Building classifiers on UCI Credit Card and Gender Recognition datasets

Zhihua Jin

zjin80@gatech.edu

1. Datasets

Both datasets are found in Kaggle. The first one is "UCI Credit Card" dataset (Kaggle.com, 2019). This is a dataset used to study card users' default payments in Taiwan. For default payments in this situation, it basically means users are unable to pay the interests of credit card or other relevant debt to the bank. I would like to determine whether certain user will have default payment or not. The second dataset is "Gender Recognition by Voice"(Kaggle.com, 2019), in which consists 3000 pre-processed voice samples and 21 variables. It is used to conduct binary classification as well because the voice is either from a man or woman.

The dataset 1 has 30000 samples, each of which is categorized into 24 features and 1 target variable. It could be seen that clients with default payments are 23364, while those without are 6636 (nearly 4:1). So it could be seen that most clients are credible, which makes the dataset somehow imbalanced. For dataset 2, it comprises 3168 samples evenly distributed to man voice or woman voice (Fig 1).

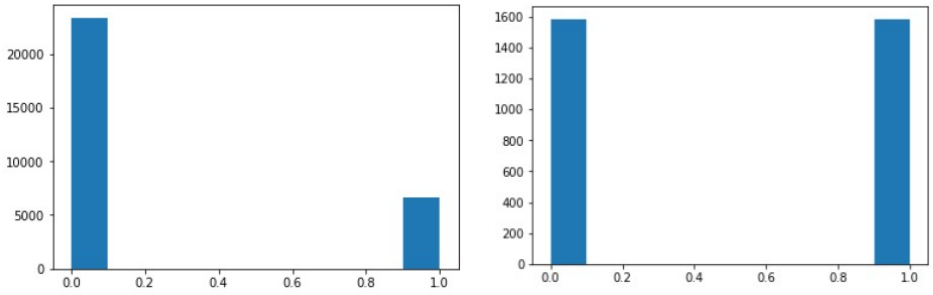

Fig. 1 Variable distribution of Dataset 1 (left); Variable distribution of Dataset 2 (right)

2. Why Interesting?

The reasons why I choose these two datasets include: 1) They may be of practical use to certain organizations and help identify users' gender or credit standing. 2) Although they are both binary problems, the voice dataset is a small dataset (3000 samples) with 50/50 classification problem while the credit dataset is an imbalanced 80/20 distribution dataset with samples 10 times more (30000 samples). I suppose pre-processing and different models will lead to different performance results for these two sets.

3. Feature Engineering

Feature engineering is more than a science but art. For dataset 1, I first checked the correlations (Fig 2) among all variables and designing new features manually so as to do better supervised learning.

As we can see in Fig 2, Light color means that the variables are highly correlated, and dark color means vice versa. 'PAY_0' to 'PAY_6' are highly correlated, so as 'BILL_AMT1' to 'BILL_AMT6'. Some variables, such as EDUCATION, MARRIAGE and PAY_0 etc. are nominal, i.e., their values are discrete, representing different meanings. However, some variables are continuous, such as AGE, we cannot use the same standard methods for them. Instead, we turn to quantize the continuous variables. A way to examine whether our new feature will work is to check whether the correlation with the target variable is larger (in absolute value) than before. So I quantize AGE variable to be in several bins: [20, 30), [30, 40), … and so on.

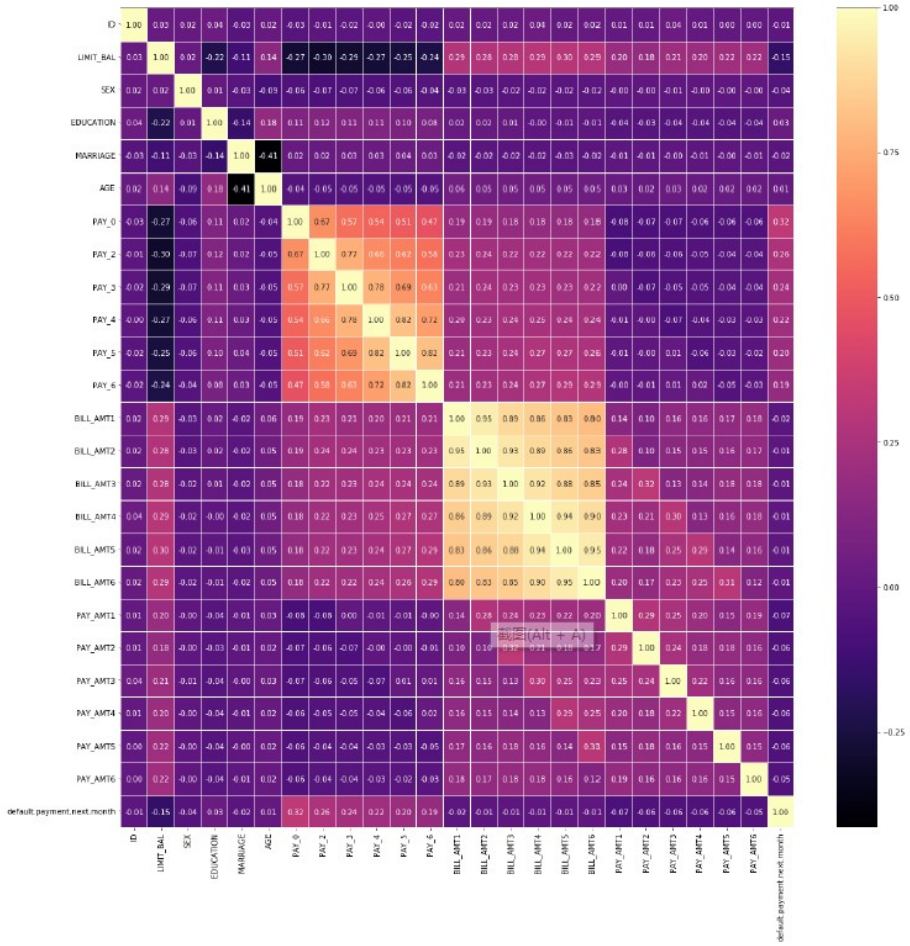

Fig. 2 Correlation between each pair of variables of dataset 1

The corrcoef raises up from 0.01389 to 0.014722. After basic data pre-processing, I designed a new variable SE_MA, combining the information from SEX and MARRIAGE, that is, married man, single man, married woman, single woman and others. The corrcoef of SEX is 0.039961, that of MARRIAGE is 0.027575, and corrcoef of SE_MA is 0.046897, showing a great improvement. Moreover, I design six new variables named 'ratio', which is obtained from dividing BILL_AMT by LIMIT_BAL. These variables shows the ratio of bill in the balance of the client. These new variables all improve the corrcoef. So, finally, I expand the number of features from 24 to 32. For dataset 2, it is a clean one with non-null values and I got the first glance of it using similar analysis mentioned above.

