CS7641 Assignment 1 – Shangci Wang

swang879@gatech.edu

1. Introduction

Customer churn refers to the loss of customers after certain period. It is a major pain point for many companies including my current company since it is much more expensive (sometimes 20 times more) to acquire new customer than to retain consumer [1]. In dynamic environment, causes for customer loss could be complicated, such as dissatisfaction, limited accessibility, competitive products, etc. There could be tremendous amount of customer data inside company that can be leveraged to help understand reasons for their leaves. But such data is challenging to analyze due to its complexity and limitation in tool, resources, capacity, etc. Machine learning is very powerful in data mining and modeling, which has strong potential in customer churn prediction. This topic is interesting to me because it closely relates to my current job and is highly relevant to various industries. I selected customer churn data from two business areas: Telcom and Bank, aiming to analyze how well different algorithms would tackle this churn prediction question. It would be interesting to compare the same business question between two completely different industries. Moreover, Telcom set has much higher dimensions (40 vs 13), yet half size of datapoints compared to Bank set, It would be interesting to see how this characteristic would impact model performance among various algorithms.

2. Methods

This assignment was completed by using Python with Spyder IDE. Telcom Customer Churn dataset and Bank Customer Churn dataset were downloaded as csv file from Kaggle, and further formatting steps were performed before model analysis: 1) drop data points with any missing value; 2) randomly assign 70% data points to train set with fixed positive to negative ratio for training and testing set; 3) convert categorical variable to dummy variable; 4) except for decision tree (DT) and AdaBoost decision tree (ABDT), all variables were normalized to [0,1].

A total of 5 algorithms were investigated for both datasets. Accuracy instead of error rate was implemented in this study. For KNN, SVM, DT, and ABDT, scikit learn library was utilized. 1-3 key hyperparameters were selected for each algorithm and a wide range of values per hyperparameter were applied if relevant. Effects from individual parameter on train and 10-fold cross validation (cv) accuracy were explored to guide model optimization. Validation curves were visualized to screen initial models. To select value for multiple hyperparameters, validation curve approach was started from one hyperparameter with all the others at default value. After evaluation, the first hyperparameter was fixed at its best value for the second hyperparameter exploration. In the case of ABDT, grid search was applied in tandem with validation curve to decide best parameter combination. Initial model screening normally results in a few promising candidates whose learning curve was then plotted to understand influence of training size on training accuracy, cv accuracy, and fitting time. As for neural network (NN), Tensorflow library was applied. All the NNs were trained by using Adam optimizer with epsilon at 0.005 and sigmoid for output layer. A different type of learning curve was used to evaluate performance of different NNs. Changes in accuracy, recall, precision and loss for train and cv (20% split) were plotted over 200 epochs with batch size of 32. For all algorithms, hyperparameters generating superior performance (discussed later) were used to train the final model and to calculate 6 metrics for cross-model comparison, which were accuracy, precision, and recall from train and test set. Fitting time for final KNN, SVM, DT, and ADBT models are obtained from learning curve, while that for NN is calculated separately and present in Appendix. Large number of plots

are generated in this assignment, so figures are resized to fit in space limit. Appendix in this link include figures in original size for better visulization.

3. Results and Discussions

3.1. KNN

Fig. 1 a-d show the interaction effects of neighbor number with weight function or distance on training and 10-fold cross-validation (cv) accuracy for **Telcom dataset**. In general, weight function seems to have largest impact on model performance, followed by neighbor numbers, while distance choice fails to show any significant effect. When used unfirm weights (regardless of distance choice, Fig. 1a &1c), small neighbor number generates higher training score (~0.85), but its corresponding cv score is noticeably lower (~0.75); as neighbor number increases to ~ 20, training and cv score converge and including more neighbors barely affects model performance. Normally at small neighbor number KNN model tends to overfit for edge samples; as k number increases, model takes more data into consideration and becomes less complex and more flexible, but using too many neighbors could negatively impact model performance. Inverse distance weighting gives closer points more influence in label prediction, but its implementation results in significant overfitting issue in this work (Fig. 1b & 1c): training scores are constantly 100% with neighbor number varying from 1 to 100, but validation scores are less than 0.8. Likely closer points in Telcom dataset are not always more similar to each other, so using inverse distance would increase edginess.

KNN model with uniform weight and 70 neighbors shows the highest validation score with no difference between Euclidean and Manhattan distance. Learning curve is plotted for individual distance option. Learning curve for Euclidean (Fig. 1e) and Manhattan distance (Fig. 1f) are similar to each other. When training examples are less than 1000, severe overfitting issue is noticed (training: 1 vs. cv: < 0.3). This overfitting issue is gradually mitigated as more training examples are included, finally training and cv curve merge at over 3000 examples. Further increase in datapoints boosts accuracy for both training and cv sets. It seems that both curves have not reached to stability point yet, suggesting additional training examples might improve KNN model performance. Training time also slightly increases as size of training set (blue line in Fig.1 e & f), but it's worthy to mention that fitting time for KNN is less than 0.05s for ~4.5k examples, which is significantly faster than most algorithms tested in this assignment. This makes sense because KNN is a "lazy" algorithm that spends most time in query not in training.

