Assignment 1 Report

1st Zebing Li
Georgia Institute of Technology
Atlanta, Georgia
zli738@gatech.edu

***Abstract*—This paper tries to resolve two classification problems using different supervised learning models. The problems are mushroom classification and heart failure prediction. And models experimented with are Neural networks, Support Vector Machines, K-Nearest Neighbors and Gradient Boosting Decision Trees. The paper compares the differences in the model behaviours on the two datasets from different aspects, and analyzes the possible reasons of such differences.**

I. Introduction

Supervised learning techniques are widely applied to classification problems and are proved to be effective and adaptive to different types of datasets and real-world problems. However, there are multiple types of common supervised learning algorithms that can be applied to classification, such as decision trees, Support Vector Machines (SVM), K-Nearest Neighbors (KNN), etc. These algorithms have different assumptions, biases and preferences on different types of datasets. And they may have different performances depending on the features of the problem and dataset, and also require various types of preprocessing, training and hyperparameter tuning. Therefore, it is very important to select the appropriate supervised learning algorithm based on the understanding of both the dataset and the algorithm itself.

In this paper, I explored different supervised learning algorithms on two classification problems, in order to compare their behaviors and analyze their differences. I will first introduce the two problems and the datasets. Then in methods part, I will briefly introduce the algorithms and model used. Then I will share the results with figures and tables demonstrating different aspects of the experiment. After that, I will analyze the results and discuss the findings.

II. Datasets

A. Mushroom Classification

The first problem is Mushroom Classification. “This dataset is a cleaned version of the original Mushroom Dataset for Binary Classification Available at UCI Library”. [1] It is a binary classification problem, giving different features of a mushroom and to predict whether it is edible or not. There are 9 features describing aspects of caps, gills, stem, etc. And the target class is edible (0) or not edible (1). Among the features, cap shape, gill color and stem color are categorical features, others are numerical. Originally there were 54K rows of data and it took a significant amount of time to train and fit the models. I picked 5100 rows out of the original data to reduce the amount of training time while keeping enough amount of data for training.

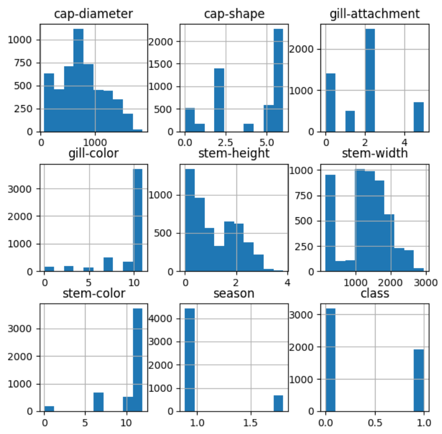

Fig. 1. Mushroom Classification

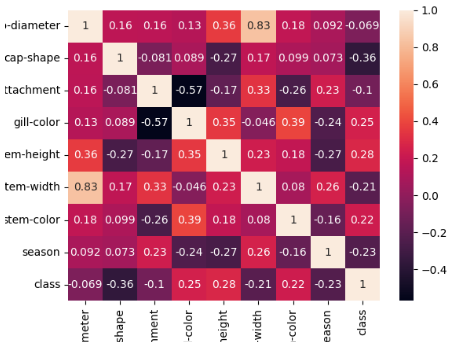

Fig. 2. Mushroom Feature Correlations

This is an interesting problem to experiment with because of the characteristics of the dataset. 1 shows a histogram of the columns. First, the dataset is relatively imbalanced. The target class is 62% edible and 38% inedible. Some features are imbalanced too, such as gill color and stem color. Second, the features have very different scales and distributions differ a lot among each other. And the data doesn’t have many outliers and noise. Third, from the correlation heat map2, we can see the target class is related to most of the features with over 0.1 correlations, except for cap-diameter. The features themselves sometimes have high correlations as well, such as gill-attachment and gill color.

As mentioned before, the data is pre-cleaned and it is not very noisy. So not too much pre-processing required from me. I applied standard scaling on the non-category features, since they are of very different scales, and some of the algorithms are sensitive to scales.

