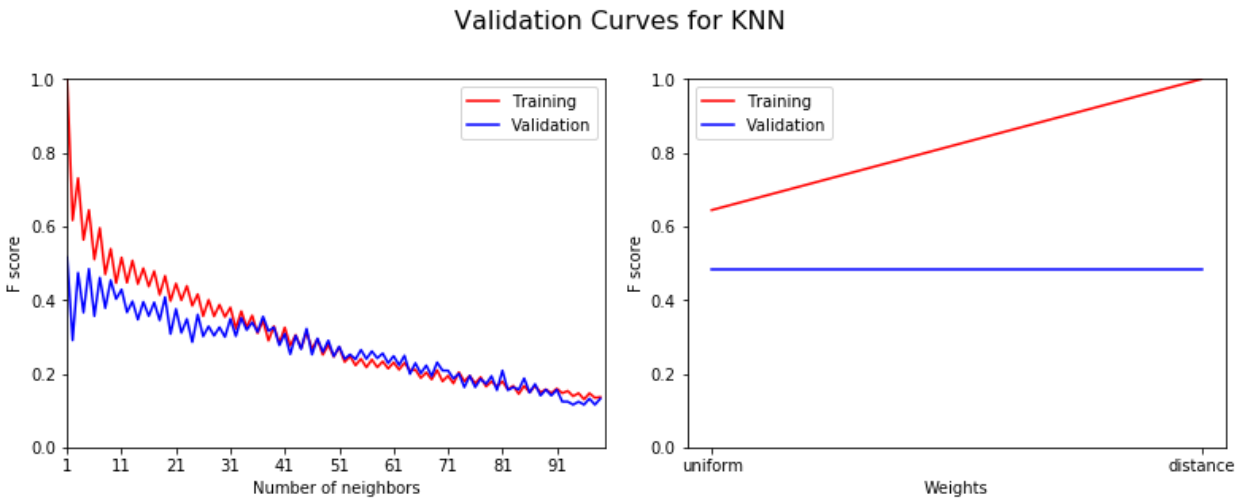

Figure 15: F-score for training and validation sets as a function of number of neighbors and weighing function.

Comparison of Learning Techniques

The testing F-scores are 0.45 for Decision Tree, 0.49 for Neural Networks, 0.39 for Adaboost, 0.46 for SVC, and 0.46 for KNN. This time ANN is the best predictor. I am not sure why that would be the case. However, I used a simpler model with 1 layer which may have helped. A high value for KNN suggests the data is relatively smoot with not much noise. Adaboost improved the decision tree only slightly without much improvements.

Comparison of Datasets

The main difference between the two datasets' performance is that the second set has much lower F-scores. That is most likely because the input features are not very good indicators of the output feature. It is possible that we need more input data to connect the input with the output.

I also noted that MLPClassifier requires 10 times more iterations for the second dataset because it uses lfbgs solver instead of adam.

REFERENCES

1. https://archive.ics.uci.edu/ml/machine-learning-databases/car/
2. https://www.kaggle.com/adityakadiwal/water-potability