Supervised Learning Analysis

Introduction

In this project, a binary classification and a multiclass classification problem were chosen to explore the required five supervised learning techniques: decision trees, artificial neural networks (ANN), boosting, support vector machines (SVM) and k-nearest neighbors (k-NN).

The dataset of the binary classification is from a lending club. The binary classification problem is to predict whether a loan will be paid off in full or the loan will be charged off and possibly go into default. The dataset is downloaded from a Coursera course I learned before. I think this dataset is interesting because it contains many of numerical and categorical attributes but difficult to find a classifier to get a high accuracy prediction using those attributes. This dataset was an example for implementing decision tree in my previous course, but the training accuracy and test accuracy I can get is only about 0.62. For such a dataset with many inter-correlated categorical variables, decision tree might not be the best choice, because the information gain in such a decision tree is biased. I was curious to see whether one of the neural networks, boosting, support vector machines and k-nearest neighbors are able to provide a better classifier for this problem.

The second dataset about page blocks classification. This dataset contains blocks of the page layout of a document that has been detected by a segmentation process. The task is to determine the type of page block: Text (1), Horizontal line (2), Graphic (3), Vertical line (4) and Picture (5). I think this is an interesting dataset, because it has five unbalanced classes, the classification accuracy of which may be dominated by classes with large number of samples. I am curious to see which technique works best to classify all the five classes correctly.

Result Analysis and Discussion

Dataset 1, safe/risky loan classification.

Data preprocessing. The raw dataset uses the 0 and +1 as labels in column “bad_loans”. I transform this information to “safe_loans”, using +1 (safe loans) and -1 (bad loans) labels. The raw data has 66 attributes, but many of them are not closely relevant to predict a loan is safe or not. To avoid redundant attributes, I selected the most important attributes: two numeric attributes 'loan_amnt' (loan amount) and 'annual_inc' (annual income), and four categorical attributes: 'grade' (grade of the credit history), 'term' (the term of the loan), 'home_ownership' (home_ownership status: own, mortgage or rent), 'emp_length' (number of years of employment). The categorical attributes are transformed to binary attributes by the function get_dummies in Pandas. Since there are less risky loans than safe loans, I calculated the ratio of the sizes of risky loans and safe loans and use that percentage to under-sample the safe loans. Then, the balanced dataset is split at 80:20 ratio for training set and test set. The numeric attributes of both train and test set are then scaled to range [0, 1]. As a result, a balanced and scaled dataset was used for all the classifier training (3117 samples) and testing (779 samples). During the hyperparameter selection and optimization, k-fold cross-validation is performed only within the training data. The test set is never touched during hyperparameter selection and optimization, but only used for the final test.

Decision Tree.

I firstly train the sklearn DecisionTreeClassifier with the training set, without any pruning, with all default hyperparameters. I got a very deep tree with tree depth 31 and 2079 tree nodes. The training accuracy is 0.9974, but the test accuracy is only 0.5764. Obviously, it is a overfitted

decision tree. To overcome overfitting, I performed some pre-pruning, such as to limit the tree max-depth and minimum number of samples on each leaf (hyperparameter "min_samples_leaf").

For this loan classification problem, a false positive (a risky loan classified as safe loan) is more dangerous than a false negative (a safe loan classified as risky loan). Therefore, besides the accuracy, the precision is also taken into consideration. As shown in the equation (1), the less false positives we have, the higher precision value we should get.

Precision = number of true positives / (number of true positives + number of false positives) *(1)*

Considering the best trade-off between bias and variance, the max tree depth should be set at 4, as shown in the Figure 1-1A. Using max_depth = 4, the cross-validation accuracy/precision does not change much with different "min_samples_leaf" values larger than 50 (See Figure 1-1B). In both complexity curve, the precision curve has the same trend as the accuracy curve. A series of "min_samples_split" values is also checked, but has little effect as well (The plot is not shown).