4. Model Building and Analysis

In this part, I will introduce 5 machine learning models and finetune the hyper-parameters of them. They are all implemented using scikit-learn (Scikit-learn.org, 2019). Not mentioned hyperparameters use the default setting. To compare different parameters and models, I use 5-fold cross validation to see the average performance. That is to say, I run the model 5 times on the datasets. At each time, the dataset is split into 5 folds randomly, 4 folds for training and 1 for testing. This can help see models' out-of-sample (test) data performance and finetune parameters. To evaluate these different learners quantitatively, several indicators are used including confusion matrix, accuracy score, f1 score and AUC score, etc. Because dataset 1 is imbalanced, when splitting two datasets , I use stratified sampling method to ensure the minor group can also be sampled representatively (Explorable.com, 2019). For confusion matrix, it can tell if classification models are confused when making prediction. There are four values including: true positive (TP) for correctly predicted true events, false negative (FN) for incorrectly predicted false values, and corresponding false positive (FP) and true negative (TN). Better models would have more TPs and TNs with fewer FP and FN. So confused matrix will look like this (Raschka, 2019):

		Predicted class	
		P	N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

Fig. 3 Confusion Matrix

Based on this, accuracy and precision can be calculated to evaluate the models. These further leads to F1 score, which is a metric that ranges from 0 to 1 that combines accuracy and precision. Higher F1 score means better model (Brownlee, 2019). In addition, receiver operating characteristic curve (ROC) is used since it is not dependent on class distributions like other metrics mentioned above. Instead, it uses false positive rate vs true positive rate. Ideal models would have larger values on y-axis and smaller values on x-axis. The area under curve (AUC) can also be calculated to compare the competencies of different models. (Brownlee, 2019). AUC value ranges from 0.5 to 1 and value closer to 1 means better classifier. To give the performance curve predicted by the learner and show how error changes as the training set size varies, learning curve would be created (Olteanu, 2019). I used accuracy and AUC to evaluate learners so better scores mean more learning. Besides, model complexity curve would be created to diagnose model performance, since we anticipate difference of accuracy smaller between training and testing sets when choosing certain hyper-paramaters.

1) Decision Tree

Decision tree recursively selects a feature to split the dataset, according to a greedy strategy (for example, maximum information gain). When the subset on a leaf is in the same class, the tree stops splitting. However, this sometimes causes overfitting. There are certainly several hyper parameters that can be pruned including max_depth, min_samples_split or min_samples_leaf, etc. Since classes and samples in both datasets are not very huge, I presume many default hyper parameters are enough, such as min_samples_leaf=1 (Scikit-learn.org, 2019). Here, I only changed max_depth to be 3, 5, 10, 20 and 30. This is called pre-pruning technique. To validate max-depth value, model complexity curve is also drawn while other metrics are calculated as well. According to the results below, the line marked bold in Tab. 1 is the best

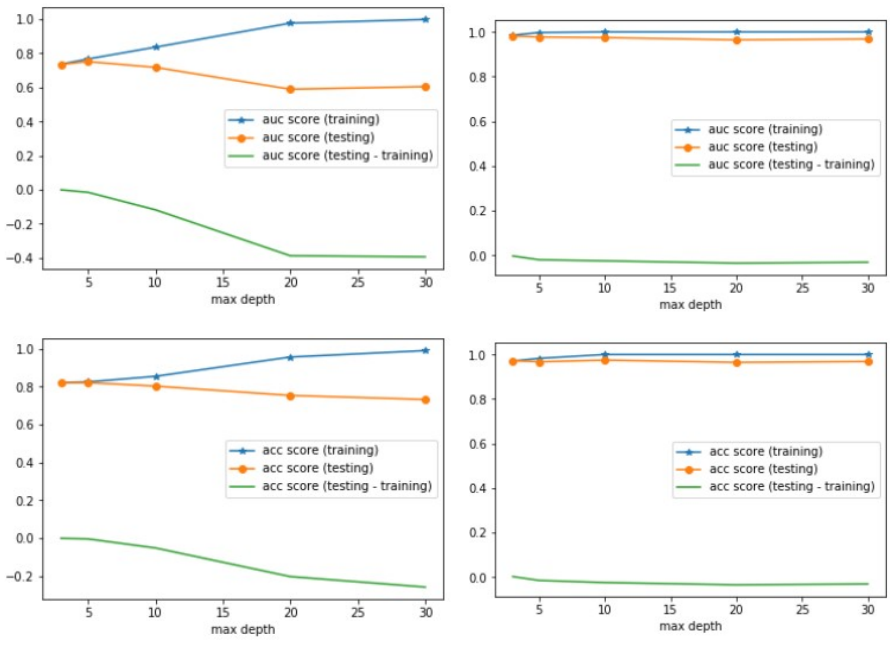

Fig. 4 Model Complexity Curve of decision tree with different maximal depth restriction: Dataset 1 (Left), Dateset 2 (Right)

one. For similar metrics, I prioritize depth with highest AUC because of the reason I mentioned above. The model complexity curve also show best performance of max_depth when acc score (testing-training) reaches near 0, as it has least variance between test and training sets. Moreover, 5-fold cross validation is done to pick the best hyper-parameter as well.