Manhattan distance is finally selected due to recommendation from Aggarwal and others [2]. This paper states that in high dimensions all the points tend to be uniformly distant from each other and such phenomenon is worse when distance is measured on Euclidean distance compared to Manhattan distance. This might also explain why unfirm distance works better than inverse distance weighting in my case since Telcom dataset has 40 features. Parameters for final KNN Telcom model are 70 neighbors with uniform weight contribution calculated by Manhattan distance. Performance for all the final models is listed in Table 1 and discussed in 3.5.

Fig. 2 a-d show the validation curves for **Bank dataset**. The effects of weight function, neighbor number, and distance choice on train and cv scores resemble observations from Telcom dataset, likely resulted from the abovementioned reasons. KNN model with uniform weight and 15 neighbors has the highest validation score, which is selected to plot learning curve with Euclidean and Manhattan distance applied respectively. It is interesting to see more differences between two distance options (Fig 2 e vs. f). Effect of training size on fitting time remains the

same for both distance choices. With Manhattan distance, increase in training size largely improves cv accuracy and reduces the gap between train and cv scores. CV curve becomes stable at end, while train curve is still rising. In contracts, with Euclidean distance there is always a gap between train and cv curves. Adding more data points after 4k examples tend to further expand the gap. Given the shape of training curve, both KNN models tend to slightly overfit the training set and overfitting issue is worse for Euclidean distance. Thus, final parameters for KNN bank model are 15 neighbors with uniform weights and Manhattan distance.

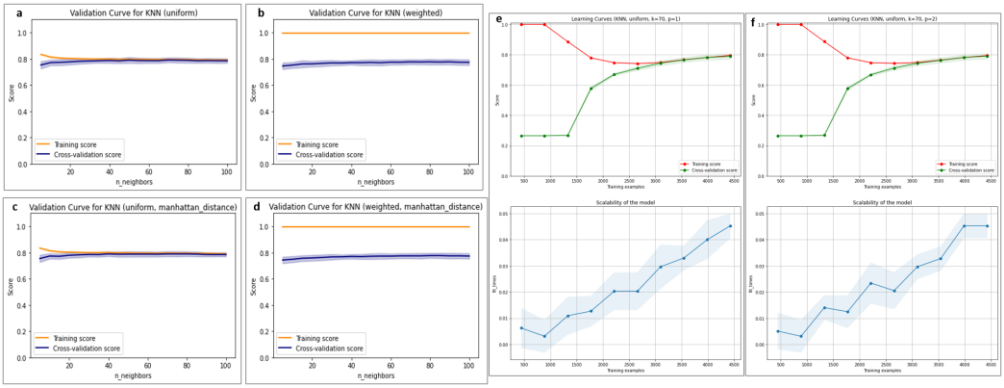

Figure 1. Telcom KNN model validation and learning curves

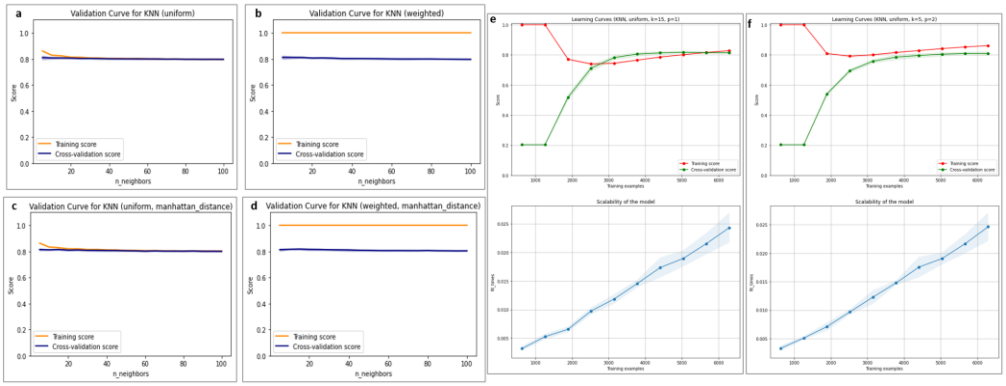

Figure 2. Bank KNN model validation and learning curves

3.2. SVM

Linear, rbf, poly (degree=3) kernels were explored. To speed up training process, only 20% data was randomly selected for initial screening. After hyperparameter optimization, all train data were used to plot learning curve and train the final model.

Fig. 3 a-e describe the influence of single SVM hyperparameter (plotted on log 10 scale) on the **Telcom data** training and cv accuracy. C controls margin of hyperplane. Larger C puts more penalty on SVM for misclassification, which narrows margin of the hyperplane, encourages less error, but typically increases fitting time. With linear kernel (Fig 3a), small bump in train and cv accuracy is noticed as C changes from 10^{-3} to 0.1, despite most accuracy scores are similar for a wide change of C (10^{-5} to 10^{3}). Therefore, kernel trick was utilized. With rbf kernel, training curve reaches to high accuracy (~1) quickly as increase in C, but validation curve lags, which indicates model overfitting (Fig 3b). C=1 shows the highest validation score and is fixed for gamma exploration. Gamma defines how far the influence of a single example reaches. Very small gamma leads to support vector influences data points far away. With

