Assignment 1: Supervised Learning

[Mingrui Sun]
[msun333@gatech.edu]

I. Introduction - Dataset Explanation

UCI adult dataset(1)[Ein05] and dry bean dataset(2)[**Koklu**] were chosen as a playground to show case difference in each supervised learning model. The adult dataset consist of 48842 entrances of 14 features of typical personal information(i.e. sex, workclass etc) and a binary class of whether that person earn more than 50k or not. The Dry bean dataset consist of 13611 entrances of 17 beans shape information (i.e. length of a bean, area of a bean etc) and label of sex class of beans. Three reasons make these two datasets interesting for our purpose in this report:

1) Both dataset can be considered as large high dimension dataset. These feature make it particularly good to showcase run time difference(whether training or running) and capacity of each model. Also they are both unbalanced dataset like most real world case.

2) Data wise these dataset contain different kinds of data type: adult dataset contain mostly category features when dry beans dataset contain all numeric features.

3) The dry bean dataset contain four shape forms feature which were a synthetic features coming from existing feature. For example shape factor 1 is just (length of the bean)/(Area of the bean) with both length and area are preexisting feature in the dataset. Literature shown these four synthetic feature added domain knowledge so resulting better modelKoklu. It would be interesting to show whether different model needed that extra kick or not - A method that have high capacity may not need those extra synthetic feature to perform the same as with it added.

4) The adult income dataset is interesting as the distribution of personal income happen in " the strange country of extremistan", in which the observed average of income for a group of people can be change greatly depending on how many outliter you sample(me and Bozos's income average would be Bozo's income). The average national household income in the year of the dataset was around 35k which is way below the 50k label threshold. So the 50k label could be really turn as label of rich person(outlier) and a average or below person(income wise). The problem with predicting things in extremistan is that the model may be good to predict the average case but not our outlier, as a luck may play a bigger role in these ourlier. As literature[**Kohavi**] shown, the best model(C4.5 decision tree) used on this dataset only got a 85% accuracy when guessing all people are below 50K would give a 75% accuracy. So this dataset would be considered as a hard dataset which would be great to show worst case for all these ML method.

In this work, all datasets were spilt into training and test set in a 80:20 spilt with set random number. The same train/test spilt was used on all model. A 5 fold cross validation was done on the training set(80 part) to retrieved the best hyper parameter setting for each model-the setting that have the closest performance between train and validation set plus the highest overall performance was selected. For adult dataset, all entrances with missing value were discard. For beans dataset, models built with and without synthetic features were also compared.

II. Decision Trees(DT)

DT model was implemented using sklearn tree.Decisiontreeclassifier method. The tree was built in a determined fashion which the tree always spilt on the best feature base on the select metric since for a single tree that normally perform the best(forest or bagging would prefer the random way). To get the best performance, different **splitting criterion** and **post purring** parameter were tested on the training set. For splitting criterion, gini index, entropy and log loss were all candidate. The splitting criterion was selected by comparing the validation curve of the each criterion with the same post purring alpha set to zero. After criterion got set, only post purring was done on the tree, i.e. the tree was built without depth or leaf number limit. For post-purring setting, different alpha(impunity penalty) level was tested in the following process:

1)DT models were trained with the whole training set with various alpha to plot effective alpha with the resulting tree impunity. Figure 1 is a example of such plot on the adult dataset, as it shown the possible range that alpha could have. As effective alpha increase, more nodes get purred which resulting in higher total impurity of the tree. In figure 1 case, effective alpha reach maximum at around 0.06 as the tree become a single node which is not interesting. Also another interesting to know is the figure 1 have square increase step kind of shape, in which change in alpha in each step actually don't change the impunity level as the number of node to purring have to be a integral number. So the space of alpha to explore is not continual but only the edge of each "step" except the single node tree case.

2)Using the resulting alpha candidate space, validation curve on alpha was then generate on the 5 fold cross validation train-cv/validation spilt. For each alpha value, the average of 5 spilt score was taken as score for each alpha. A validation plot was then generated for alpha, the alpha with highest average cross validation score and best agreement between validation and training score was then selected as the alpha

use in final model.

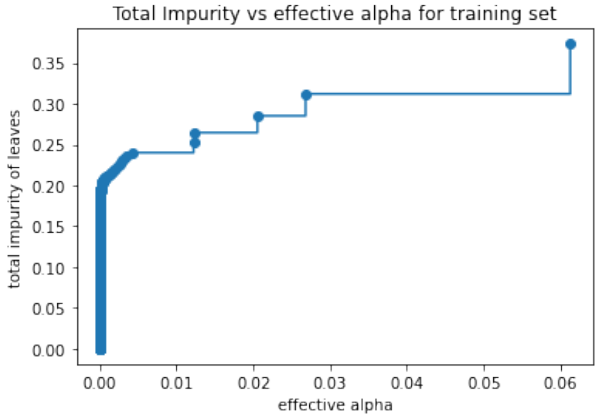

Fig. 1. Impunity pathway of effective alpha on the adult dataset

A. *Adult Dataset(1)*

As suggested in the introduction, the adult dataset was a unbalanced binary classification problem, so weighted f1 score was selected as the scoring metric for this dataset as it account for dataset imbalance.

Figure 2 shown the 5 fold cross validation test score and training time for all three splitting criterion. Entropy and log loss ends up have better f1 score compare to gini index. This result is not surprising as the adult dataset contain mostly categorical feature so entropy and log loss(information gain) would performance better than the mostly for continual value's gini index. log loss(information gain) and entropy ends up performed exactly the same both time and f1 score wise, indicating their high similarity in this binary classification case. Another interesting to note is that gini index's tree consistently trained faster than entropy/log loss suggest gini index to be less computational expansive(p*p is faster than p*log2(p)).[**QuantDare**] This train time difference may be importance when dataset size increased dynamically so you may choose gini index only based on train time concern. For this work, the entropy was selected as the default as f1 score of the resulting model was important.

Figure 3 shown the 5 fold cross validation test score and train score for different cost complexity purring alpha(penalty for node number) level. As alpha increase, more of the tree's branches were removed so problem with overfitting decrease(the training score more closer to validation score). In order to achieve the best performance without too much fitting, the peak of the validation curve was then used as the best alpha setting which give a 0.85 f1 score.

With all hyperparameters selected, a learning curve on different training set spilt was generated in figure 4. Though validation and training score reach agreement in the end, at the first half of the training sample, the our DT model show severe sign of over-fitting as it reach perfect 1 f1 score when started. This shown that even with purring, DT model