So we set the optimal depth to be 5 for dataset 1 and 3 for dataset 2.

max depth	acc	F1	auc	Confusion matrix
3	0.8209	0.4821	0.7353	[[5532 309] [1034 625]]
5	**0.8211**	**0.4807**	**0.7517**	**[[5537 304] [1038 621]]**
10	0.8027	0.4602	0.7180	[[5389 452] [1028 631]]
20	0.7533	0.4097	0.5895	[[5008 833] [1017 642]]
30	0.7321	0.4033	0.6048	[[4812 1029] [980 679]]

max depth	acc	F1	auc	Confusion matrix
3	**0.9722**	**0.9722;**	**0.9823**	**[[385 11] [11 385]]**
5	0.9672	0.9669	0.9773	[[386 10] [16 380]]
10	0.9747	0.9746	0.9747	[[388 8] [12 384]]
20	0.9646	0.9645	0.9646	[[384 12] [16 380]]
30	0.9684	0.9682	0.9684	[[386 10] [15 381]]

Tab. 1 Average of testing performance of 5-fold cross validation result: Dataset 1 (Left), Dateset 2 (Right)

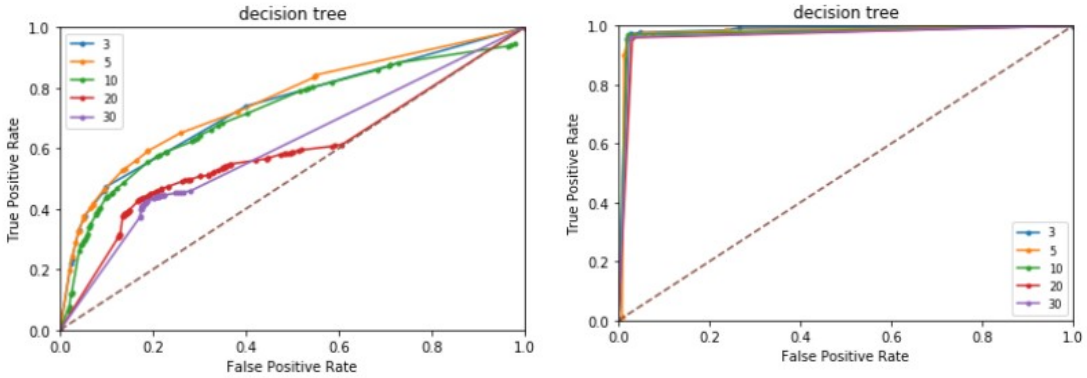

Fig. 5 ROC curve of decision tree with different maximal depth restriction: Dataset 1 (Left), Dateset 2 (Right)

In the learning curve for two datasets, we could see after the first overfitting stage, the metrics for training and testing data become closer as training data increases. However, the second dataset has much better performance than the first one. First dataset is still having issues with lower accuracy.

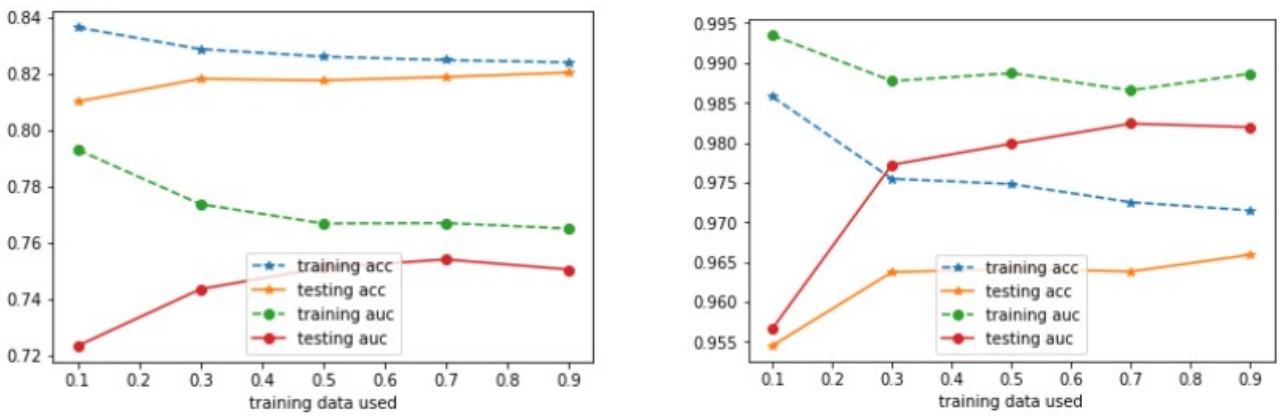

Fig. 6 Learning curve with varying training datasets: Dataset 1 (Left), Dateset 2 (Right)

For post-prune, I tried to prune a tree with max_depth=50 to show post-pruning performance better. I set 20, 50, 100, 200, etc. as threshold and remove those children nodes with minimum class less than that

threshold (Trees, Dale and Gülcan, 2019). For depths like 3 or 5, it is not essential. As we can see in Fig. 7, as we prune more nodes, the performance will go up as we obtain a good model for generalization, but for dataset 2 it will go down because the tree is too small now.

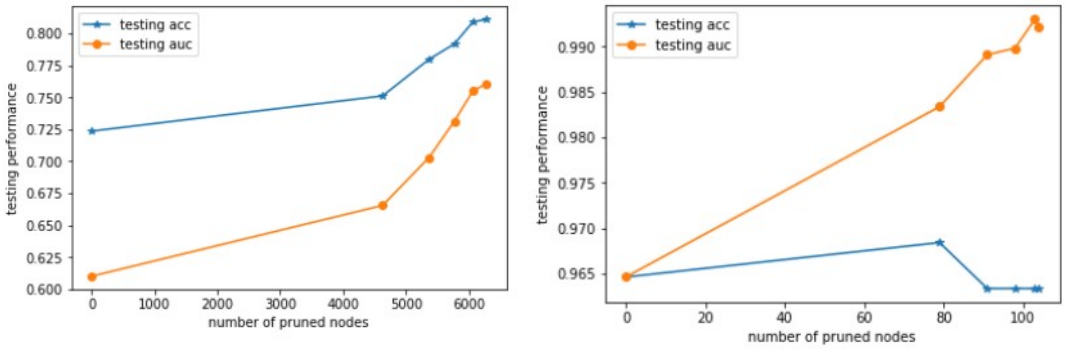

Fig. 7 Post-pruned datasets: Dataset 1 (Left), Dateset 2 (Right)

2) Neural networks

I studied neural networks that are multiple layers of non-linear transformation. Neural networks take original input, then feed forward to final layer one by one to get the prediction (En.wikipedia.org, 2019). Because our feature space is not that high-dimensional, so I just use a single hidden layer. As stated (Hyndman, 2019), occasions when 2 or 3 layers increase the performance is scarce. So I checked how the performance will change as I use different number of neurons in this layer from 10 to 100 and train iteratively (defalut max_iter=200). The range is chosen according to reference (Sciencedirect.com, 2019). Hidden neurons numbers in each hidden layer represents connection number. Iteration is not early stopped.