B. *Heart Failure Prediction*

The second problem is Heart Failure Prediction. Heart failure is one of the top causes of death globally. And people with higher risk of heart failure need early detection and management. The dataset contains 11 attributes that are considered useful in predicting a possible heart failure death event, the follow up time and the target is whether the patient has died. Among the features, ejection fraction, diabetes, serum sodium and sex are binary, others are continuous. The dataset has 2500 rows. And since follow up time is not a predicting feature, I excluded it from the plots and models.

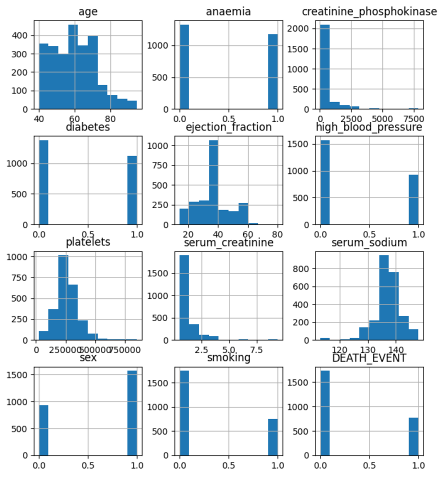

Fig. 3. Heart Failure Prediction

This problem is also interesting to look into because it shares some similarities with problem 1, but also have some differences. First, it is also a binary classification problem, but compared with the first dataset, this dataset has less data points, but more features. And this dataset has more binary features. Second, from the histogram, the target class is even more imbalanced, containing only 30% death events (1). And there are more outliers in the features. Third, from the heat map, the target event doesn't have too much correlation with many of the features. It only has larger correlations with age, ejection_fraction, serum_sodium and serum_sreatinine. Therefore, I'm interested in how different algorithms would deal with data containing irrelevant features. Same as dataset

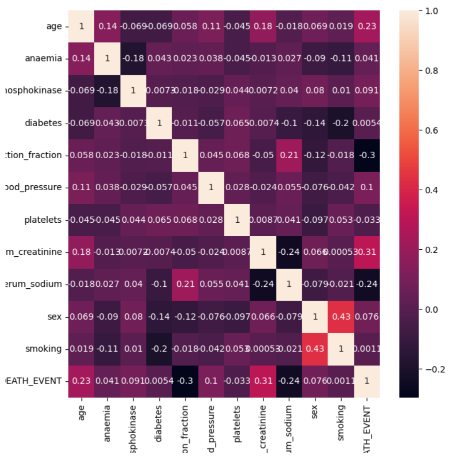

Fig. 4. Heart Failure Correlations

1, I did standard scaling on numerical features as they are of different scales.

C. *Hypothesis*

Based on the features of two datasets, I have two hypothesis: 1. Heart Failure Detection dataset in general requires more complicated models and more training time than mushroom data, as it has more complicated features and it is more imbalanced. 2. Both dataset has non linearity and hen Linear SVM won't perform well given the complexity of the datasets.

III. Methods

I investigated four supervised learning algorithms to approach the above two classification problems: Neural Network, Support Vector Machines (SVM), K-Nearest Neighbors (KNN) and Gradient Boosting. The two datasets are split into 80% and 20% as training set and test set. I used cross validation in training by equally splitting the training data to 5 pieces. Learning curves with respect to training sizes and training time are included to show the training progress and it also gives some hints about over-fitting. Validation curve with respect to different hyper-parameters are included to show hyper parameters tuning. The main metric used to measure the model performance is F1 score, as it is a useful metrics on binary classification and is very useful on imbalanced data as the dataset here. I also used confusion matrix as it provides a detailed breakdown of each group. The machine learning models in this paper all use implementations from Scikit-Learn package.

IV. Results

A. *Mushroom Classification*

I will show the results of mushroom classification from learning curves showing results from the training phase and cross validation; validation curves showing hyper parameter tuning, and prediction accuracy in below sessions.