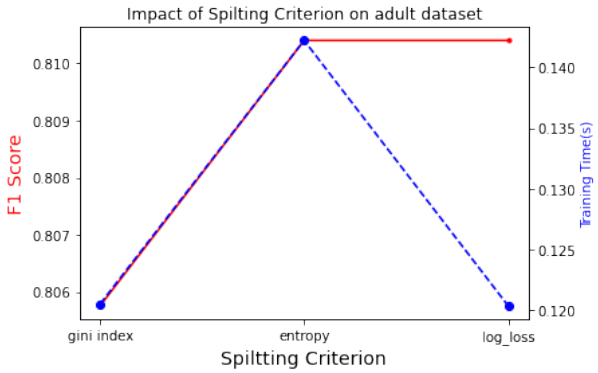

Fig. 2. Cross Validation Performance of different Splitting Criterion

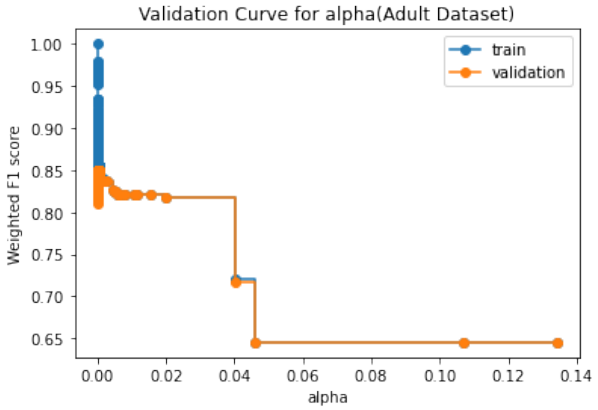

Fig. 3. Validation Curve for Post Purring Alpha Level

is still prone to overfit and require all data available to battle that. The final DT model for adult dataset was then trained with all training data and then tested with the test dataset which take 0.2s to train and 8.5s to predict the test label. As common knowledge, construct a balanced binary tree take time complexity of O(n_sample*n_feature*log(n_sample) and query time would be O(log(n_sample). The DT tree would take noticeably less time predicting than trained. Overall this model achieved a 0.8541 training F1 score and a 0.8495 testing score which indicate a tiny amount of overfitting. The snip shot of the resulting DT tree was shown in figure 5, though this is a pretty large tree with a max depth of 7 so it is hard to fit the whole tree in the figure, every node decision in the tree can shown as if statement that human can understand. This is contrast to more complex model shown later that works more like black box so user cant understand.

B. *Bean Dataset(2)*

A similar validation curve for spilting criterion was generated as figure 6, OG here stand for original which is used to distinguish the future all synthetic features removed beans dataset. Since each label of bean is equally important, a weighted f1 score was used as performance metric. Figure 6 shown similar result as figure 2 as entropy/log_loss achieved better scoring when gini index run faster. Though scoring

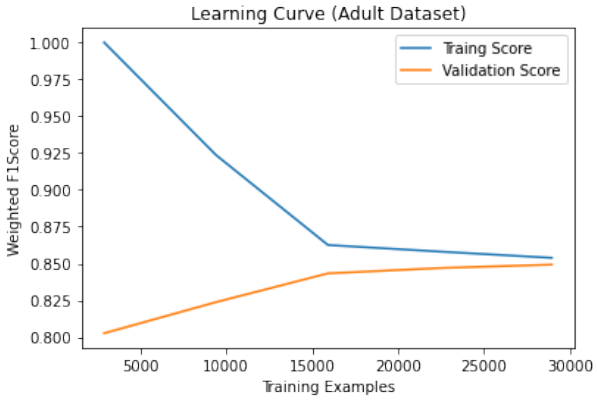

Fig. 4. Learning Curve of DT model on adult Dataset

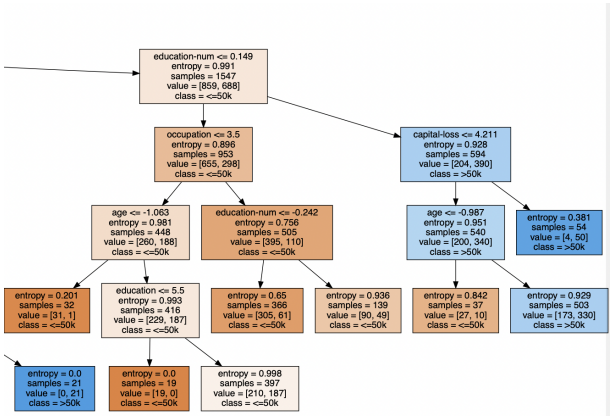

Fig. 5. A piece of the resulting DT model for adult dataset

wise entropy/log_loss looks like better choices, gini index was used as the criterion for beans dataset as all features in it is continues numeric value. In addition to that, the beans dataset have 17 dimension which make run time of model a bigger concern.

The validation curve for complexity purring setting alpha was then generated as figure 7. Unlike figure 3 where the peak of the validation curve also have the best agreement between training score and validation, the peak of validation curve(alpha = 0.00038) in figure 7 dont have the best agreement with training which suggested the extent of overfitting is high. In this case validation score was sacrificed a bit for better agreement, a alpha of 0.0013 was used as it maintained high validation score but training and validation curve converge to closest. The same alpha selection process(not shown) was also done on the bean dataset without synthetic features, a alpha of 0.0012 was selected for the without synthetic feature vision of bean dataset. As number of features decrease, it make sense to lower the purring setting.

With all hyperparameters selected, the learning curve of both OG bean dataset and bean dataset without synthetic feature were generated in figure 8. Shape wised these two learning curve looks similar as the learning and validation curve start to converge together after seeing around half of the training set. But the DT model that train on the OG dataset achieve better higher validation score(0.9) when synthetic features removed's model have lower validation score(0.88) but achieved better generalization(training curve reach closer to the validation curve). It turns out that the added synthetic features do increased our model performance since more domain knowledge is added. DT is a relative weak model compare to all other model in this works so it need that extra kick from added domain knowledge to help. One interesting experiment to study is if we just have duplicated features in the dataset, for example instead of having A/L(ratio of A and L) feature, we just have duplicate A and L features in the dataset, would that still build tree with similar performance to the tree that trained on dataset with A/L feature? By doing this experiment, we can show whether changing weight on different feature can fake our expert domain knowledge. But the experiment is not ran here due to time limit.

Overall, the DT model on the beans dataset with all features take 0.2s to train(whole training set) and take 10.4s to predict the test data. Overall the model achieved a 0.9092 weighted F1 score on the test and 0.9359 weighted F1 score on training set.

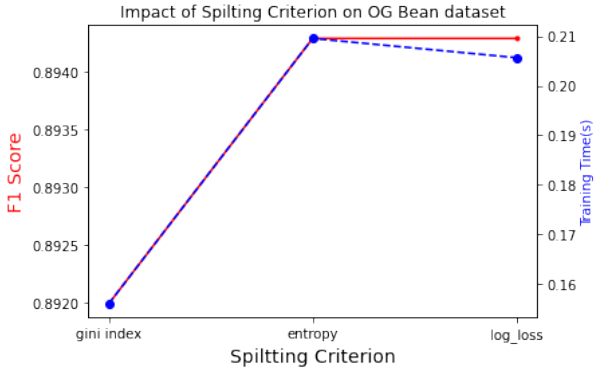

Fig. 6. Impact of Splitting Criterion on DT performance

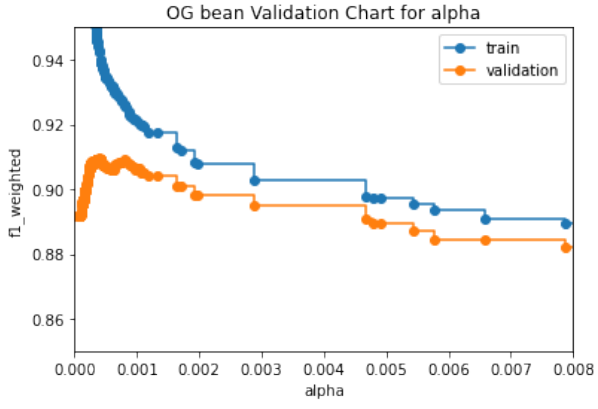

Fig. 7. Validation Curve for alpha on OG bean dataset

]

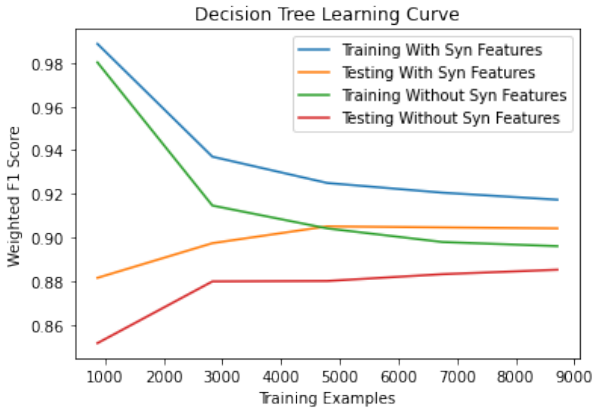

Fig. 8. Learning Curve of DT model (With/Without Syn Features)

III. KNN

KNN models were implemented using sklearn neighbors.KNeighborsClassifier. For both dataset, a uniform weight were used as distance weight result in serious over fitting issue. The **K** value was selected through constructing validation curve using 5 fold cross validation on the training set. The best K value was then selected based on both validation score and train/validation agreement.