contribution from too many data points, SVM model cannot precisely define shape of decision boundary. Very large gamma generates support vector with small influence that fails to generalize. This well aligns with findings from Fig 3c where rbf model has lower accuracy at small gamma but overfits at large gamma. Overfitting issue is also found for large Cs and gammas in poly kernel (Fig 3d-e). It is unexpected that kernel trick fails to better classify the examples that are not linearly separable. Likely dimension of this dataset is too high for rbf and poly (degree=3) kernel to effectively separate classes. Learning curve was then plotted to further compare three kernel methods. For linear (Fig 3f) and rbf (Fig 3g) kernel, both training and cv accuracy converge to ~0.8 with ~1000 training examples and more training data fails to benefit much. In contrast, a noticeable gap consistently exists between the training and cv curve for poly kernel despite the gap is slightly narrowed by adding more data. As for training time, rbf and poly models take ~0.1 to 0.15s longer to fit than linear one. Linear kernel with C=1 is selected for Telcom SVM model due to better generalization and higher validation score.

Fig. 4 a-e plot the accuracy of **Bank** model on the training and cv score against the kernel, C, and gamma (rbf and poly only). Findings are similar to Telcom, but a few differences are noticed. Firstly, Bank SVM linear model has train and cv curves completely overlap regardless of C (Fig 4a). Secondly, overfitting issue in Bank rbf and poly methods seem to appear at a C or gamma larger than Telcom counterpart. In other words, Bank model tends to generalize better than corresponding Telcom model likely due to its less features (13 vs. 40). A new problem is noticed for Bank linear SVM model. At C of 10^{-5}, linear kernel method has the highest cv score but all examples are predicted as negative to boost up overall accuracy because this dataset has more negative cases (data not present). In reality, this model would be useless. Finally, poly kernel with degree =3, C=1, gamma =1 is selected for Bank SVM model due to its highest validation score (0.825).