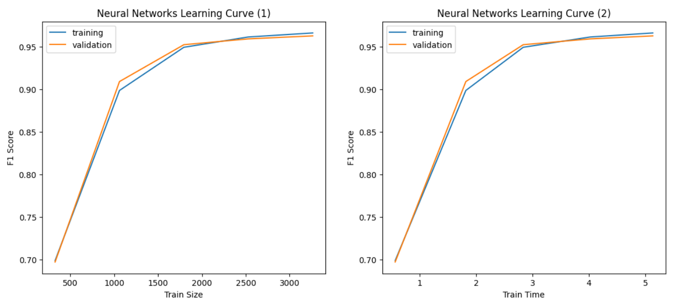

Fig. 5. Neural Network

1) Learning Curves: From the neural network learning curve5, we can see both training score and validation score have steep increases at the beginning. It increases from around 0.7 to above 0.9 in the first 1/3 training session. And as the amount of training data increases, both training and validation scores increase and stabilize at a high level at around 0.97. The learning curves indicate that the neural network model generalizes well and has a good balance between bias and variance. For the plots, hidden layer size is set to (40,40), which makes the model complex enough to learn the data, but in the meanwhile not too complex to cause overfitting. The validation score reaches to 0.95, showing neural network can model the complex relationships in the dataset.

And from the training time plot, it shows that the model takes sufficient amount of time to train. This is because neural networks is computationally intensive and often takes considerable time to train.

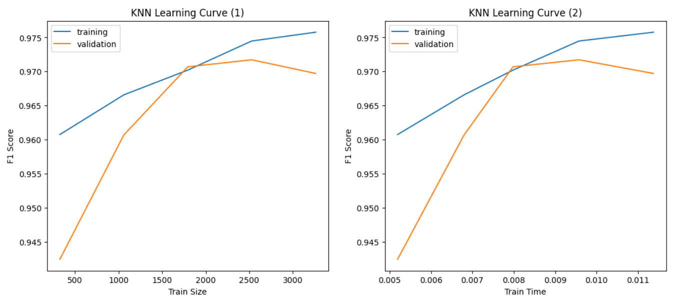

Fig. 6. KNN

From the **??**KNN learning curve, we can see that the training score and validation score are high from the beginning. They were both above 0.9 at small amount of training data. As more training amount, both training and validation scores increase until nearly half of the process when validation score flattens and eventually starts to decrease. The gap between the two scores starts to increase at the same time. This indicates overfitting happens during the training process. I chose K=5 for learning curve, and it is relatively a small value and that makes the model more sensitive to details as well as noise, which might lead to overfitting here. But overall, KNN achieves even better scores than neural network. I think one reason is that the dataset is relatively small for neural network, as it is a complex structure and requires large dataset. And KNN is also very flexible and robust to different distribution of the data.

Also, the training is very fast for KNN from the training time plot. This is because KNN simply stores the training dataset and doesn't involve explicit training phase. This saves significant amount of training time. Also, I found it is very sensitive to scaling as features with larger ranges can dominate the distance metric.

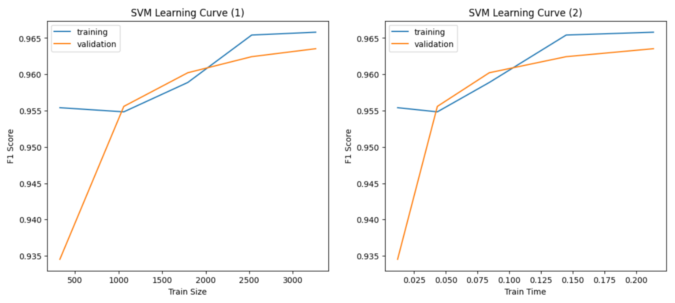

Fig. 7. SVM

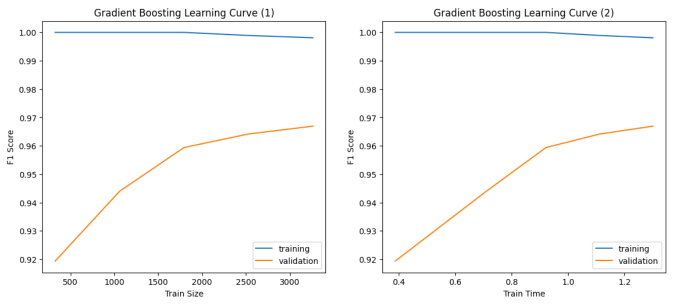

Fig. 8. Gradient Boosting

For the SVM learning curves, I used polynomial kernel with degree =10, which has the highest prediction accuracy among other choices of kernel functions. This indicates that the dataset is highly nonlinear. Similar as KNN's curve, the model achieved a good learning score (above 0.9) at early stage, and validation score increases very fast at first 1/3, when both curves reach 0.95. However, training curve is relatively flat initially, but starts to increase after some periods. I think this might be because degree=10 makes the model complicated and take longer for the model to learn and set up initially, which result in an initial flat curve until the model starts learning effectively. In the last 2/3 of the training session, both training and validation curve goes up and flattens slowly in the end, indicating that the model generates well.