A. *Adult Dataset(1)*

The adult dataset contained mostly categorically feature which make hamming distance as the KNN parameter the best choice. Manhattan and euclidean were both tried but both result in sub 0.7 weighted f1 score which were not preferable. Validation curve on the numbers of neighbours k was generated as figure 10. As k increase, training score curve and the validation score curve converge together as KNN model generalize better. When k = 1, the 1NN would predict whatever closest point label, which result in horrible overfitting that f1 weighted score go to perfect. As k increase more point are taken into account so problem of overfit diminish. At the extreme that k = number of sample in the training set, the model would then would always predict whatever the highest probability label is which would achieved guessing performance. So if k value is being explored extensively(cover all space), the validation score curve would increase then decrease at some point and reach flat guessing performance. Unfortunately adult dataset contained some numeric features which dynamically increase the time that require to generate validation chart in larger space. Figure 10 take 10mins to generate with only k = 1,10,50,100,500,1000 being explored. The long run time maybe a result of using hamming distance metric in a dataset that have both categorical and numeric features as switching back to euclidean distance do speed up the training. A better way to build KNN from this dataset maybe using custom distance metric which calculate euclidean distance for numeric feature and hamming distance for categorical features and then combined them into a new distance. But the custom distance idea haven't been implemented due to time constrain.

In the end a k of 100 was selected as it achieved the great model generalization in the same time being the closest to the peak of the validation score curve. It is preferable to explored k more around k=100 to achieve better KNN performance but it is also not done due to time issue.

Overall, the resulting KNN model achieved a 0.8166 weighted F1 score testing wise and 0.8266 score for training. Score wised it show nearly no sign of over fitting as training score and testing score is nearly identical. A learning curve was generated for the result KNN as figure 11. As figure 11 shown, the KNN is pretty data hungry as it basically require every bit of data for the training curve to converge the testing curve. Even with the whole data set, they still don't match. Run time wise it take 0.2s to train and 4.8s to generate prediction for the testset. The training is notability faster than any model in this work since KNN training essentially was just recording all data(O(1) complexity. But querying take a lot longer as for each point you need to calculated distance to every data point in the training set. Unlike most model that training take longer than predicting.

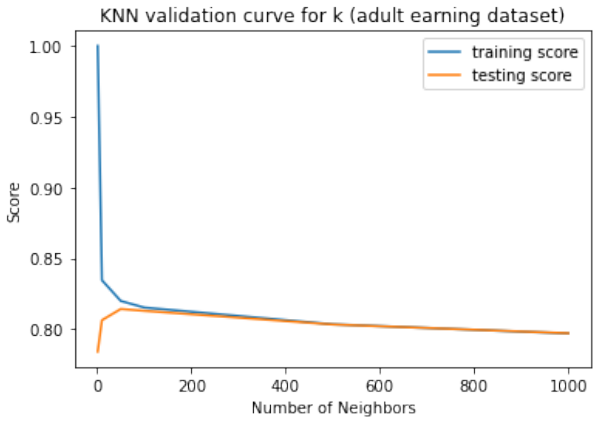

Fig. 9. Validation Curve of k on Adult Dataset

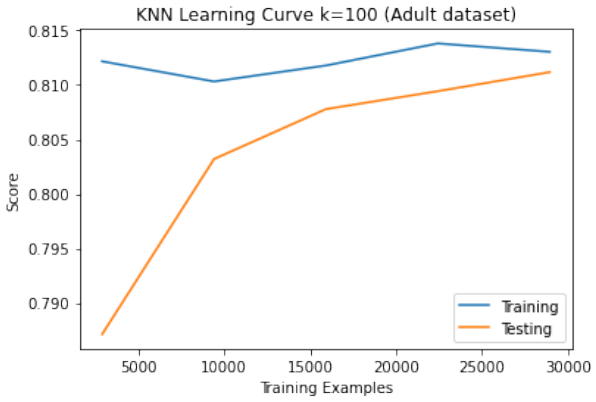

Fig. 10. Learning Curve of K=100 KNN on Adult Dataset

B. *Bean Dataset(2)*

Validation curve on Bean training set were generated on different k using both Manhattan and Euclidean distance. It turns out the resulting validation curve were identical so Manhattan distance was select for all bean dataset analysis as it is the least expansive computing wise. Validation curve for k on the OG beans dataset and synthetic feature removed dataset were shown in figure 12 and figure 13. Synthetic feature bumps up the validation performance of the KNN model, so domain knowledge for these feature do help also in the KNN case. But if time or storage complexity is more of a concern, it do make sense to use the dataset with all synthetic feature removed. When increasing k, test/validation score first increase then decreased like suggested before. In the ends a k value of 30 was selected due similar logic describe before.

Overall, the resulting KNN model achieved a 0.9237 weighted F1 score testing wise and 0.9236 score for training. A learning curve was generated as figrue 14. It shown the data hungry features of KNN as the same plot for last section. It require all the data to get a good generalization. Run time wise it take 0.3s to train and 0.1s to generate prediction for the testset.

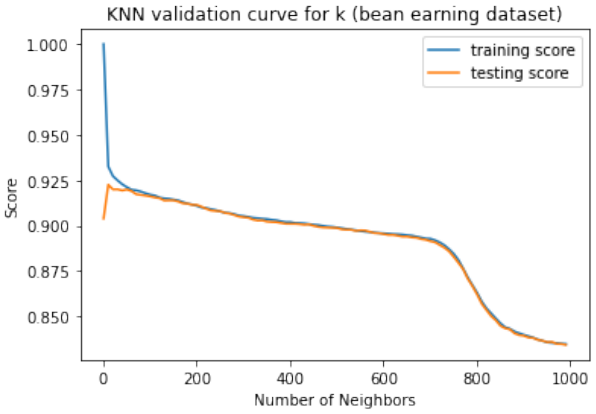

Fig. 11. Validation Curve of different K(OG Bean Dataset)

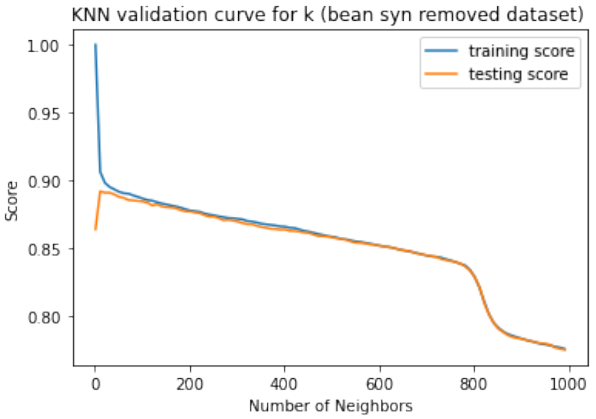

Fig. 12. Validation Curve of different K(Synthetic Feature Removed Bean Dataset)