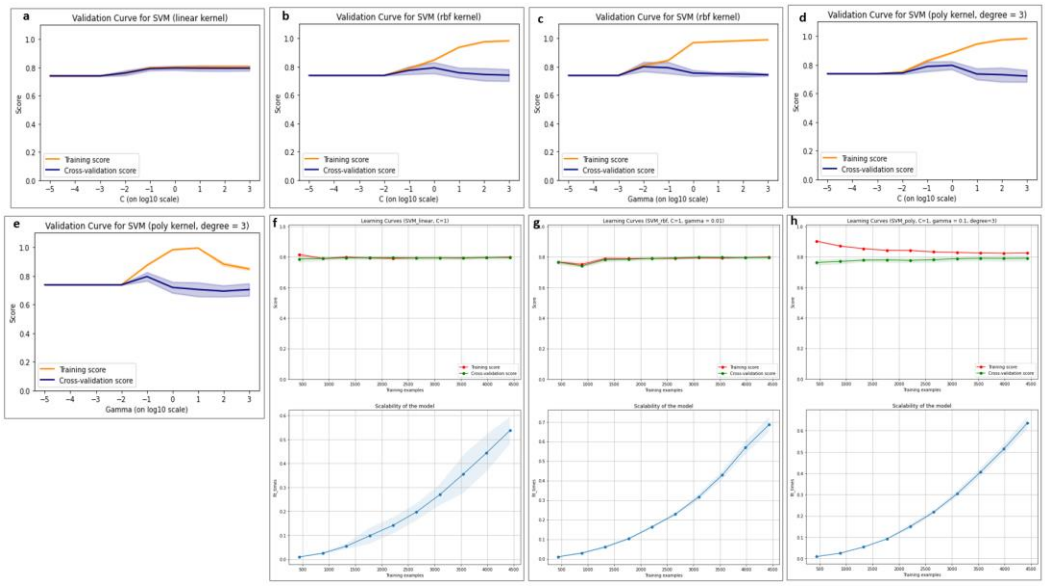

Figure 3. Telcom SVM model validation and learning curves

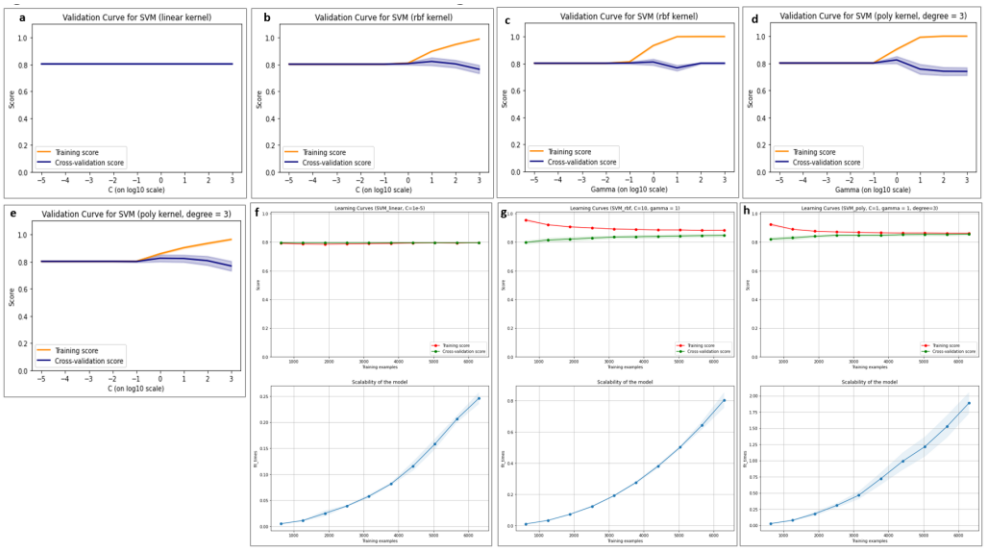

Figure 4. Bank SVM model validation and learning curves

3.3. Decision tree (DT) and AdaBoost decision tree (ABDT)

Fig 5a presents pruning effect on train and cv accuracy for **Telcom dataset**. Gini impurity is used to split attribute, which describes the probability of misclassifying an example. A post-pruning method using cost complexity pruning parameterized by ccp_alpha (alpha) was applied to control tree size. Alpha is a threshold for overall impurity improvement. Greater alpha leads to smaller tree size and deeper pruning [3]. In Fig 5a, when alpha is ~ 0, DT model has so many nodes that overfits training set and fails to generalize. Increase in alpha gets more nodes trimmed, which bridges the discrepancy between train and cv scores. Therefore, tree pruning significantly addresses overfitting of this DT model. The best performance is found with alpha of 0.000826. Learning curve for Telcom DT model is in Fig 5b. With small data size, train accuracy is much greater than corresponding cv score (1 vs. ~0.7), but including more data significantly facilitates model generalization. Meanwhile, training time also slightly increases as data size although fitting process is very fast (~0.025s) and comparable to KNN.

Adaptive boosting was selected in this assignment to create boosted DT. It is an ensemble algorithm. At the beginning it assigns equal weights to all data, grows a DT during each iteration. Based on weighted error rate, the specific DT's influence in the ensemble is calculated. DT with less error rate obtains higher power in final voting which is decayed by learning rate parameter (LR). Weight of wrongly predicted example is also updated. Iteration stops until predetermined trees numbers is reached [4]. Fig. 5 A-C explore individual effect of LR, alpha, and n_estimators on train and cv scores for Telcom ABDT model. Surprisingly, boosting method seems to generate worse model performance than regular decision tree. Regardless of change in hyperparameters, big difference always appears between train and cv score, indicating overfitting. Cutting decision tree deeper by using larger alpha also fails to effectively reduce the difference in train and cv scores. Train and cv validation curve only converge at n_estimator =1 (Fig. 5C), using more trees exacerbate overfitting. In other words, DT likely better fits this dataset than ABDT. To facilitate hyperparameter tuning, grid search was applied and the best performance for Telcom ADBT is alpha= 0.002, lr = 0.1, and n_estimators = 25. The corresponding learning curve is in Fig 5 D. As increase in training size, cv and training scores slowly converge to ~0.8. Its training time also grows linearly as training size, which is close to SVM model and ~ 20 times longer than corresponding DT model. Unfortunately, the extra

complexity from boosting tends to slow down training process rather than improve model accuracy.

Fig. 6a demonstrates how pruning by alpha influences train and cv accuracy for **Bank dataset**. Similar to Telcom data, this DT model overfits when DT has a lot of nodes (alpha ~ 0) and this issue is effectively mitigated by pruning. The best alpha is 0.002 and its learning curve is in Fig. 6b. Train and cv learning curves converge relatively fast as sample size grows to ~2.5k and further addition in training size tends to only increase training time. Validation curves of Bank ABDT models are in Fig. 6 A-C. In this case, only pruning (Fig. 6B) shows noticeable effects on model accuracy. As alpha increases from 0 to ~0.002, differences in train and cv accuracy gradually get smaller and are then completely eliminated after alpha reaches 0.002. In both Bank and Telcom case, changing LR and n estimator fail to influence model. LR controls influence of individual tree in final prediction, n_estimator is the number of trees allowed in training. In Telcom case, large gap is in train and cv curve regardless of parameter value; while in Bank case, two scores are almost identical. Ada-boosting positively affects Bank model prediction. This difference likely results from inherent differences in features (such as size, types, etc.). Based on grid search, alpha of 0.002, lr of 0.5, and n_estimators of 15 have highest cv score and are selected to plot learning curve and train the final ADBT. According to Fig. 6D, adding training examples can effectively improve model generalization although training time linearly grows.