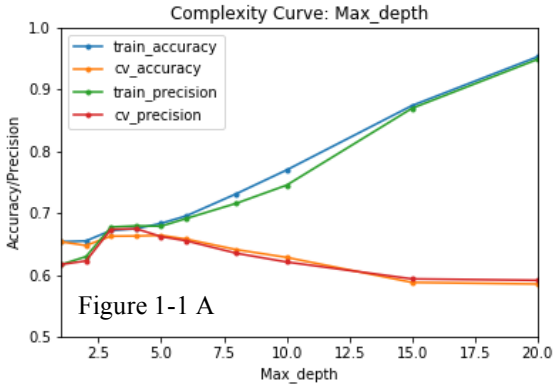


Figure 1-1 A

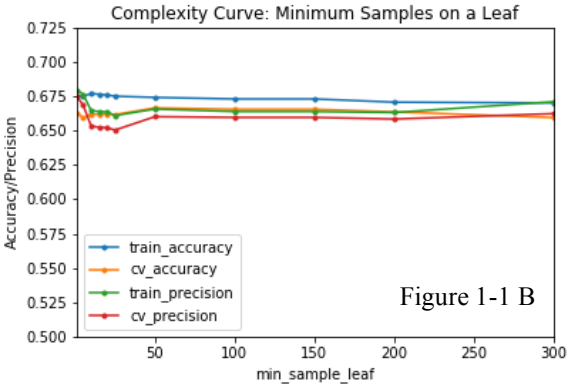
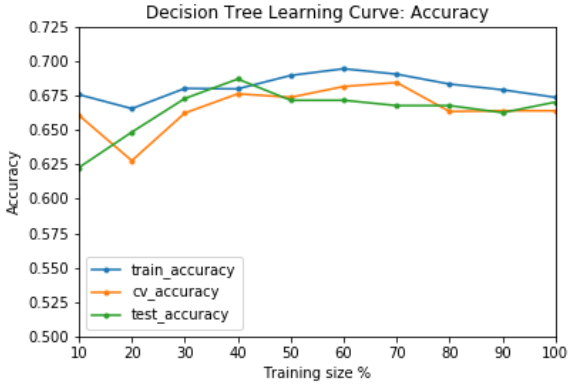


Figure 1-1 B



Learning curves of accuracy against training size are shown in Figure 1-1C. The accuracy of train, test and cross-validation converges well at all different train size larger than 30%, indicating low variance of the classification.

Neural Network.

The MLPClassifier in the sklearn package was used for the neural network classification. Using RandomizedSearchCV for the combination of hidden layer sizes ("hidden_layer_sizes"), activation functions ("activation"), solvers for weight optimization ("solver"), L2 penalty parameter ("alpha") and batch sizes ("batch_size"), I found out that solver='lbfgs', activation='identity' always generate good performance, but the "alpha" and "batch_size" do not affect the cross-validation score significantly.

The accuracy/precision at different max iterations is plotted in Figure 1-2A, but the iterations does not affect the train and cross-validation accuracy much. With the tuned parameters and with a single hidden layer, the accuracy/precision at different hidden layer size (the number of neurons in the single hidden layer) was studies. The plot of accuracy/precision vs. hidden_layer_size is

shown in Figure 1-2B. The train and cross-validation accuracy/precision does not change much as the hidden_layer_size increases.

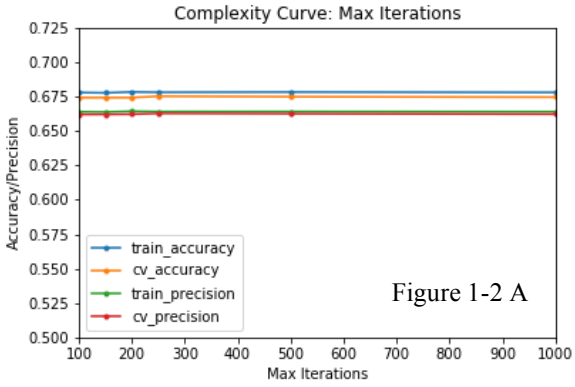


Figure 1-2 A

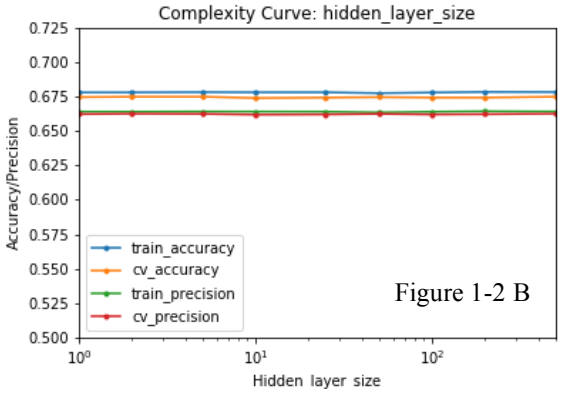
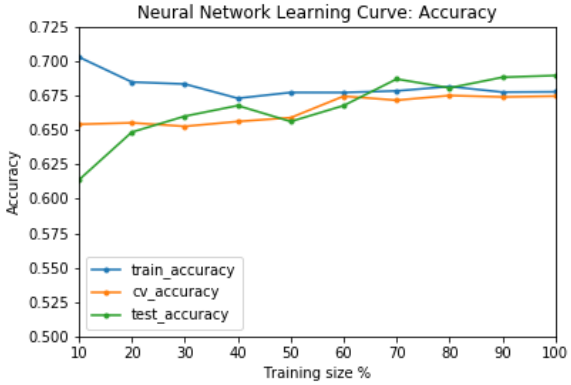


Figure 1-2 B



I note that classifiers with more one hidden layers were also evaluated, but multiple hidden layers do not help to improve the cross-validation accuracy. Therefore, a single hidden layer with 5 neurons in it is the best classifier. The learning curve in Figure 1-2C shows that when the training size smaller than 40%, the cross-validation and test accuracy are lower than the train accuracy and has an increasing trend. When the train size is higher than 40%, the three accuracy curves converge.