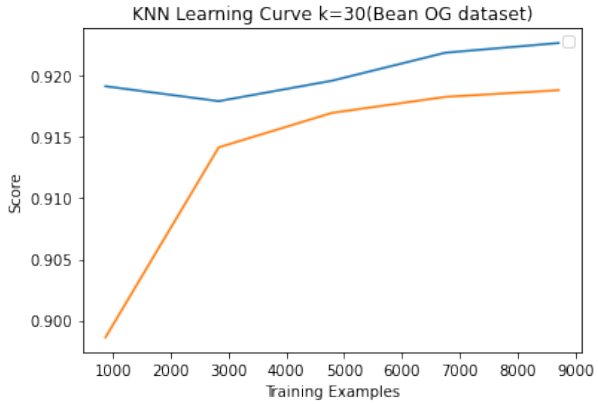

Fig. 13. Learning Curve of k=30 KNN model on OG Bean Dataset)

IV. Adaboost with DT

Adaboost DT model was implemented using sklearn ensemble.AdaBoostClassifier method. Unlike DT model that utilize one large DT tree, Adaboost DT built a serise of smaller DT tree(weak learner). The weakest learner that Adaboost can make use is a stump which is a tree with only one node and two leaves. What's special about Adaboost is the training process, unlike random forest that all trees are trained on the same training set, each tree in adaboost is trained one by one on a changing sub sample of the training. For the first tree it would be trained on equally weight sample of the training set. The weight of data in the training set was then reset to make data that first tree do bad on have higher weight. The next tree is then trained on the sub sample of this newly weighted training set. Also a voting weight is assigned to each tree depending on total error rate on the data it is trained on. In the end a series of trees were trained with voting weight relate to it. The logic behind adaboost is a merit weighted version of "wisdom of the crowd", a "smarter" person's opinion(low error rate tree) deserved more weight than normal person's opinion(high error rate tree).

In this section, Adaboost DT model trained were performed on Adult and Beans datasets. Splitting criterion were all set to gini index as it take the least time training. In Adaboost, optimizing performance of each tree become less important since it can still work even with weak learner. This can be shown as in figure 15 which the learning curve for each criterion merge together and were too similar. This shown a major benefit of Adaboost that most time user could use it out of the box and still get reasonable performance. For all trees in Adaboost model in this work, their are all decision stump(max depth =1). The only hyperparameter selected to tuned in this work is the **number of estimator** and the **splitting method**(spilt on the best feature or spilt randomly select feature) for each dataset.

A. *Adult Dataset(1)*

Validation curve of of different spiltters(random or best) and different number of trees were all plotted in figure 16. As

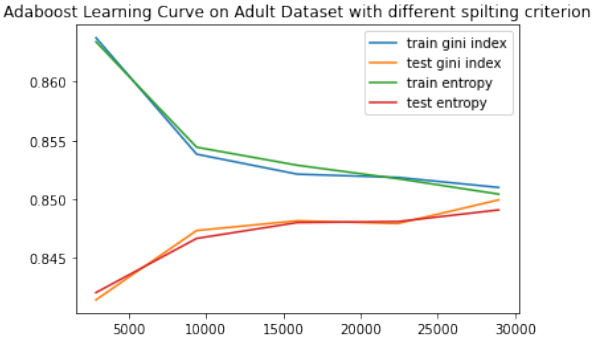

Fig. 14. Entropy vs Gini Index for Adaboost model on Adult Dataset, it don't make much difference in the Adaboost case.)

the figure shown, the spiltters don't have any impact on the validation score and agreement level between training score and validation. When using the best feature to spilt, the tree build have bias towards trees that have the lowest depth. Since for Adaboost all trees only have a depth of 1(in our case), it wont make any difference. Using random spiltter can decrease the training time as the algorithm don't have to calculate gini index to rank features each time. Also the added serendipity factor make random spiltter less likely to overfit. So in the end, random spiltter was selected as the spiltter of choices.

On the number of estimator side, unlike other learner in this work that the training score start as overfitting and then decrease to converge with validation score, Adaboost validation and training score both increase as number of estimator increase. Though the training score was always higher than test score, the extend of this overfitting don't increase as more tress is added to Adaboost. Adaboost seems have the ability suppress overfitting even when the model capacity increase. Another interesting note on the Adaboost is the out of the box performance mention before, once the number of estimator reach a certain size(like 30 in figure 16), the classification performance stay great. Increase in number of estimator in those case do increase performance but the difference is not that important. The default setting of number of trees in sklearn adaboost model is 50 and it seems to work perfectly for our case. So the default setting was selected. It would be better if a bigger number of tree was selected but it would increase model run time without much increase in performance.

A learning curve was generated using hyperparameter decided before as figure 17. Adaboost achieved low variance from the start with low number of training example seems(only a 0.03 difference on the training score and testing score). Seems like Adaboost model converge faster compare to learner discuss before. Overall the final Adaboost model that trained on all training set achieved a 0.8421 weighted f1 training score and 0.8407 testing score(on the whole test set) in the end. The variance is so low that it is hard to tells. Run time wise it took 0.3s to train and 0.5s to predict the test case. The prediction time for Adaboost model is a lot faster than the tuned DT model for the same dataset, as went through 50 decision stump was a lot faster than querying binary tree with 18 level of depth. Overall Adaboost achieved the best overall performance in classifying Adult dataset so far.

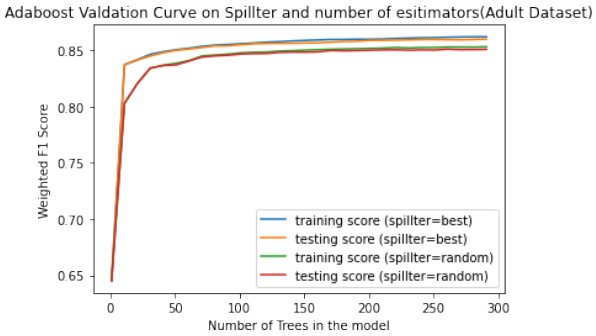

Fig. 15. Validation Plot of number of estimator for Adaboost(Adult Dataset)

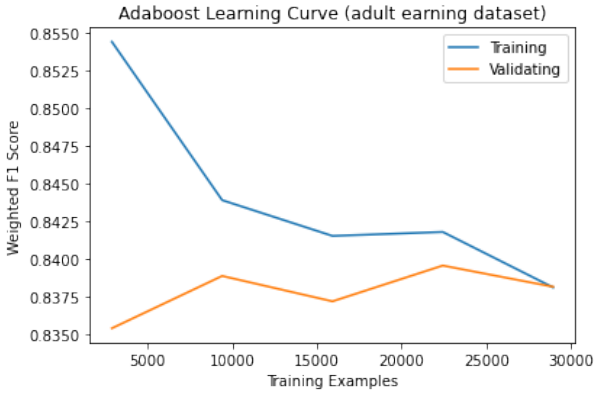

Fig. 16. Learning Curve of Adaboost Model(Adult Dataset)

B. *Beans Dataset(2)*

Validation curve of of different spiltters(random or best) and different number of trees were all plotted in figure 18. In the Beans dataset case, the spliter do matter as random spilting significantly increase the weighted F1 score for both training and validating compare to the best feature spiltter. So random spillter was also chooses as the spiltter of choice for Adaboost. In terms of number of tress impact on model variance, no matter which spillter used, the model achieve low various through out number of tree space. Once number of estimators(trees in our case) reach a certain point like 60, the validation performance plateau. So just like in the adult dataset, using the number of estimators of 50 was enough for the Bean dataset. The validation score(0.76 weighted F1) shown in figure 18 is signficiantly worst than tuned DT model shown in 8 or 9(around 0.88 weighted F1), so seems like for bean dataset, the Adaboost dont quite performed out of the box.