From the training time plot, we can see it is also relatively fast algorithm. Although it is not as fast as KNN who doesn't require training phase, it is still quite efficient. Also, I noticed scaling makes a lot of difference in SVM, as the distance metrics would be highly affected by the numerical scales.

For the Gradient boosting decision trees, I used 50 week estimators, and max depth =7. I found the hyper parameters affect the performance a lot. Either too large or too small would harm the performance. We can observe that the training curve is almost flat during the whole training session. But validation curve keeps growing up. As the training score is almost 1 at start, meaning that it perfectly fit the data in the beginning, there's no much room for it to go up. But since validation score keeps going up, I think it indicates the model is improving and generalizing during the training process. Also, gradient boosting gives very high score, showing it is very effective here. And it is also not very sensitive to scales because of the feature of decision trees. Also, it requires a significant time to train the decision trees as well.

2) Validation Curves: The validation curve of Neural Network shows the training and validation error with respect to the hidden layer sizes. I tried for sizes = [(10,10), (20,20), (40, 40), (60,60), (100, 100)]. We can see that as size increases, both training and validation curve increase initially. Because larger sizes meaning more neurons and more learning ability of the neural network. However, after size is larger than (40,40), validation score starts to decrease. Training score goes up till 60 and then starts to decrease too. This indicates that the model gets too complicated and overfitting starts to happen. And as it gets more complicated, the model becomes less efficient and losses generalization.

Besides the hidden layer size, I also tuned the activation function, which applied to the output of each neuron to introduce non-linearity. I tried sigmoid, relu and logistic, and turned out logistic performs best among these. Therefore the validation curve is plotted while activation function set to logistic.

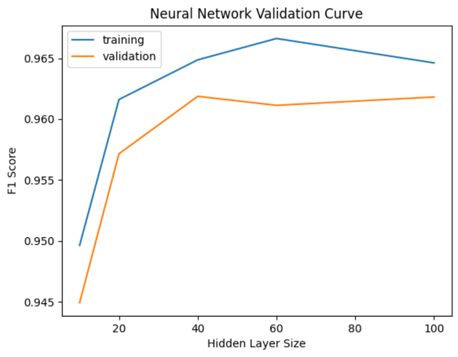

Fig. 9. Neural Network

For KNN, the validation curve is plotted against K, the number of nearest neighbors to consider when making predictions. The range included is [1,15]. From the plot, we can see the training score keeps decreasing as K increases. Because when K is small, the model might capture the fine details and decision boundary becomes more flexible, therefore the training score is better when K is small. However, small K makes the model more sensitive to noise and could lead to over fitting. And larger K can usually generalize better. This can be observed as validation score goes up as more neighbors considered. However, when K goes too large, validation score starts the decrease too and indicates underfitting. Therefore, K=7 looks optimal from the plot.