Boosting.

AdaBoostClassifier in the sklearn package was used to address this problem. With the same combination of learning_rate and n_estimators (number of estimators), the SAMME.R boosting algorithm always better than the SAMME. After gridsearching (by sklearn.model_selection.GridSearchCV function) the combination between n_estimators and base decision tree depth, the decision tree max_depth = 1gives highest cross-validation accuracy. This result is constant with the theory that as long as the weak learners has better accuracy than random guessing we can combine many of them to have a strong learner. With simpler the base weak learners, there is less chance to overfit the data.

With base decision tree max_depth = 1, the cross-validation accuracy/precision is studied with the different n_estimators. (See the Figure 1-3A) The cross-validation accuracy and precision increase as the classifier gets more complex when the n_estimator value smaller than 10. When n_estimator > 10, the classifier won't gain more performance on the cross-validation results. Interestingly, the training and test accuracy by adaBoosting with decision tree as base learner is not higher than those accuracies obtained by a pre-pruned decision tree. The possible reason I think is: the data itself is noisy, so too many iterations of adding weak estimators won't really train the classifier better, but train the model on noise.

The learning curves are shown in Figure 1-3B. The train accuracy decreases slightly as the train size increases and the test accuracy increases as the train size increasing, and they converge after 30% train size.

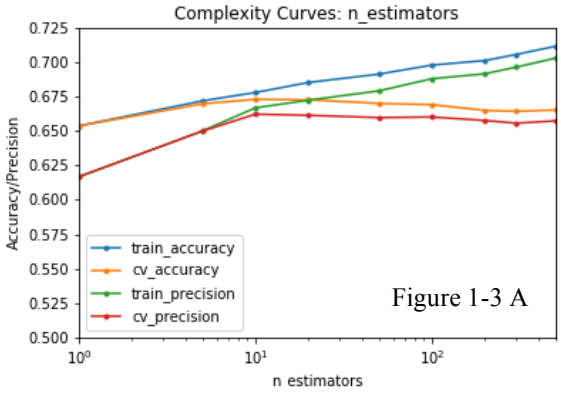


Figure 1-3 A

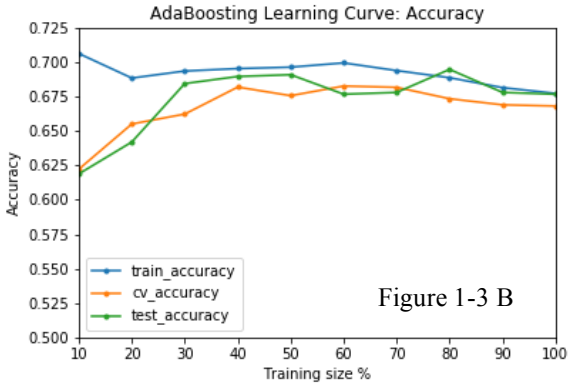


Figure 1-3 B

Support Vector Machines (SVM).

First, I use 'rbf' as the kernel function and explore the penalty parameter C of error term and the gamma which defines how much influence a single training example has. The best C parameter should be 1.0 when gamma = 0.1, considering both the cv_accuracy/precision and train_accuracy/precision, as shown in the Figure 1-4A.