In order to further tuned the Adaboost model, the max depth level of the DT tree was increase. Validation curve for different max depth value was generated as figure 19. As the number of

depth increase, training and validation score both increase as capacity of the boosting model increase. But at around depth of 3, variance between training and testing increased indicating overfitting. In the end a depth of 3 was selected as it strike good balanced between low variance and high validation score. With hyperparameter all figure out, the last problem to discuss was synthetic features in bean dataset. To see whatever they made a difference or not, a learning curve charts were generated in figure 20 with Adaboost(one trained with all features but one trained without synthetic features) as in figure 20. Synthetic features did give a kick to Adaboost model performance as it increased validation score of the model by 0.05. The added domains knowledge of these feature were helping. Learning curve of the Adaboost learner showed that the model retained low variance from the start and didn't shown any sign of overfitting at all. Overall on the original beans dataset, a 0.8479 training score and 0.8489 testing score was achieved with Adaboost. It took 0.7s to train and 0.4s to predict the test set. Though score wise this may seems a bit low compare to DT trees' 0.90 testing score, the Adaboost model still did better than DT overall as it reach better data generalization with unbeatable low variance.

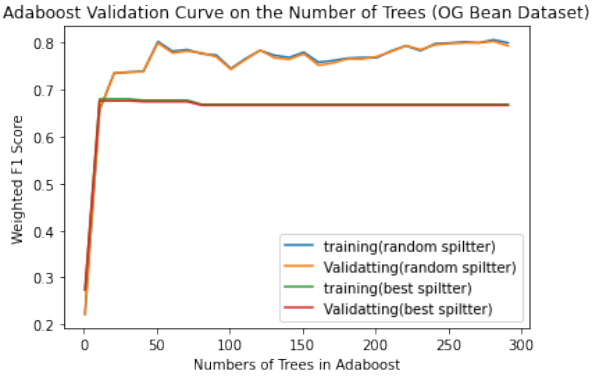

Fig. 17. Validation Plot of number of estimator for Adaboost(Bean OG Dataset))

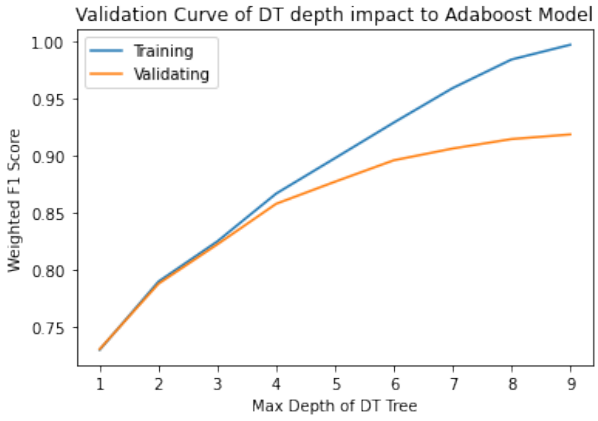

Fig. 18. Validation Curve of Depth of DT tree's impact on Adaboost performance))

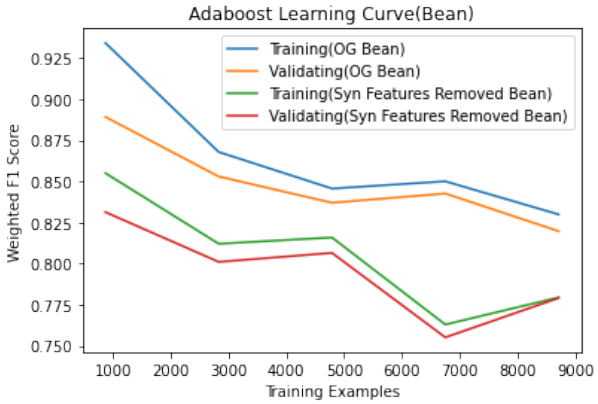

Fig. 19. Learning Curve of Adaboost Model(Bean Dataset)))

V. SVM

In this work, the support vector machine learner was implemented using sklearn svm.SVC method. In classification, SVM construct lane(for 2D dataset) or hyperplane to separate different class in the space. The shape of this line or hyperplane was defined with the kernel kind used in the SVM which make kernel selection the most import hyperparameter for SVM. Generally speaking, linear kernel is the default choice for most case, but it have a training time complexity of $O(\max(n,d)*\min(n,d)^2)$ where n and d are number of training sample and features respectively.[**Chapelle**] When n and d number explode like in a high dimension large dataset, the linear SVM kernel can take days to train. So for both dataset in this work, a non linear kernel radial basis function(RBF) as non linear kernel normalily have training complexity between O(n2) to O(n3).

A. *Adult Dataset(I)*

Linear kernel was first tried on the adult dataset, but it take more than an hour to train so it had been abandon as a kernel choice for this dataset. A learning curve using RBF kernel was generated as figure 21. Well the result looks terrible compare to every model in this work. It turned out that the encoding method for categorically features could have huge impact performance of the SVM model. Before, all numeric features in the dataset is all being standardized(value between -1 and 1) but not the categories features(which is all integral like 1,2,3). This data encoding problem ends up make categorical features have more of a say in the SVM hyperplane optimization process, which cause huge model bias. After all features being standardized, a new learning curve was generated with balanced training data as shown in figure 22.

One problem to discuss is the impact of unbalanced dataset on SVM model, the training set used for figure 22 was an unbalanced dataset as more ¡=50k class were presented. A balanced training set was generated by sub sampling the unbalanced dataset to the same sample number but with 50:50 class mixed. The resulting learning curve of the balanced

SVM was shown in figure 23. Comparing figure 23 and 23, we can see that the balanced training set didn't improved the validation score but only decrease model variance at low training example. It turns out for the imbalance of this dataset(75:25 label ratio) wasn't that big of a deal for SVM. But it would be preferable to have balanced training set as it flat out noise in learning curve due to low chance of sampling minority case at low training example see.

One interesting thing to note is the impact of features standardization on SVM fitting speed for linear kernel. Liner kernel was first tried without standardizing categorical features which result in never ending training wait time(longer than 1 hour, training session was abandoned so no exact train time recorded). After all standardization issue fixed, a learning curve was generated for using linear kernel as figure 24. It only took around 3mins with all 5 fold cross validation done. Literature shown that linear SVM was a degenerate version of the rbf SVM[**Keerthi**] so prediction wise it would never out perform rbf SVM. Our result mirror this literature statement as validation score wise the rbf did have better score than linear SVM. Training speed wise, the learning curve require 3m 32s vs 3m 41.3s for linear vs rbf. Though linear kernel could have larger than $O(n^3)$ complexity, it can be faster than non linear kernel rbf assumed data is linearly separable. For the adult dataset, rbf was used as the kernel of choice.