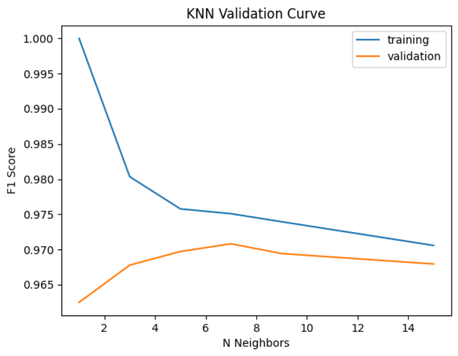

Fig. 10. KNN

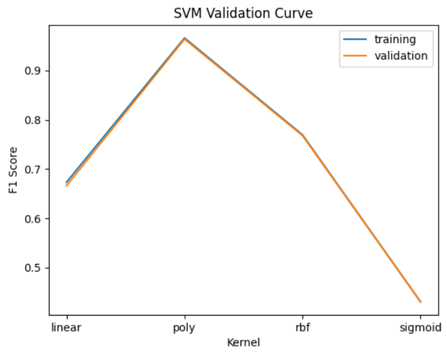

Fig. 11. SVM

SVM validation curve is plot against kernel functions. Kernel functions are able to transform the data into higher dimensional spaces for a linear separation. I tried linear, poly with degree=10, rbf and sigmoid. Training and validation curves almost fully coincide. Linear kernel doesn't perform well indicating the data isn't linearly separable; sigmoid doesn't perform well either as it resembles a neural network structure. Poly and RBF performs better as they can model more complex relationships. Polynomial performs best indication that the data has a polynomial relationship between the features. I also tuned the polynomial degrees and noticed that smaller degrees don't perform very well either. Only when the degree is larger than 5, poly starts to be significantly better than RBF.

For Gradient Boosting, validation curve is plotted against the number of week learners. The range I tested is [10,50,100,150,200]. From 10 to 50, both training and validation curves increase indicating that more weak learners, the model has better capability to learn and to explain the data. As the number increases, training score keeps going up but validation curve flattens. This indicates overfitting happens and the model is too complex.

I also tuned the max tree depth parameter. Default was 3

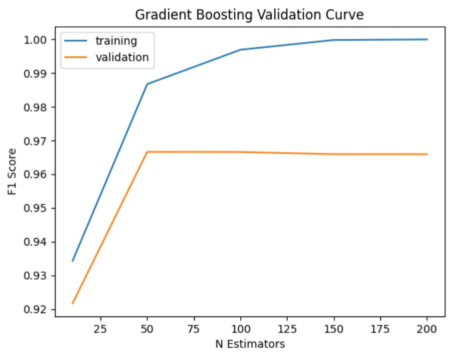

Fig. 12. Gradient Boosting

and I increased the depth, and I found the model can predict better as the max depth grows. Since the max depth makes the model more complicated too. However as it grows larger till 6, there starts to have some overfitting. So I end up setting max depth to 5.

3) Prediction Accuracy: We can see that final prediction accuracy of the four models are all above 96%. KNN has the highest score 97.5%, then gradient boosting 97.1%. Neural networks and SVM are very similar, both 96.6%. As the initial dataset is imbalanced, I also plotted the confusion matrices to show accuracy of each label. We can see prediction is higher for label =0 as there're more training data with label 0. Four models can all achieve about 98% accuracy. The differences of the models mainly show in the label =1. Neural network and SVM have lower accuracy. And it shows that KNN and Gradient Boosting are more robust to imbalanced dataset.

Model	F1 Score
Neural Network	96.6%
KNN	97.5%
SVM	96.6%
Gradient Boosting	97.1%

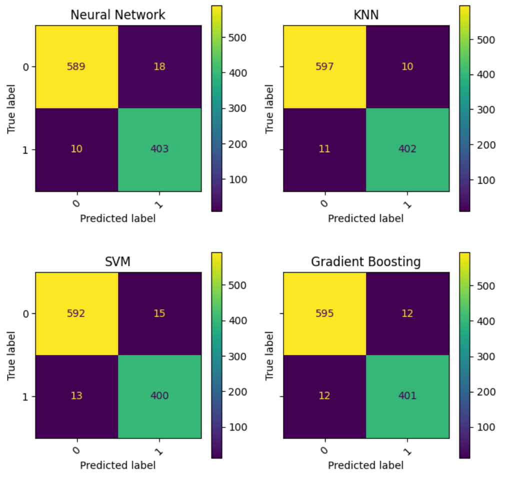

Fig. 13. Confusion Matrix

B. Heart Failure Prediction

Similar as the first dataset, I also show the results from learning curves, validation curves and prediction accuracy.