According to similar workflow above, the optimal n is set to 50 for dataset 1 (because auc in 5-fold CV is the highest) and 100 for dataset 2. Due to page length, similar analysis is not included here.

#neurons	acc	F1	auc	Confusion matrix
10	0.7367	0.3951	0.6530	[[4880 961] [1014 645]]
20	0.3535	0.3747	0.5886	[[1198 4643] [206 1453]]
50	**0.7280**	**0.4311**	**0.6725**	**[[4687 1154] [886 773]]**
100	0.7611	0.4117	0.6773	[[5081 760] [1032 627]]

#neurons	acc	F1	auc	Confusion matrix
10	0.8548	0.8606	0.9003	[[322 74] [41 355]]
20	0.8523	0.8536	0.9532	[[334 62] [55 341]]
50	0.9242	0.9229	0.9747	[[388 8] [12 384]]
100	**0.9684**	**0.9682**	**0.9742**	**[[373 23] [37 359]]**

Tab. 2 Average of testing performance of 5-fold cross validation result: Dataset 1 (Left), Dateset 2 (Right)

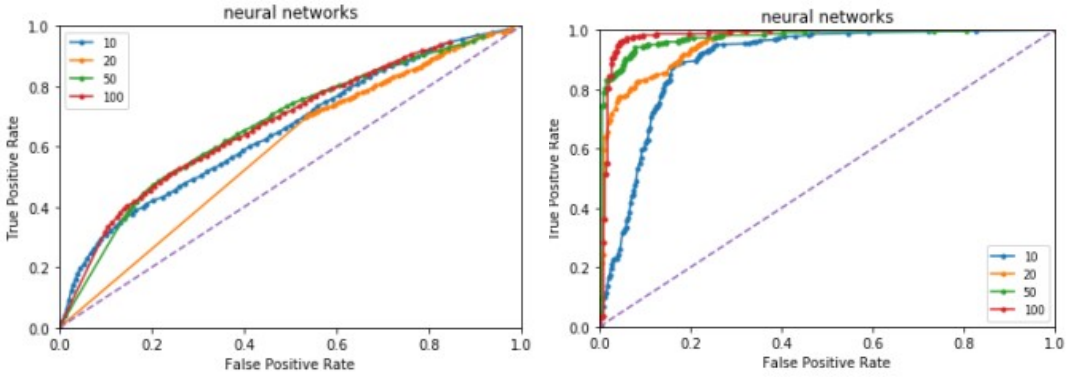

Fig. 8 ROC curve with n: Dataset 1 (Left), Dateset 2 (Right)

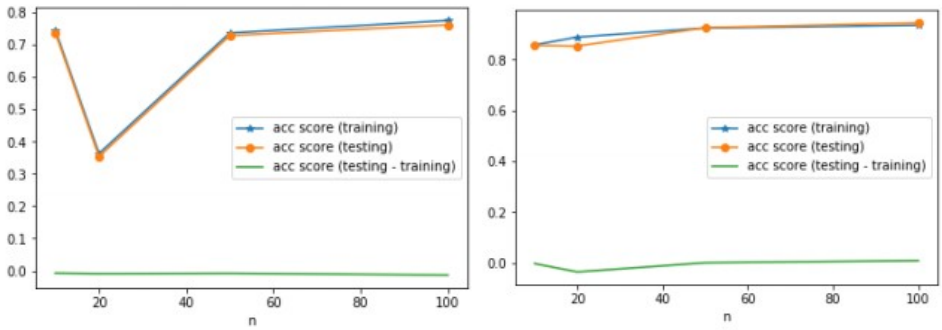

Fig. 9 Model Complexity Curve with different n: Dataset 1 (Left), Dateset 2 (Right)

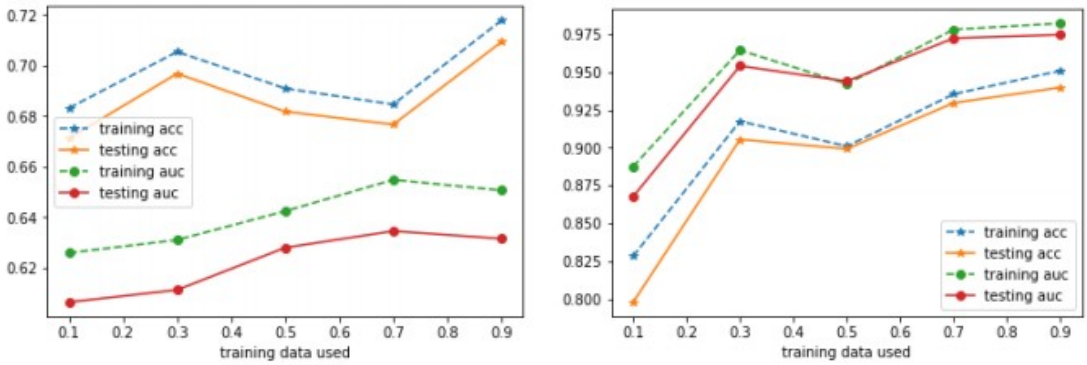

Fig. 10 Learning curve with varing training datasets: Dataset 1 (Left), Dateset 2 (Right)

We can see that dataset 2 again has better performance while dataset 1 has higher bias and underfitting.

3) Boosting

Boosting is an ensemble of base classifiers (in this case they are decision trees) to improve the final performance. Ensemble learning usually improves weaker classifiers into stronger one. Another advantage of boosting is that it suffers little from over-fitting. The hyper-parameters here is the number of base classifiers. In my work, I vary it to be 10, 20, 50, 100, and 200. According to similar workflow above, the optimal n is set to 20 for dataset 1 and 200 for dataset 2.