Overall, the rbf kernel SVM achieved a test 0.8346 weigthed F1 score and 0.8448 training score. The model took 17.8s to train and 3.2s to predict the test set.

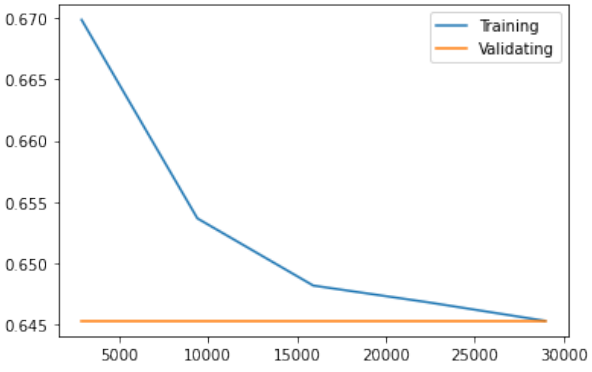

Fig. 20. Learning Curve of SVM(rbf kernel))

B. *Beans Dateset(II)*

The impact of different kernel on Beans dataset was shown in figure 25. RBF SVM constantly out performed linear SVM in this case which mirrored the literature statement cited before. For both kernel choice, SVM all reach low variance between training score and testing. RBF was then selected as the kernel setting to use for bean dataset.

The impact of synthetic features on SVM model was shown in figure 26 as two SVM model was trained on the same training set with and without those features. For SVM, synthetic features manage to also boosted validation score by around 0.02 similar to what it done for other model discussed in this report. One interesting to note was that this performance boost matter the most the smaller training set used, as in figure 26 the synthetic features' testing curve variance with without synthetic features testing decrease as both model saw more training example.

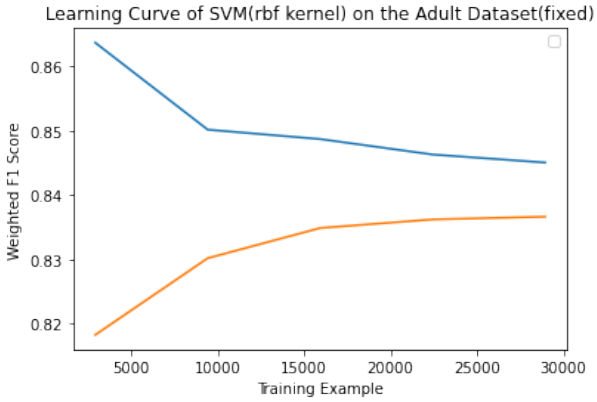

Fig. 21. Learning Curve of SVM(rbf kernel) on Adult Dataset(fixed))

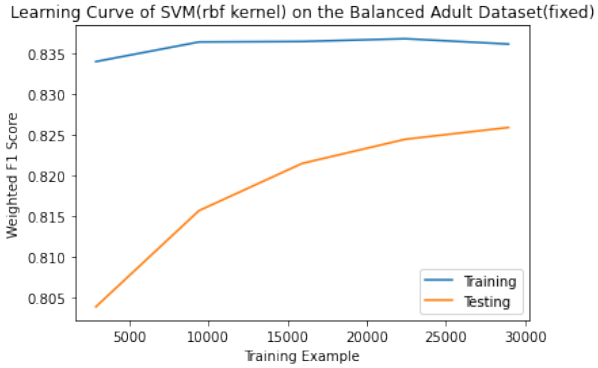

Fig. 22. Learning Curve of SVM(rbf kernel) on Adult Dataset(Balanced))

In the end SVM model using rbf kernel achieved a training weighted f1 score of 0.9317 and a testing f1 score of 0.9346. The testing score is a hair higher than the training score which is more likely causing by luck with the train/test spilt. Overall SVM model achieved high score with low variance. Run time wise the model took 0.4s to train and 7.9s to predict the test set.

VI. Neural Networks

Now we reached the last learner for this report, the fun one, the one and only neural network(NN). Compare everything else for this report, NN can have highest freedom to customize which lead to nearly infinity amount of hyperparameter to tune. So to make sure this report can by submitted before due day only the simplest NN model- Multilayer perceptron was implemented for this section using the sklearn neural_network.MLPClassifier. Multilayer perceptron network consist of layers of fully connected neurons like what shown in figure 27[**MLP Sklearn Doc**]. Each neurons takes the last layer output and do a linear combination of all output received

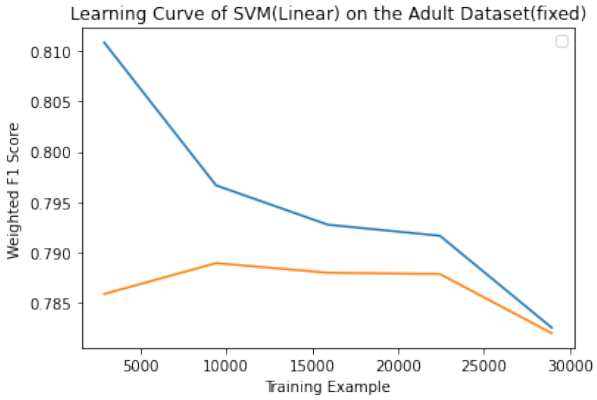

Fig. 23. Learning Curve of SVM(Linear kernel) on Adult Dataset)

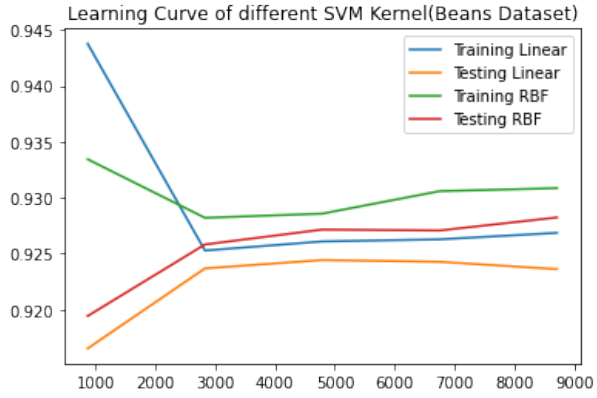

Fig. 24. Learning Curve of Linear SVM vs RBF SVM)

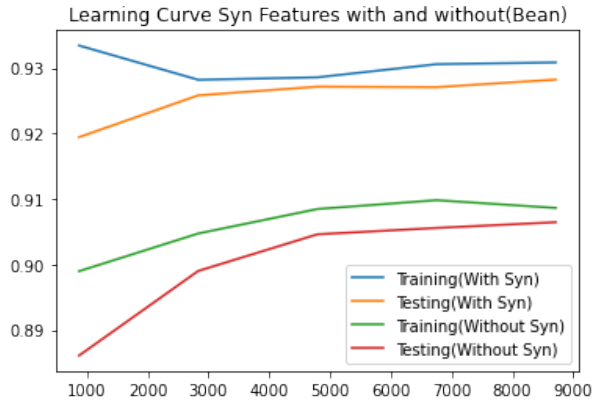

Fig. 25. Learning Curve Syn Features or Not(Bean))