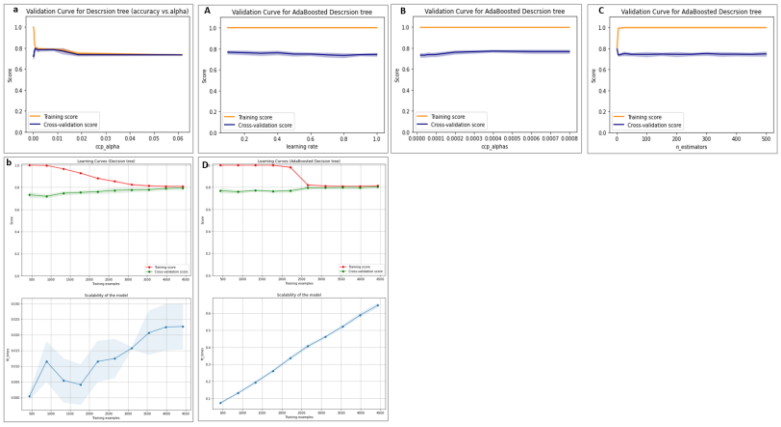

Figure 5. Telcom decision tree and AdaBoost decision tree model validation and learning curves

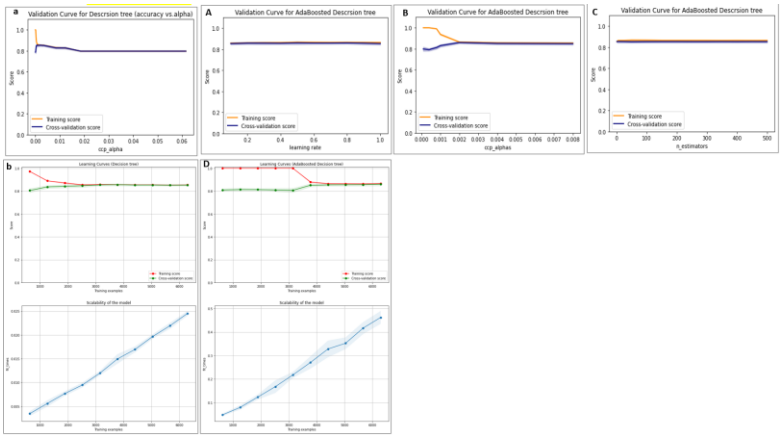

Figure 6. Bank decision tree and AdaBoost decision tree model validation and learning curves

3.4. Neural Network (NN)

For **Telcom dataset**, initial exploration on hidden layer was performed for neuron size in the range of [25, 50, 128] with one layer, ReLu activation, and LR of 0.01. Results are in Fig.7 a-c where changes of loss (black), accuracy (blue), precision (orange), and recall (green) are plotted against epoch ranging from [0 to 200] (Left: training; Right: cv). At neuron size of 25, model performance barely changes as epochs. As neuron size increases to 50 or 128, epoch has more significant influence on both training and validation results. Training accuracy, precision, and recall gradually rise over epochs with reduced loss. Larger neuron size, better training results. But validation accuracy, precision and recall show large fluctuation around the initial values over 200 epochs, suggesting model is very sensitive to noise and likely overfits. Another evident of overfitting is dramatic growth of validation loss over iterations (eg. ~ 2 times increase for 50 neurons, ~3 times for 128 neurons). Model with two hidden layers were also tested. Fig 7d uses neuron size of 25/10, Fig 7e uses neuron size of 25/50. Both models' hidden layers use ReLu and 0.01 LR. Similar overfitting symptoms are noticed in both learning curves. For example, training loss decreases over epochs while validation loss largely increases. Compared to Fig 7a, no meaningful improvement in model performance is noticed after adding a second layer. Then, LR is set to 0.001 (single layer/25 neuron, ReLu) and its effect is visualized in Fig 7f. Compared to Fig 7a (LR = 0.01), reducing LR to 0.001 helps model learn especially during the first 25 epochs. Also, train and cv results in Fig 7f are more comparable than those in Fig 7a. This suggests better model generalization when learning rate is reduced from 0.01 to 0.001, which agrees with other's experience [5].

Activation function of the hidden layer is the next factor analyzed. Fig 8a used single hidden layer with 25 neurons, sigmoid, and LR of 0.01. Compared to Fig 7a (ReLu), sigmoid results better generalize with slightly less validation loss (sigmoid:0.46; ReLu: 0.55). Given previous findings, changing LR seems to improve model performance more effectively than using larger neural network. Thus, effects of LR on hidden layer with sigmoid activation are investigated. When LR reduces to 0.001 (Fig 8b) and 0.0005 (Fig 8c), several advantages are noticed: a) both training and validation loss curves drop over iterations; b) training and validation precision scores are much more consistent; c) validation recall score is higher than corresponding training one at the same iteration step; d) less fluctuation in validation precision and recall values among epochs. When LR is further reduced to 0.0001, recall and precision scores drop to ~0 in the first 50 iterations. Likely this LR is so small that weight update becomes negligible and is overwhelmed by random noise. After comparing all NN results for Telcom, the one with best performance is single hidden layer with 25 neurons, sigmoid and LR of 0.0005.