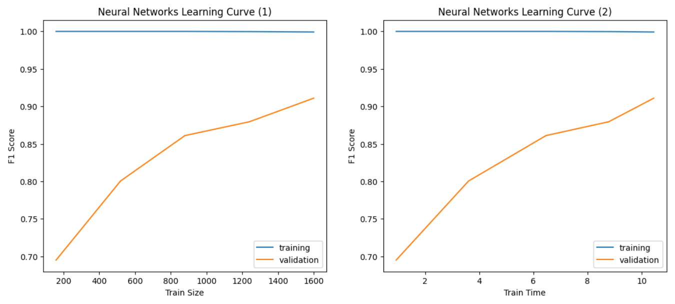

Fig. 14. Neural Network

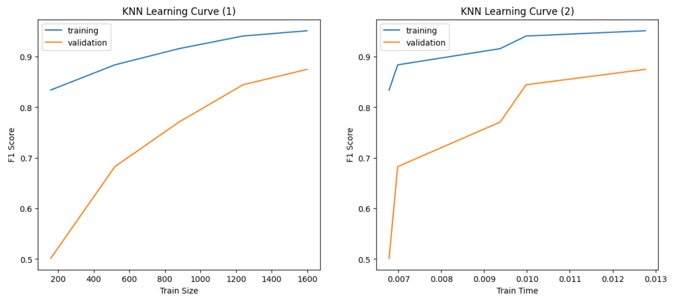

Fig. 15. KNN

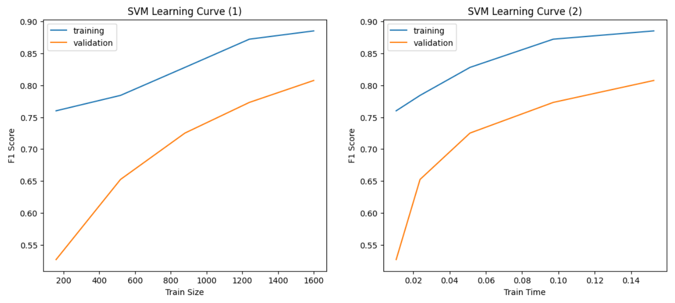

Fig. 16. SVM

1) Learning Curves: From the neural network learning curve, we can see training curve is flat during the whole training process. But validation score keeps increasing. I think this is because I used large hidden layer size (250*250) so that the model is complex enough to remember the training dataset. So the training score is almost 1 from the beginning, and not able to improve further. But as the training goes, weights are still being updated and the model generalizes well, so the validation curve keeps improving. Also, it takes even longer to train the model since I increased the hidden layer sizes.

From the KNN learning curve, I used K=2 as the dataset is noisy and might require a smaller K. As we can see, both training and validation score keep increasing during the whole process. And at the end, the increasing rate starts to slow down.

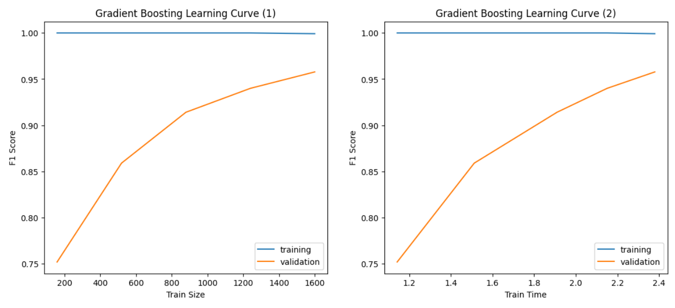

Fig. 17. Gradient Boosting

I think this indicates that the KNN model generalizes well and no overfitting happens although I chose a small K value. Also, very little training time is required because of reason we discussed already.

For the SVM learning curves, I used polynomial kernel with degree =4. Similar as KNN's curve, both training and validation curves increased during the whole training process, indicating that the training is effective and model keeps improving. But it doesn't flatten at end of the training period. I think this might indicate the amount of data isn't large enough for the model, so that the model isn't given enough training.

For the Gradient boosting decision trees, I used 200 weak estimators to ensure enough learning abilities. We can observe that the training curve is almost flat during the whole training session. But validation curve keeps growing up at a high rate. As the training score is almost 1 at start, meaning that it perfectly fit the data in the beginning, there's no much room for it to go up. But since validation score keeps going up, I think it indicates the model is improving and generalizing during the training process. Also, the validation curve doesn't flatten at end indicating that a larger dataset would help with the training. Similarly, the model takes long to train as it is complicated.