#base	acc	F1	auc	Confusion matrix
10	0.8179	0.4256	0.7639	[[5577 264] [1090 569]]
20	**0.8195**	**0.4567**	**0.7719**	**[[5628 213] [1153 506]]**
50	0.8189	0.4641	0.7653	[[5554 287] [1071 588]]
100	0.7611	0.4117	0.6773	[[5081 760] [1032 627]]
200	0.7700	0.4745	0.7223	[[5526 315] [1045 614]]

#base	acc	F1	auc	Confusion matrix
10	0.9684	0.9686	0.9900	[[382 14] [11 385]]
20	0.9710	0.9709	0.9912	[[385 11] [12 384]]
50	0.9722	0.9722	0.9918	[[385 11] [11 385]]
100	0.9735	0.9735	0.9954	[[386 10] [11 385]]
200	**0.9760**	**0.9760**	**0.9960**	**[[387 9] [10 386]]**

Tab. 3 Testing performance result: Dataset 1 (Left), Dateset 2 (Right)

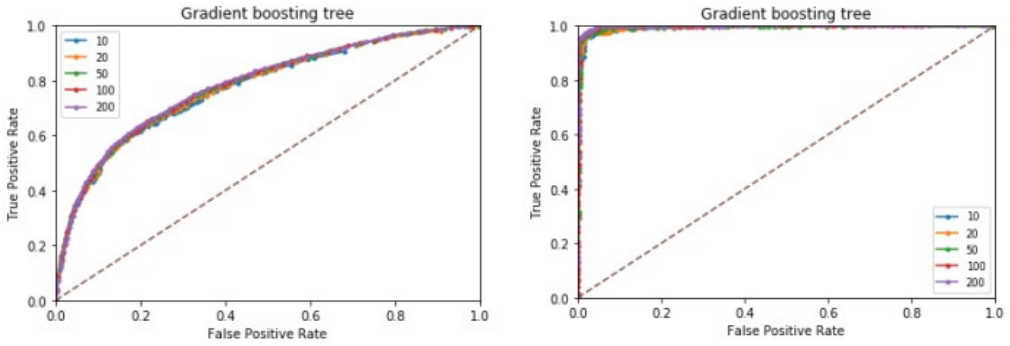

Fig. 11 ROC curve with different number of tree classifiers: Dataset 1 (Left), Dateset 2 (Right)

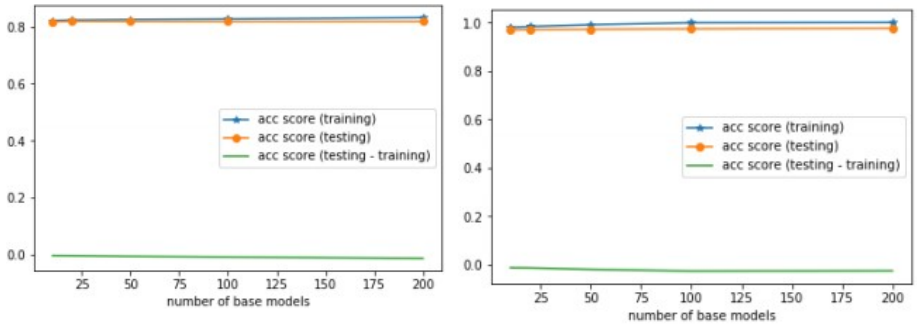

Fig. 12 Model Complexity Curve with different number of tree classifiers: Dataset 1 (Left), Dateset 2 (Right)

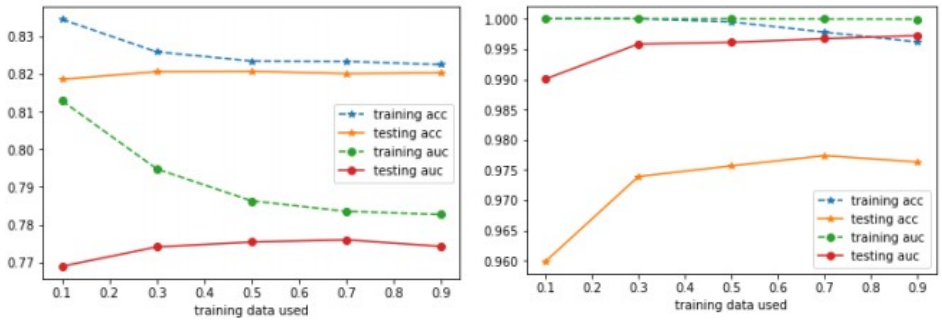

Fig. 13 Learning curve with training datasets: Dataset 1 (Left), Dateset 2 (Right)

Compared to decision tree, the ROC curve with different tree numbers is similar for both datasets. We could see that they are overall better than single tree learners. Since there is no obvious differences among different hyper-parameter, it is not a must to run 5-fold CV here.

4) Support Vector Machines (SVM)

In SVM, in order to accelerate convergence, I standardize the data in to range [0,1]. Support vector machines are max-margin classifiers, which select samples to be 'support vector' and determine the decision boundary by them. SVM with different kernel can map data into high-dimensional space, leading to a linear-separable problem (Medium, 2019) . Also, to avoid wrong or hard samples, SVM can tolerate them by adjusting the cost of misclassifying them. So, in my work, I first vary the kernel to see which one is most suitable for this problem. In linear, rbf and polynomial kernel, C is set to0.1 when max_iter equals 5000. I found that optimal kernal is rbf for dataset 1 and linear for dataset 2 (mainly depends on acc here).

kernel	acc	F1	auc	Confusion matrix
linear	0.5597	0.4259	0.7150	[[1870, 3971], [241, 1418]]
rbf	**0.8207**	**0.4464**	**0.7133**	[[5593, 248], [1096, 563]]
polynomial	0.7961	0.5126	0.7067	[[930, 4911], [146, 1513]]

kernel	acc	F1	auc	Confusion matrix
linear	**0.9760**	**0.9763**	**0.9923**	**[[382 14] [5 391]]**
rbf	0.9722	0.9724	0.9935	[[382 14] [8 388]]
polynomial	0.9583	0.9591	0.9824	[[372 24] [9 387]]

Tab. 4 Testing performance of SVM with different kernels:Dataset 1 (Left), Dateset 2 (Right)

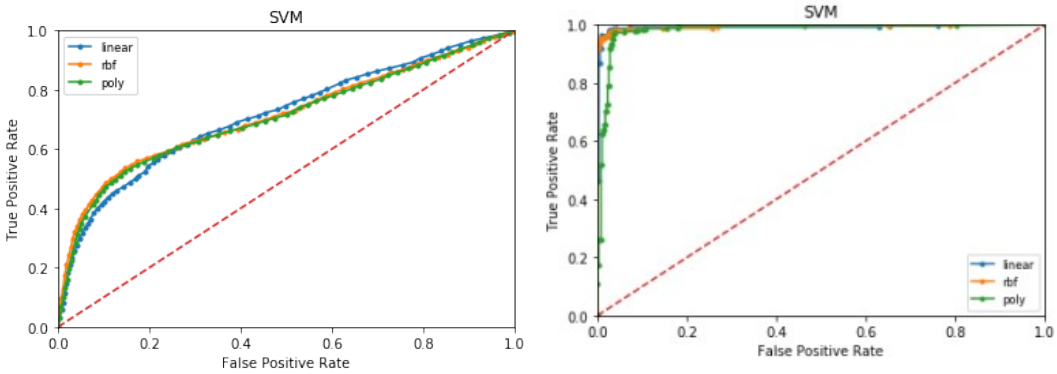

Fig. 14 ROC curve of SVM with different kernels: Dataset 1 (Left), Dateset 2 (Right)