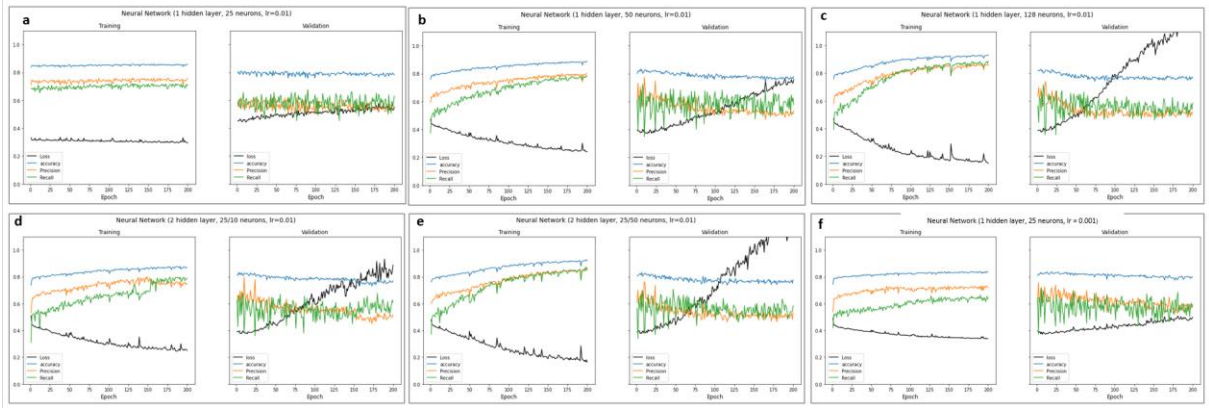

Figure 7. Learning curves for Telcom neural network model using ReLu for hidden layer(s) with different neuron size (a-c), hidden layer (d-e vs. a), learning rate (f vs.a)

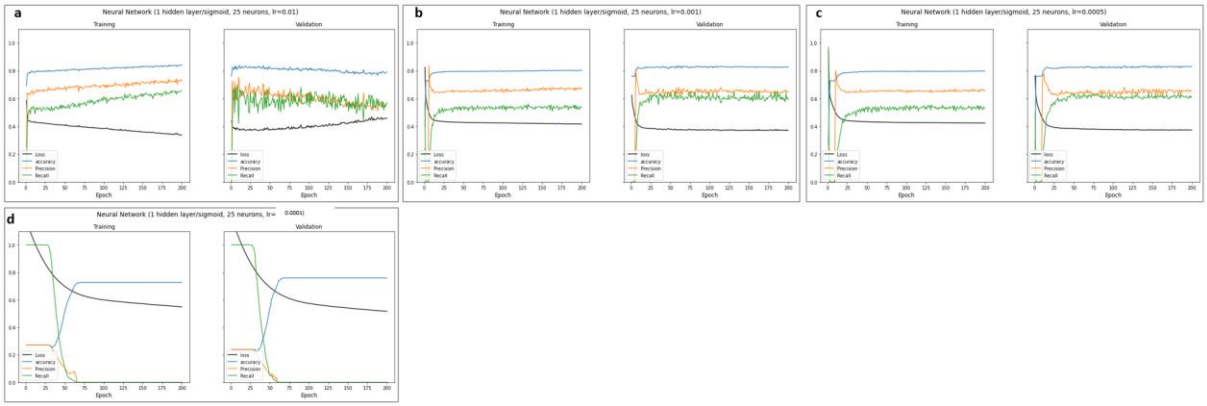

Figure 8. Learning curves for Telcom neural network model using Sigmoid and 25 neurons for hidden layer with different learning rates.

The same strategy for hyperparameter exploration is applied to **Bank dataset**. Plots in Fig 9 are results from ReLu. Fig 9a-c show the effects of neuron size in single hidden-layer on learning curves. In this case, smaller neuron size (10, 20, 50) is analyzed because this dataset only has 13 features. Interestingly, increasing neuron size does not cause overfitting as I observed from the Telcom results, possibly because smaller neuron sizes are selected in this exploration (Bank: 10 to 50 vs. Telcom 25 to 128). Fig 9 d-e present how two hidden layers (20/10 neurons, 20/50 neurons) influence model performance. Similar as findings from Telcom, adding another hidden layer tends to overfit model rather than improve performance. So far model with single hidden layer/20 neurons has the best performance, so its LR is reduced to 0.001 (Fig 9f). Compared to Fig 9b (LR=0.01), validation precision and recall curves in Fig 9f are more stable after the first 25 iterations. Also, validation precision is higher after reducing LR. Results for NN model with sigmoid, 20 neurons and different LRs are in Fig 10 a-c. Unlike Telcom case, changing hidden layer's activation function from ReLu to sigmoid fails to noticeably improve model performance (Fig 10 a vs. Fig 9b). Further reducing LR from 0.01 (Fig 10a) to 0.001 (Fig 10b) or 0.0005 (Fig 10c) causes large drop in precision and recall with increased loss over iterations. Overall, single hidden layer with 20 neuron, ReLu, LR = 0.005 show the best performance among all the explored architectures.