2) Validation Curves: The validation curve of Neural is plotted against hidden layer size. I plotted for sizes = [(50,50), (100,100), (150,150), (200,200), (250,250)]. Because I observed this dataset is more imbalanced and has more irrelevant features than the mushroom dataset, it requires larger sizes and more neurons to get the model performs well. We can see that as size increases, training curve first go up but then stops at a level. Interestingly, validation curve first goes down then come back up after (150,150).

For the activation function, I tried sigmoid, relu and logistic, and turned out relu performs best among these. Therefore the validation curve is plotted while activation set to relu.

For KNN, the validation curve is plotted against K, the range included is [1,15]. We can see the training and validation curve go straight down as K increases. Because when K is small, the model capture the fine details and decision boundary becomes more flexible, and since this dataset has more edge cases and outliers, smaller K is more ideal.

SVM validation curve is plotted against kernel functions. I tried linear, poly, rbf and sigmoid. Training and validation

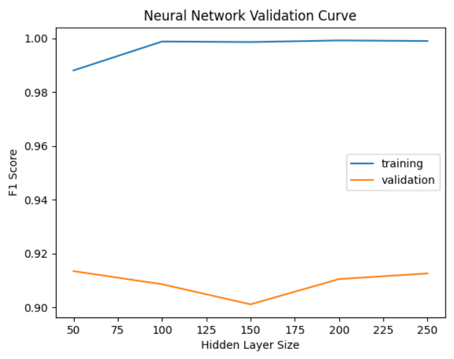

Fig. 18. Neural Network

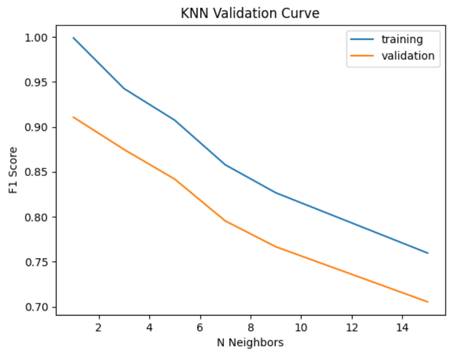

Fig. 19. KNN

curves both shows poly and rbf perform better than the other two, indicating the feature are not linearly separatable and require kernel to be able to model more complex relationships. RBF performs slightly better than Polynomial in validation score. But when I tuned the polynomial degree and found that degree=4 makes polynomial better choice than RBF.

For Gradient Boosting, validation curve is plotted against the number of week learners. The range I tested is [10,50,100,200,300,500]. We can see both curves keep increasing as N increases. Again this shows the dataset has more

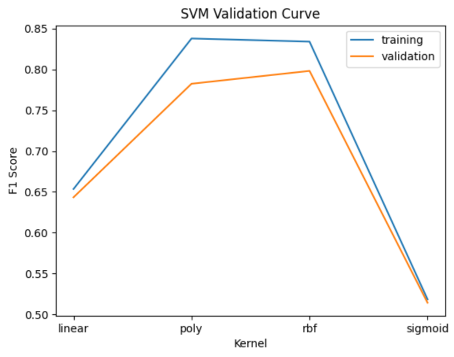

Fig. 20. SVM

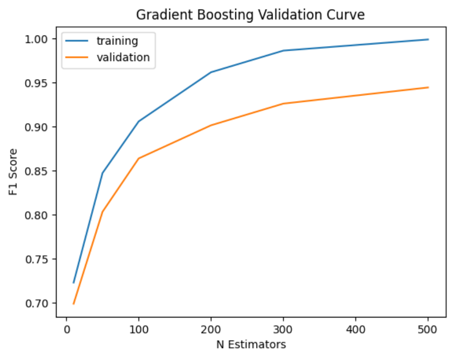

Fig. 21. Gradient Boosting

complex relationships and requires a more complicated models than dataset 1.

I also tuned the max tree depth parameter. And I found out max_depth also can greatly improve the model prediction. When I increase the max_depth from 3 to 6, and n weak learners = 200, it can already beat the n_weak_learner=500 model.

3) Prediction Accuracy: We can see that final prediction accuracy of the four models are not as good as the mushroom dataset. Neural Network and gradient boosting have best accuracy, 91.2% and 95.3%. Score of KNN and SVM are 88.7% and 82.7%. As the initial dataset is also imbalanced, I plotted the confusion matrices. We can see prediction accuracy is higher for label =0 as there are much more training data for label=0. SVM doesn't perform worst in label 0, while other models are similar. Differences are mainly demonstrated in label=1. Gradient boosting work best in label =2 and that's why it outperforms other models.