Next, I finetune the hyper-parameters for SVM. Specifically speaking, I vary different tolerance degree (C) to see how the performance changes. In this work, I set C to be 0.01, 0.1, 0.5, 1.0, 5.0 and 10.0 for both kernels in two datasets. A larger C means that SVMs allow more data points to be wrongly classified when training, in order to avoid overfitting problem. For non-linear kernel like RBF, gamma could also be applied. With larger gamma, the model will have higher bias and low variance, so it is a trade-off question. To standarize the comparison of two datasets, I only take C into consideration. Too small C value results in underfitting while C too large will have overfitting problems and poor generalization performance. For dataset 1, since the accuracy score are nearly the same, I choose the optimal C = 0.1, which has highest F1 score. For dataset 2, C=5 has a little better acc and auc performance than C=0.1, but considering the increasing running time, I still chose C=0.1 with highest F1 score.

C	acc	F1	auc	Confusion matrix
0.01	0.8064	0.3879	0.7210	[[5588 253] [1199 460]]
0.1	**0.8144**	**0.4243**	**0.7144**	**[[5595 246] [1146 513]]**
0.5	0.8165	0.4337	0.7106	[[5597 244] [1132 527]]
1.0	0.8169	0.4394	0.7060	[[5589 252] [1121 538]]
5.0	0.8133	0.4248	0.7027	[[5594, 247], [1078, 581]]
10.0	0.8107	0.4147	0.6879	[[5577 264] [1156 503]]

C	acc	F1	auc	Confusion matrix
0.01	0.9684	0.9686	0.9943	[[383 13] [10 386]]
0.1	**0.9710**	**0.9711**	**0.9946**	**[[383 13] [10 386]]**
0.5	0.9672	0.9673	0.9947	[[381 15] [11 385]]
1.0	0.9697	0.9698	0.9946	[[383 13] [11 385]]
5.0	0.9672	0.9673	0.9947	[[381 15] [11 385]]
10.0	0.9634	0.9637	0.9948	[[378 18] [11 385]]

Tab. 5 Testing performance of SVM with different C : Dataset 1 (Left), Dateset 2 (Right)

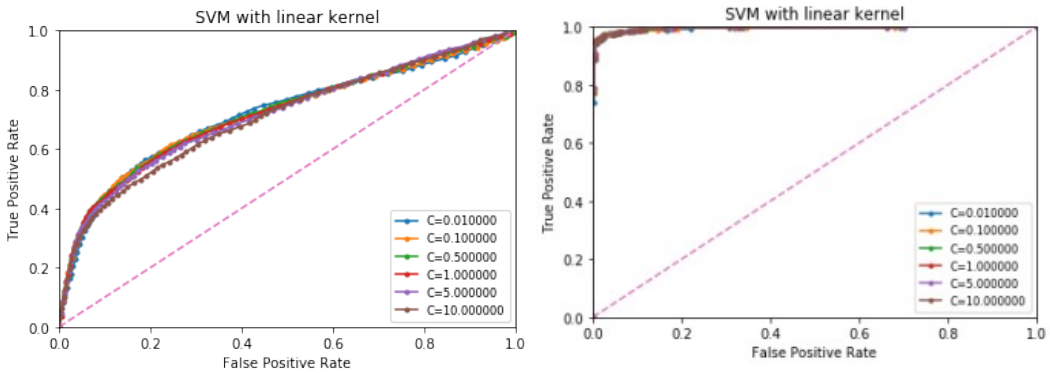

Fig. 15 ROC curve of SVM with different C: Dataset 1 (Left), Dateset 2 (Right)

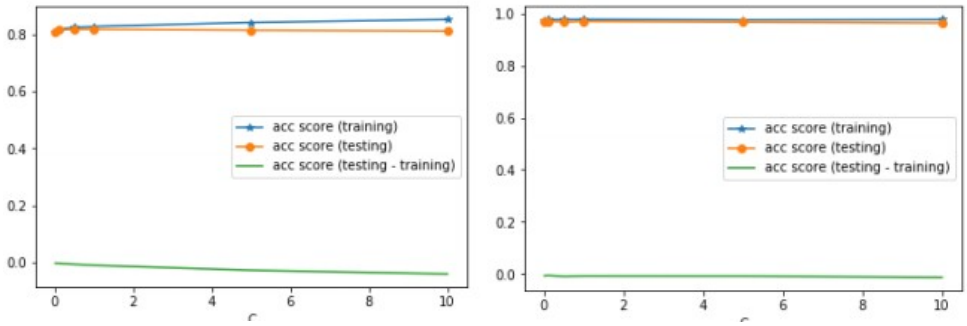

Fig. 16 Model Complexity Curve with different C: Dataset 1 (Left), Dateset 2 (Right)

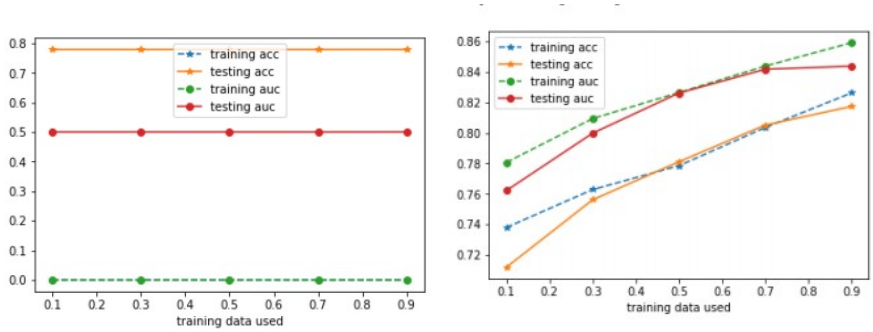

Fig. 17 Learning curve with different training dataset: Dataset 1 (Left), Dateset 2 (Right)

The learning curve for dataset using rbf kernel is quite interesting. As an imbalanced dataset, the support vectors may be limited, so it remains the same with varying training data. For dataset 1, the tendency is still similar to previous ones.