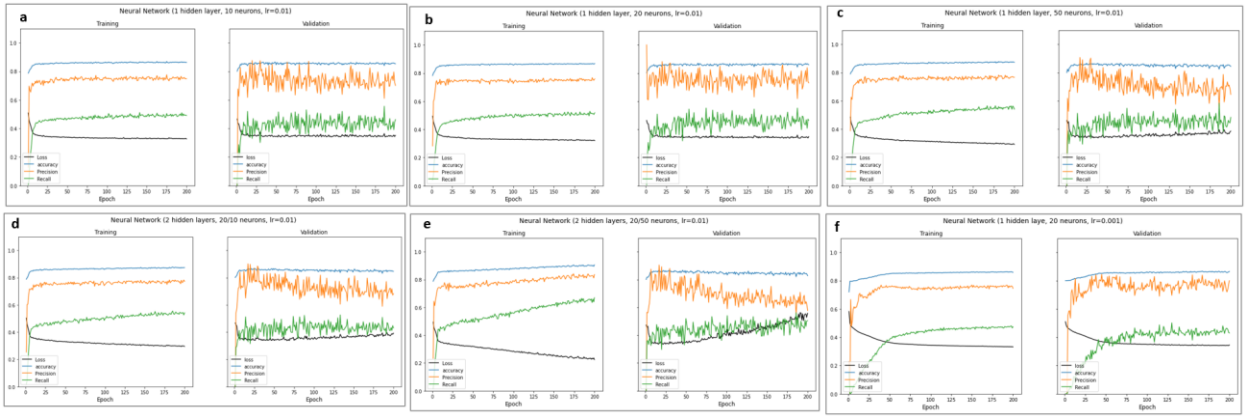

Figure 9. Learning curves for Bank neural network model using ReLu for hidden layer(s) with different neuron size (a-c), hidden layer (d-e vs. a), learning rate (f vs.a)

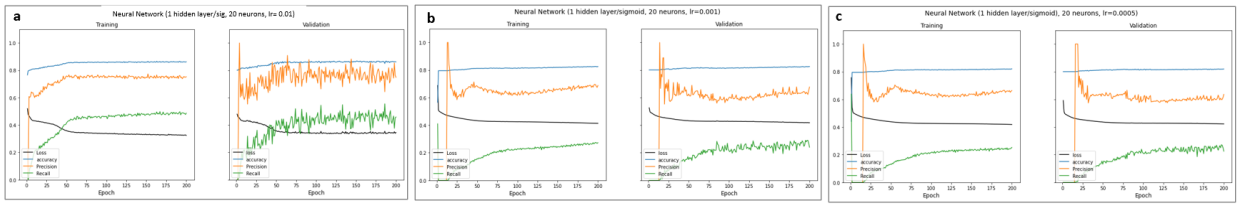

Figure 10. Learning curves for Bank neural network model using Sigmoid and 20 neurons for hidden layer with different learning rates.

3.5. Final model performance comparison among 5 algorithms

The following findings are consistent between Telcom and Bank datasets. Firstly, testing performance (accuracy, precision, and recall) is similar to corresponding training score, suggesting generally good model generalization. Secondly, within each dataset, accuracy scores are comparable among 5 algorithms. Difference between highest and lowest test accuracy is <1% for Telcom and ~ 5% for Bank. In other words, using more complex algorithm does not significantly improve model accuracy. But both datasets have more negative cases than positive ones, accuracy alone could not fully represent model performance. Precision and recall are also applied. Precision quantifies the proportion of correct positive identifications; recall indicates the proportion of actual positives that is identified correctly. In this work, it is assumed that false negative is worse than false positive thus companies can take action in advance to retain customers. Therefore, model with relatively larger recall is more desired if accuracy and precision are comparable. Noticeable differences in precision and recall scores are observed among 5 algorithms.

Table 1. Final model performance comparison between 5 algorithms and 2 datasets

Datasets	Algorithms	Train Dataset (70%)			Test Dataset (30%)			Fitting Time
		Accuracy	Precision	Recall	Accuracy	Precision	Recall	
Telecom Churns	KNN	79.5%	62.8%	55.8%	79.1%	61.7%	56.7%	~ 0.045s
	SVM	79.9%	64.7%	53.5%	79.5%	63.8%	53.3%	~0.55s
	Decision Tree	80.4%	68.3%	48.7%	79.7%	66.1%	48.7%	~0.025s
	Boosted Decision Tree	81.2%	74.4%	55.3%	80.5%	67.1%	52.0%	~0.65s
	Neural Network	80.5%	66.1%	54.3%	80.9%	66.8%	56.3%	~50s
Bank Customer Churns	KNN	82.3%	75.8%	24.1%	81.5%	65.1%	20.4%	~0.025s
	SVM	86.1%	81.5%	41.2%	86.1%	79.4%	42.8%	~1.9s
	Decision Tree	85.4%	69.6%	50.4%	85.3%	68.5%	52.1%	~0.025s
	Boosted Decision Tree	86.8%	78.4%	48.8%	86.5%	75.7%	49.8%	~0.45s
	Neural Network	86.2%	74.0%	49.8%	85.7%	70.6%	51.3%	~60s

For Telcom, ADBT had highest training precision (74.4%), followed by decision tree (68.3%). Thus, using ada-boosting was able to improve train precision by ~6%, because more weights are assigned to misclassified examples. But such advantage does not translate well to

test results since test precision of ADBT drops to 67%. ADBT also has the second highest train recall (55.3%) that is ~6% higher than that of decision tree, so boosting algorithm in this case tends to correct both false positive and false negative predictions. KNN shows lowest train and test precision (62.8%, 61.7%), but its recall scores are highest (train 55.8%, test 56.7%). This agrees with the precision/recall tradeoff . Overall, neural network is considered to be the best model for Telcom Customer Churn prediction because it has highest test accuracy (80.9%), second highest test precision (66.8%) and recall (56.3%).