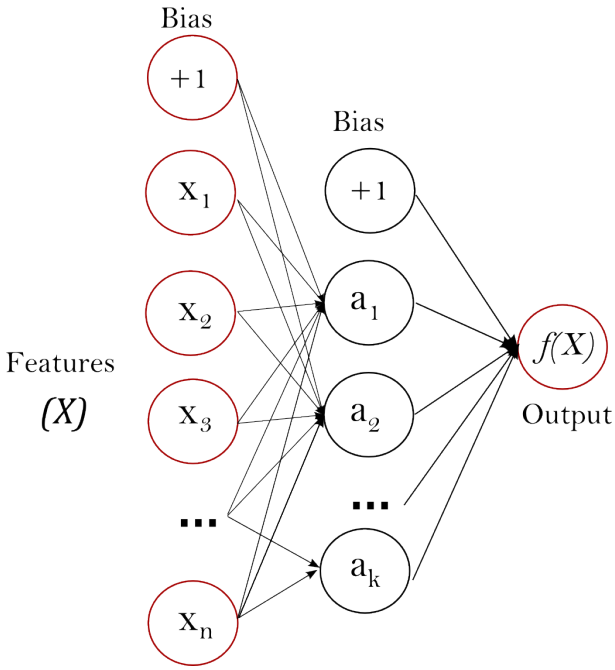

Fig. 26. Example of MLP with one hidden layer(cite sklearn doc))

using its own weight and compare the result to its own bias to decide whether a zero(below bias) or one(above bias) is generated as output to the next layer. For classification problem, the output layer(last layer in MLP) would have every neuron corresponding to every class to predict. So whether the neuron in the output layer output 1 or 0 would indicate whether a sample is in a class or not. The comparing to bias in each neuron is called a activation function which is a non linear of user choices. So though MLP use linear combination in each neutron, the final relationship it can represent can be non-linear as non linear activation is used.

As suggested before, the MLP have tons of hyperparameters to tune. For this project, i focus on only the structure of the MLP network and the learning rate alpha. The hidden layer structure was first tuned assumed a fixed learning rate of 0.001. Then the learning rate was further tuned with the selected hidden layer structure. For all experiment here, adam solver, Relu acitvate, early stop with a 0.1 validation set were all used. The early stop was used to reduce overfitting when training.

A. *Adult Dataset(I)*

Using a fixed learning rate of 0.001, impact of different hidden layer structure were shown in figure 28. The figure was plotted in ways that the increase in x axis is corresponds to increase of complexity of the MLP model(more layer+more neutrons). As the hidden layer complexity increase, the training score increase but the test/validation score stagnate even decrease. From (8) to (16,16,16), the means of the validation score stay mostly the same but the standard deviation varied. In the end (16) was selected due to its highest validation score and variance to the training score. The worst case performance(bottom of the std range) of it is better than almost everything in this range. Loss curve of the selected hidden layer structure was plotted in figure 29. Under all learning rate, loss curves were well behaved as it were all smooth(no oscillations) and reach a low loss plateau. To large learning rate like 0.1 get stop too early and to small learning rate like 0.0001 get struck in a local minimal for loss. Both are unfavorable due to the lack of proper exploration of the space. The sweet point for learning rate is 0.01 as it reached the lowest loss in

training so it was selected as the learning rate to used for this dataset.

The learning curve of this MLP model was shown in figure 30. The tuned MLP model for adult really require more than 1/3 of the training set to reach a low variation between training. As more learning sample added in, the train/validation score variance did decrease and reach the minimum when all sample added. In the end a training weighted f1 score of 0.8427 and testing score(on the test set) of 0.8413 was achieved in the final model. Run time wise it take 15.6s to train this MLP model and 7.7s to predict the test label. Since this MLP model only have one hidden layer, it is actually a linear classifier which is different from everything else in this project.

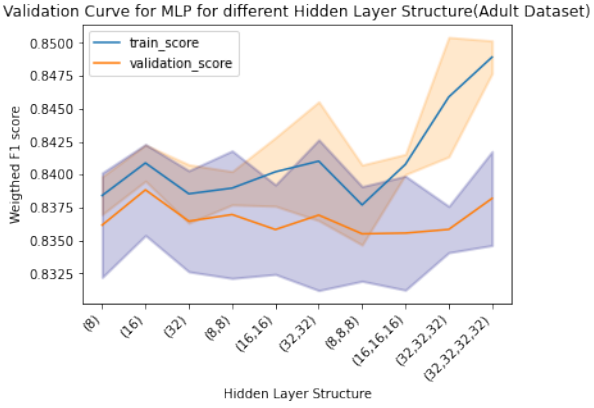

Fig. 27. Validation Curve for different hidden layer structure(Adult), color area around each curve represent the one std range of the means.

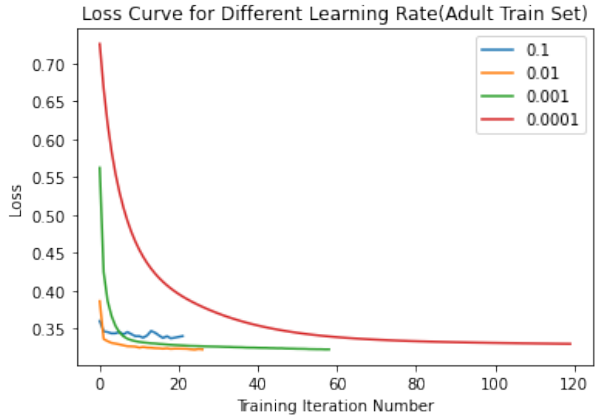

Fig. 28. Loss Curve of the MLP Model Trained with Different Learning Rate

B. *Beans Dateset(II)*

Impact of hidden layers structure on MLP model were shown in figure 31 using a fixed learning rate 0.001. The peak of validation curve is (32,32,32) and (32,8). In the end (32,8) was choosed due to better worst case validation score(the bottom of the blue STD range) and smaller variation between train and validation. Loss curve of this structure under different learning rate is plotted in figure 32. The only learning curve

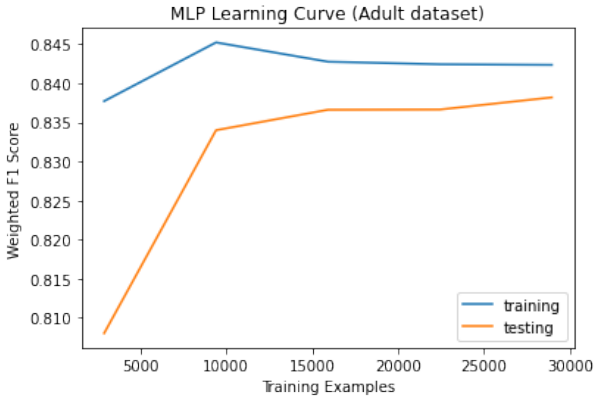

Fig. 29. Learning Curve of MLP model on Adult Train Set

here is not well behaved was the 0.1 learning rate one as it was big fluctuation in loss. 0.01 seems to be the sweet spot as it reach the lowest loss compare to all learning rate in the range. A learning rate between 0.1 and 0.01 may do the job better but due to time limit space of learning rate is not fully explored. (32,8) and learning rate of 0.01 were then used as the hyper-parameters for final models.

The effect of synthetic features on MLP model was investigated by building two MLP model for beans dataset with or without synthetic model. For the model that without synthetic features, a (25,8) hidden layer structures was used instead of the (32,8) as 7 features were removed. Loss curve of these two model were shown in figure 34. As the loss curve chart shown, the model can reach lower loss by having synthetic features. This result also are shown in the learning curve comparison in figure 35 which the with synthetic features model have better validation score and test score overall. So in the MLP case, synethetic features with domain knowledge do help the model's performance. This comparision here have fault that as we didn't tuned hidden layer structure and learning rate more as the data changed, but it still provide some evidence that synethtic features do boost model performance. So for our final model the model that trained on all features was used instead of synthetic features removed one.