5) *k*-Nearest Neighbors (kNN)

kNN is one of the simplest but powerful machine learning techniques. It discovers k (usually predefined) nearest neighbors according to some distance metric, then use them to vote for the final prediction. The assumption under kNN is that, things are similar are likely to be the same class. For kNN, we vary the value of k, that is, the neighborhood region, to see what the performance will change. I set k to be 1,3,5,10,20 and 100. Here follows the running performance. For dataset 1, as k=100 classified almost all data into the first class (default payment=0), we select optimal k to be 20. For dataset 2, the optimal k is 10.

K	acc	F1	auc	Confusion matrix
1	0.6887	0.2881	0.5453	[[4683, 1158], [1205, 454]]
3	0.7272	0.2487	0.5755	[[5121, 720], [1319, 340]]
5	0.7467	0.2314	0.5943	[[5303, 538], [1369, 290]]
10	0.7693	0.1439	0.6200	[[5642, 199], [1501, 158]]
20	**0.7739**	**0.1174**	**0.6372**	[[5718, 123], [1537, 122]]
100	0.7783	0.0589	0.6576	[[5792, 49], [1604, 55]]

K	acc	F1	auc	Confusion matrix
1	0.7159	0.7279	0.7159	[[266 130] [95 301]]
3	0.7109	0.7204	0.7701	[[268 128] [101 295]]
5	0.6944	0.7105	0.7673	[[253 143] [99 297]]
10	**0.7045**	**0.7075**	**0.7707**	**[[275 121] [113 283]]**
20	0.6856	0.6960	0.7683	[[258 138] [111 285]]
100	0.6742	0.6884	0.7501	[[249 147] [111 285]]

Tab. 6 Average of testing performance of 5-fold cross validation result: Dataset 1 (Left), Dateset 2 (Right)

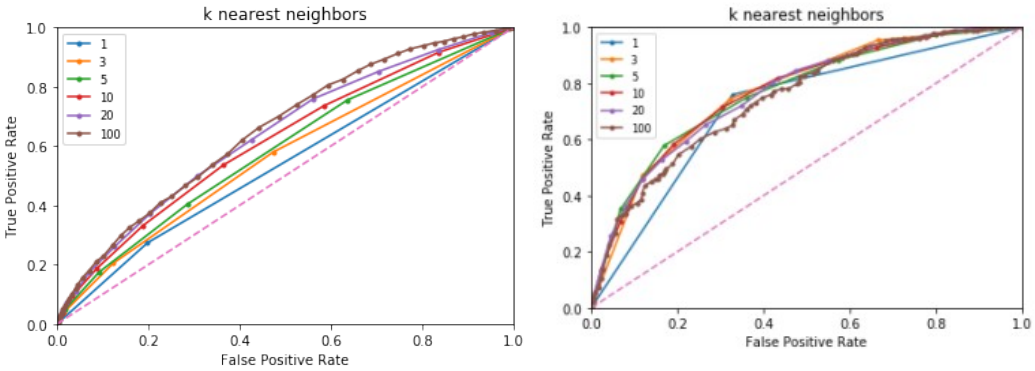

Fig. 18 ROC curve of kNN with different k: Dataset 1 (Left), Dateset 2 (Right)

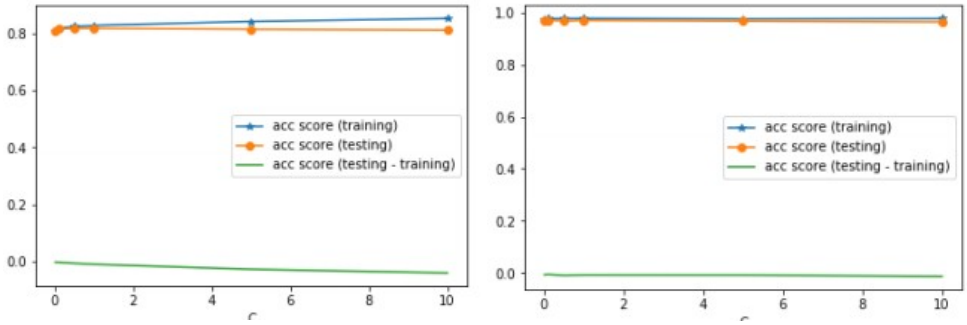

Fig. 19 Model Complexity Curve with different k: Dataset 1 (Left), Dateset 2 (Right)

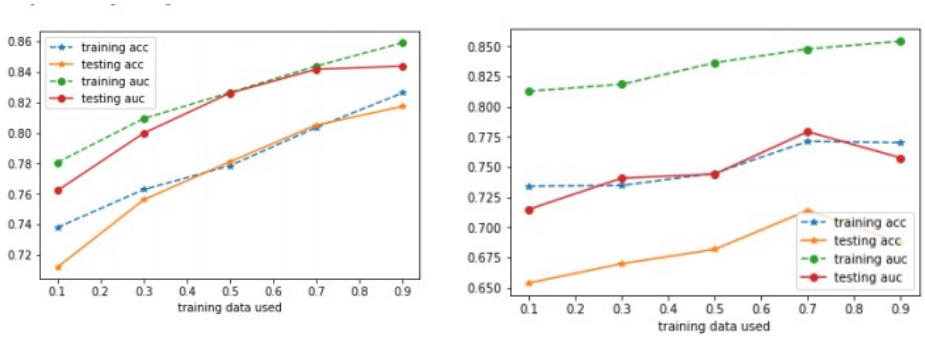

Fig. 20 Learning curve with different k: Dataset 1 (Left), Dateset 2 (Right)

As we have seen in previous learning curves as well, when training data used increase, the Accuracy and AUC lines converges. But when training data is too huge, the overfitting happens.

5. Final comparison and conclusion

Cross validation for all classifiers corresponds to the optimal hyper parameters tuned.

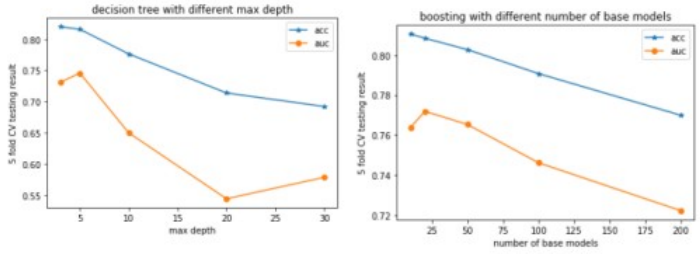

Fig. 21 Cross validation with different hyper-parameters: Dataset 1.