For Bank data, SVM has highest precision (train 81.5%, test 79.4%), followed by ADBT (train 78.4%, test 75.7%). Compared to DT, ada-boosting significantly improves precision by ~7-9%. However, DT has highest recall (train 50.4%, test 52.1%). Recall of KNN (20.4%) is ~ 50% less than others (42.8%-52.1%), suggesting that some positive cases in bank dataset are close to negative ones distance-wise in the feature dimension. KNN is not the proper algorithm for this dataset and removed from consideration. Despite SVM has highest test precision (79.4%)) and second highest test accuracy (86.1%), its recalls are 7-10% less than other 3 algorithms. As stated earlier, recall plays more important role in model selection, so SVM is also eliminated. ADBT has highest test accuracy (86.5%) among 5 models with test precision noticeably higher than the remaining 2 candidates (75.5% vs. ~70% or less). Its recall (49.8%) is only ~1-2% less than DT (52.1%) and NN (51.3%). Thus, AdaBoost decision tree model is considered to be the best option for Bank Customer Churn prediction.

For both datasets, fitting time for 5 algorithms rank from high to low in the order of NN >>> SVM ~ ADBT >>> DT ~ KNN. Comparing fitting time of the same algorithms between two datasets, Bank is faster in terms of KNN and AdaBoost decision tree and slower for the rest. Given that Bank has ~ twice of data points and ~ one-third features Telcom, this findings suggests both feature and data point size affect fitting time with effects vary among algorithms. Interestingly, training time complexity of DT model is O(d * n * log(n)) which is more complex than that of KNN (O(1)), but their fitting time is comparable. This is because time complexity is a theoretical concept assuming infinite input size [6]. With specific data size and its unique feature characteristics, algorithm with less time complexity could run faster in practice.

3.6. Potential directions for model performance improvement

Cross-validation is very helpful and useful in hyperparameter optimization and overfitting diagnosis. It has been applied throughout this assignment. There is significant room to further improve model performance. Some potential directions are listed as follows.

Dimension reduction could be the first approach to improve model performance. One option is feature selection. Some algorithms (such as decision tree, SVM) could import feature importance by using training data. Important features identified by different methods can be compared. Domain knowledge and literature research are also critical for sound feature selection. PCA would be another classic option to reduce dimension. Both datasets have significantly more negative cases than positive ones. This imbalance could be addressed by putting more weights on correct positive prediction or using oversampling approach such as SMOTE.

In KNN work, “auto” was applied to compute the nearest neighbors, other algorithms (such as BallTree, KDTree) together with leaf_size adjustment might generate better results. Performance of SVM could be improved by using other kernel methods or fine-tuning C/gamma. As for decision tree, only pruning was investigated in this work. Adjusting hyperparameters such as max_features and max_depth might create better model. Gradient boosting and eXtreme Gradient Boosting algorithms are also worth exploring. NN is a very complex algorithms and

very limited exploration has been conducted in this assignment. Both data sets are high dimensional, likely causing gradient descent gets stuck to a saddle point. Adding some momentum in optimizer could help gradient descent escape from saddle points. Adam optimizer only varies learning rate from 0 (no update) and initial setup (max update), so its degree of adaptation is limited[7]. Using exponential decay might further reduce loss.

At least the abovementioned areas needs to be explored before I could conclude whether obtained performance is from inherent characteristics of this dataset or not.

Appendix

Fig.1-10 with better resolution and fitting time for final NN models are in this link: https://drive.google.com/file/d/1FsZ-X2YpPD3ZRt0ZAGXbGEzLGlJv95pW/view?usp=sharing

Reference

1. T. Vafeiadis, K.I. Diamantaras, G. Sarigiannidis, K.Ch. Chatzisavvas. (2015) A comparison of machine learning techniques for customer churn prediction. Simulation Modelling Practice and Theory, 55, 1-9.
2. C. C. Aggarwal, A. Hinneburg, D. A. Keim. (2001). On the surprising behavior of distance metrics in high dimensional space. Database Theory — ICDT 2001. ICDT 2001. Lecture Notes in Computer Science, vol 1973. Springer, Berlin, Heidelberg.
3. A. Das. (2020). Decision tree classifier and cost computation pruning using Python. https://towardsdatascience.com/decision-tree-classifier-and-cost-computation-pruning-using-python-b93a0985ea77.
4. L. Chen. (2019). Basic ensemble learning (random forest, AdaBoost, gradient boosting)- Step by Step Explained. https://towardsdatascience.com/basic-ensemble-learning-random-forest-adaboost-gradient-boosting-step-by-step-explained-95d49d1e2725.
5. https://stats.stackexchange.com/questions/313278/no-change-in-accuracy-using-adam-optimizer-when-sgd-works-fine.
6. https://stackoverflow.com/questions/4915842/difference-between-time-complexity-and-running-time#:~:text=Running%20time%20is%20how%20long,classes%20and%20big%2DO%20notation.
7. https://stackoverflow.com/questions/39517431/should-we-do-learning-rate-decay-for-adam-optimizer