With all hyperparameter selected, the final MLP model was built by training with the whole training set with all features. As learning curve in figure 34(the with Syn Line) shown, the MLP model have reach great generalization(low variance between training score and validation) even with lower number training example seems on this dataset. In the end a training weighted f1 score of 0.9359 and testing score(on the test set) of 0.9306 was achieved in the final model. Run time wise it take 16.5s to train this MLP model and 12.7s to predict the test label.

VII. Conclusion

Performance of all tuned model in this work were summarized in the Figure X. The test score of models on Adult dataset was in order of: DT > Adaboost> NN(MLP)> SVM> KNN, and for beans dataset: SVM> NN(MLP)>

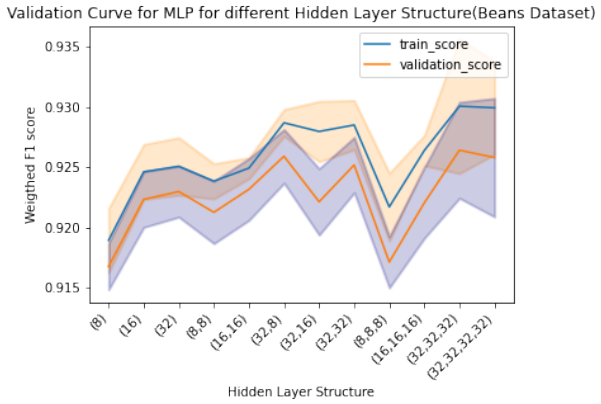

Fig. 30. Validation Curve of MLP model on Hidden Layer Struture(Bean Dataset)

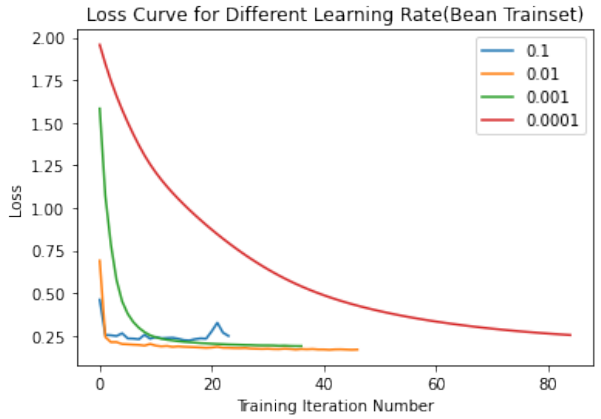

Fig. 31. Loss Curve of MLP model Training under Different Learning Rate(OG Beans Trainset)

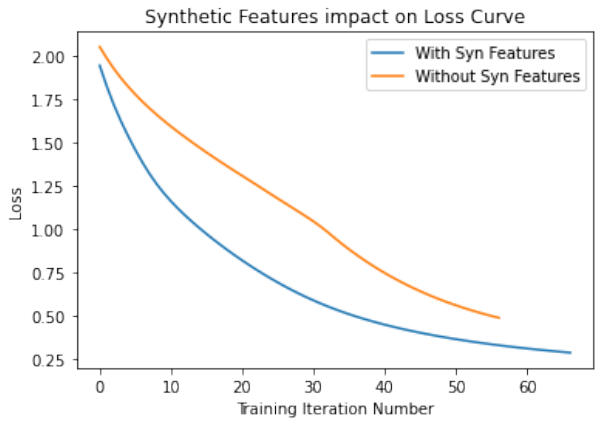

Fig. 32. Loss Curve of MLP model With/Without Synthetic Features

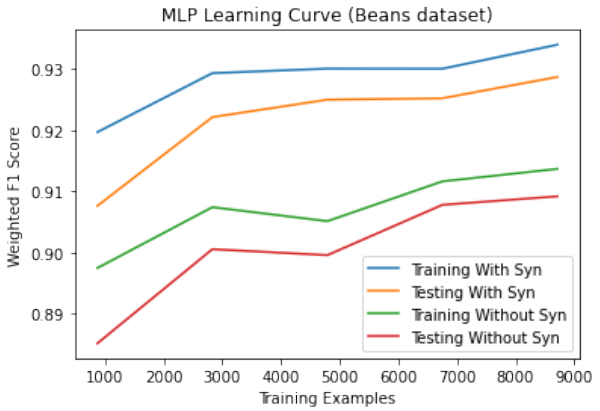

Fig. 33. Learning Curve of MLP model With/Without Synthetic Features

KNN > DT > Adaboost. This performance ranking seems to be really data specify. The Beans dataset seems to be a harder problem as it models performance varied more in it. This is probably due to the fact that beans problem consist of more features(dimension) and more label than the adult dataset which is binary classification. From a data generalization stand point, Adaboost have low variance in both dataset which showcase Adaboost low susceptablility to overfit. In terms of training time, SVM and NN(MLP) model particular have high training time. This would make these two kinds of model less scale able training wise when apply to large number of data. Overall, all models in this work reach a respectable performance for these two dataset. The model selection process is highly data depend, so one may need to try everything to make a decision. One interesting thing to note is that among all model in this work, Adaboost require the less amount of tuning, low train time, low scoring time and also reach a reasonable performance(not the best). Adaboost would be always my first choices when it come to a random classifcation problem to work with. On the other hands of the spectrum, NN models always seems to can achieve good score, have a lot hyperparameters to tune, have high run time overall. So NN would always be my last bet when a state of the art performance is required for some reason(TBH I cant think of one as SOTA are mostly unrealistic).

For the synthetic features problem in beans dataset, all model shown a performance boost consistently when the synthetic features were added. This result showcase the power of domain knowledge based feature engineering when dealing with real world data. The more domain knowledge we can transfer to model the better performance it would ends up perform. But this is all assumed those domain knowledge are actually real as bad/fake domain knowledge would increase bias for the model. One way to test the impacts of fake domain knowledge would be randomly have duplicate features in the data or random ratios of existing features and then compare the resulting model performance to what it was before. But this test wasn't done in this work due to time limit.

Overall, in this work. DT, KNN, Adaboost, SVM and NN(MLP) models were all implemented and tested in two dataset. Limited number of hyperparameters was tuned to show its impact for each model. It all reach reasonable

performance and get the classifcation problem done. As a practitioner, the model selection process should be done in a case by case way as accuracy/precision requirement, data availability, training device fire power(computational), scoring device fire power(computational), run time requirement and deadline for the project are all concern in the real world.

Adult Dataset	Training Score	Test Score	Variance (Train/Test)	Train Time(s)	Score Time(s)
DT	0.8541	0.8495	0.0046	0.2	8.5
KNN	0.8266	0.8166	0.0100	0.2	4.8
AdaBoost	0.8407	0.8421	-0.0014	0.3	0.5
SVM	0.8448	0.8346	0.0102	17.8	3.2
NN(MLP)	0.8427	0.8413	0.0014	15.6	7.7
Beans Dataset					
DT	0.9359	0.9092	0.0267	0.3	17.7
KNN	0.9237	0.9236	0.0001	0.3	0.1
AdaBoost	0.8479	0.8489	-0.0010	0.7	0.4
SVM	0.9317	0.9346	-0.0029	0.4	7.9
NN(MLP)	0.9359	0.9306	0.0053	16.5	12.7

Fig. 34. Summary of Model Performance in this Work

VIII. References

References

[Ein05] Albert Einstein. “Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]”. In: *Annalen der Physik* 322.10 (1905), pp. 891–921. DOI: http://dx.doi.org/10.1002/andp.19053221004.