Model	Score
Neural Network	91.2%
KNN	88.7%
SVM	82.7%
Gradient Boosting	95.2%

V. Discussion

From the results, we have several findings. First, in general models have a better prediction accuracy for mushroom dataset than heart failure prediction. Also, for same type of models, heart failure prediction usually requires larger sizes of hyper-parameters and more complex model to achieve satisfactory results. This matches with the hypothesis one. I think it is due to the characteristics of the two datasets. Heart Failure prediction is more imbalanced, and it has more number of features, also more irrelevant features. And more outliers. This makes the dataset harder for machine learning models to be trained on, and therefore would need more complicated models to enhance learning ability.

Second, we can compare the behaviors of same type model on two different datasets. Neural networks perform relatively well on both datasets, because it can model complex non-linear

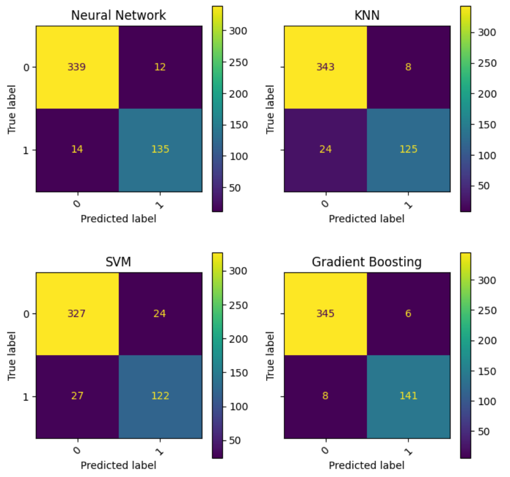

Fig. 22. Confusion Matrix

relationships between inputs and outputs. However, it requires a good amount of training data and I think a larger dataset would further improve its model ability. And it also takes longer to train than other models. KNN performs the best in the mushroom dataset but less accurate for the second dataset. I think this is because KNN is more sensitive to irrelevant or noisy features. Also, it performs poorly with imbalanced datasets as major class would dominate. Therefore, it doesn't perform as well on dataset 2. And I found it is very sensitive to scales because feature with larger ranges can dominate the distance. SVM doesn't perform very well on both datasets, especially for linear kernel. Because both datasets are not linearly separable. This matches with hypothesis 2. Though we can change the kernel function and map it to a higher dimension, it does effectively improve the performance. But still assumes that the data can be separated by hyperplanes in higher dimensions and it is not always the case. Lastly, gradient boosting behaves best in both datasets. I think it is because the model have ability to model complex relation-ships, and it can handle various data types well. And it is robust to overfitting. But in the meanwhile, it takes longer to train as well.

Third, since both datasets are imbalanced dataset, I found out that all models tend to perform better on the majority class. And the difference in predicting the minority class is more dominant in the model selection for a highly imbalanced dataset.

VI. Conclusion

Supervised learning on widely applied on real life classi-fication problems. In this paper we discussed various types of supervised learning algorithms and applied them on two different problems. I took steps including understanding the features of the two datasets, pre processing the data, training the models, evaluating their performance and tuning hyper-parameters. The results are discussed and analyzed from various aspects, for example learning curves, validation curves, prediction accuracy, etc. In conclusion, each algorithm has its own assumptions and preferences, and also its advantages and disadvantages. This tells me how crucial model selection is for machine learning. In order to find an appropriate model for a given dataset, we not only need to identify a model that performs well on training data. We also need to consider how it generalizes effectively to unseen data, and also consider its training time, computational complexity, difficulty of hyper parameter tuning, etc. All of these requires a deep understanding of the algorithms and their underlying theory, as well as enough amount of experiments.

References

[1] https://www.kaggle.com/datasets/prishasawhney/mushroom-datasethttps://www.kaggle.com/datasets/prishasawhney/mushroom-dataset

[2] https://www.kaggle.com/datasets/rithikkotha/heart-failure-clinical-records-dataset