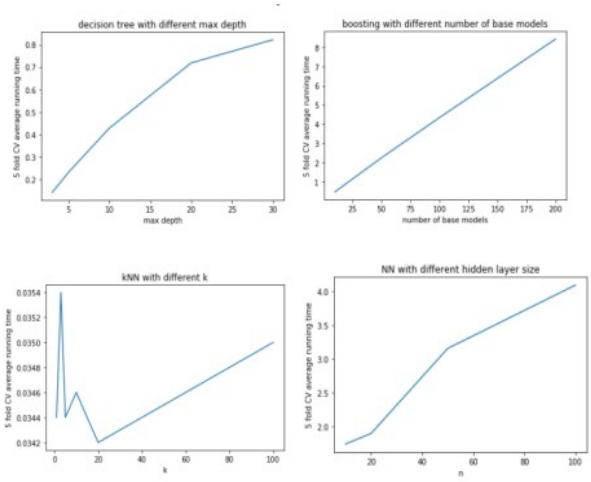

Fig. 22 Running time of hyper-parameters: Dataset 1.

For running time, if the iterations are divided, then the running time of each in dataset 1 is NN (0.01625), DT (0.25), KNN (0.0354) ,Boosting (0.052, 50 base models), SVM (0.36). The running time of each in dataset 2 is NN (0.008), DT (0.175), KNN (0.0020) ,Boosting (0.004, 20 base models), SVM (0.11). I only showed figures for dataset 1 here due to page limitation.

To this end, 5-fold CV results could be used to compare the 5 classifiers with best hyper parameters.

Models	acc	F1	auc
kNN	0.7739	0.1174	0.6372
NN	0.7280	0.4311	0.6725
DT	**0.8211**	**0.4807**	**0.7517**
Boosting	0.8195	0.4567	0.7719
SVM	0.8144	0.4243	0.7144

Models	acc	F1	auc
kNN	0.7045	0.7075	0.7707
NN	0.9684	0.9682	0.9742
DT	0.9722	0.9722;	0.9823
Boosting	**0.9760**	**0.9760**	**0.9960**
SVM	0.9710	0.9711	0.9946

Tab. 7 5-fold CV results of 5 classifiers: Dataset 1 (Left), Dateset 2 (Right)

Among them, decision tree is the best model for UCI_credit_Card dataset, and follows is boosting model. For voice_gender_recognition, boosting performs the best, following decision tree, SVM, Neural network. KNN is far worse than others in this balanced dataset.This is probably because kNN and SVM are usually used for continuous value inputs. But for this dataset I did not convert all categorical values into continuous one. Decision tree and its boosting can do better job for both categorical input and continuous valuesKNN and Richie, 2019).

We could see the classifiers used on a larger imbalanced dataset is still not satisfactory despite the fact that I have done several data-cleaning and feature engineering. On the contrary, a balanced, smaller balanced dataset with minor data pre-processing achieves much better results. To improve performances even further, I would like to upsample & downsample the two target variables and look for other methods to deal with imbalanced datasets. Moreover, due to the standarization of this report, I mainly change 1 or 2 hyper-parameters in different classifiers. However, a lot more work can be done with better computation resources. For example, XGBoost (eXtreme Gradient Boosting) could be used and learning rate tuned to do better subsampling and build more accurate models. The default learning rate is 0.1 but it usually ranges from 0.0001 to 0.1. Meanwhile, n_estimaters have to be trade-off. Slower learning would reduce contribution of each tree and add more trees, but with too many n_estimaters, it can be very slow. Another example is about neural network, more hidden layers can be add and perceptions can be increased, iteration changed, etc. Last not least, GridSearchCV is a tool to automate the tuning and training process. For more complex datasets with multiple classes or for for saving time, it can be used to search all possible parameter combinations. Then it will tell you the best hyper parameter for each models. Anyway, for my datasets, they

can be easy, but as the aim of this class is not to automate everything, this can be used in future exploration.

References

1. Brownlee, J. (2019). *How and When to Use ROC Curves and Precision-Recall Curves for Classification in Python*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python

2. En.wikipedia.org. (2019). *Artificial neural network*. [online] Available at: https://en.wikipedia.org/wiki/Artificial_neural_network

3. Explorable.com. (2019). *Stratified Sampling Method*. [online] Available at: https://explorable.com/stratified-sampling Hyndman, R. (2019). *How to choose the number of hidden layers and nodes in a feedforward neural network?*. [online] Cross Validated. Available at: https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw

4. Kaggle.com. (2019). *Default of Credit Card Clients Dataset*. [online] Available at: https://www.kaggle.com/uciml/default-of-credit-card-clients-dataset

5. Kaggle.com. (2019). *Gender Recognition by Voice*. [online] Available at: https://www.kaggle.com/primaryobjects/voicegender

6. KNN, D. and Richie, R. (2019). *Decision tree vs. KNN*. [online] Data Science Stack Exchange. Available at: https://datascience.stackexchange.com/questions/9228/decision-tree-vs-knn

7. Medium. (2019). *Support Vector Machine: Kernel Trick; Mercer's Theorem*. [online] Available at: https://towardsdatascience.com/understanding-support-vector-machine-part-2-kernel-trick-mercers-theorem-e1e6848c6c4d

8. Olteanu, A. (2019). *Tutorial: Learning Curves for Machine Learning in Python for Data Science*. [online] Dataquest. Available at: https://www.dataquest.io/blog/learning-curves-machine-learning/

9. Sciencedirect.com. (2019). *Hidden Neuron - an overview | ScienceDirect Topics*. [online] Available at: https://www.sciencedirect.com/topics/engineering/hidden-neuron

10. Scikit-learn.org. (2019). *scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation*. [online] Available at: https://scikit-learn.org/

11. Raschka, S. (2019). *Confusion Matrix - mlxtend*. [online] Rasbt.github.io. Available at: http://rasbt.github.io/mlxtend/user_guide/evaluate/confusion_matrix/ [Accessed 23 Sep. 2019].

12. Trees, P., Dale, D. and Gülcan, S. (2019). *Pruning Decision Trees*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/49428469/pruning-decision-trees [Accessed 23 Sep. 2019].