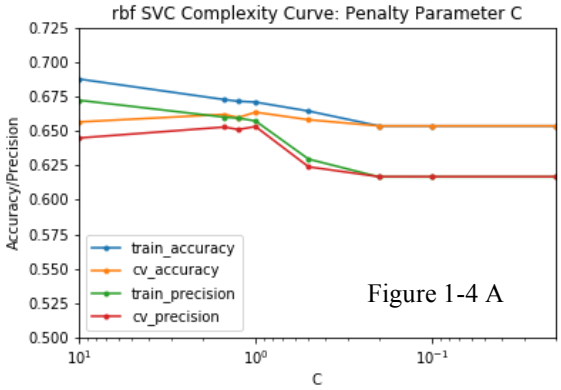


Figure 1-4 A

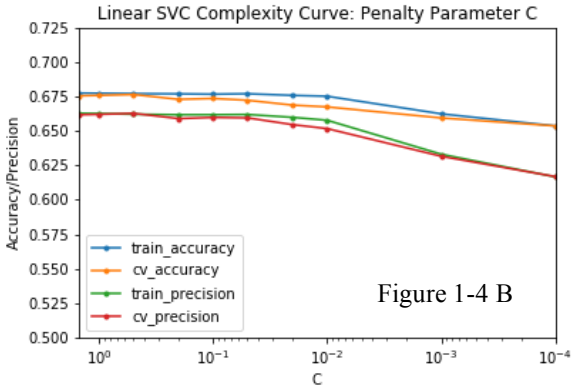


Figure 1-4 B

Next, I explore the support vector classification with linear kernel function. Instead of using the sklearn SVC classifier with parameter kernel = "linear", I used the sklearn LinearSVC classifier, which is similar to SVC with kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples. I studied the penalty terms 'l1' and 'l2' with gridSearch on parameter C and max_iter. I found out that the l1 and l2 penalty has similar effect for the cross-validation score, and the max_iter does not affect the score as long as it is large enough (>= 100). A complexity curve of accuracy/precision against different C parameters is obtained by using the LinearSVC with 'l2', 'squared_hinge', and default max_iter (= 1000). The best C parameter should be 0.5, considering both the cross-validation accuracy/precision and train accuracy/precision, as shown in the Figure 1-4B.

Learning curves with 'rbf' kernel function and linear kernel function are shown in Figure 1-4C and Figure 1-4D, respectively. The accuracy of training, test and cross-validation converges well for both kernel functions, indicating an acceptable variance of both classifiers. However, the linear kernel SVM classifier is able to predict a higher accuracy than the rbf kernel at the full range of training size. This behavior suggests the SVM with linear kernel is more robust with different training size on this dataset.

In summary, using the linear SVM classifier, the highest testing accuracy of this classification problem is 0.6983 with 80% of the training data (training accuracy = 0.6774).

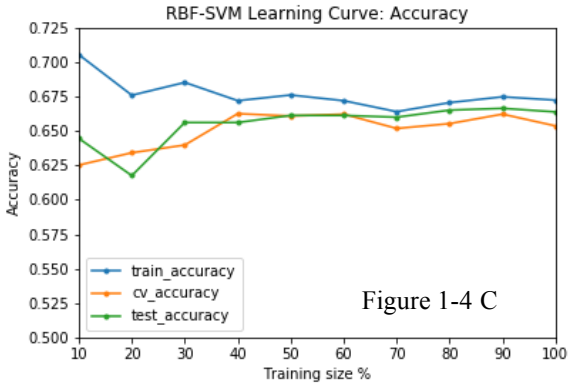


Figure 1-4 C

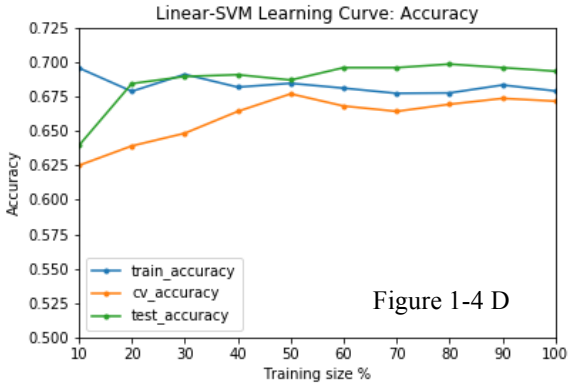


Figure 1-4 D

k-nearest neighbors (k-NN).

I firstly grid-searched the weight functions "uniform" or "distance" ("uniform" weights, all points in each neighborhood are weighted equally; 'distance': weight points by the inverse of their distance.) and a series of "n_neighbors" values. For this dataset, the uniform weight function performs better than the "distance" based weight function. Next, I plot the complexity curve of accuracy/precision vs. n_neighbors. As shown in Figure 1-5A, the training accuracy/precision decreases as the "n_neighbors" value increases, and cross-validation accuracy/precision increases as the "n_neighbors" value increases. The accuracy curves convergence around "n_neighbors" = 100, indicating a best trade-off between bias and variance at this point.

Learning curves with "n_neighbors" = 100 using uniform weight are shown in Figure 1-5B. The accuracy of training, test and cross-validation converges well through the full range of train size. This suggests that the k-NN classifier is robust to classify this data set at various number of train size. When using 70% of the training samples, the test set gets the highest accuracy.

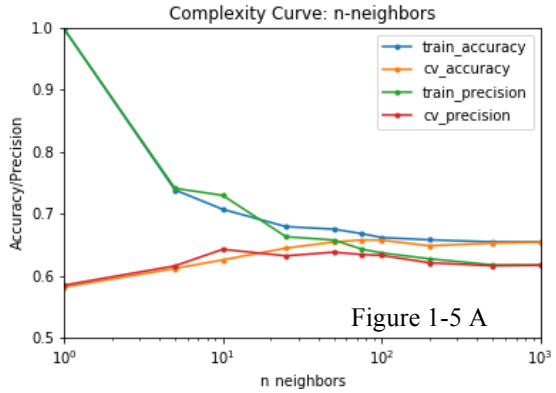


Figure 1-5 A

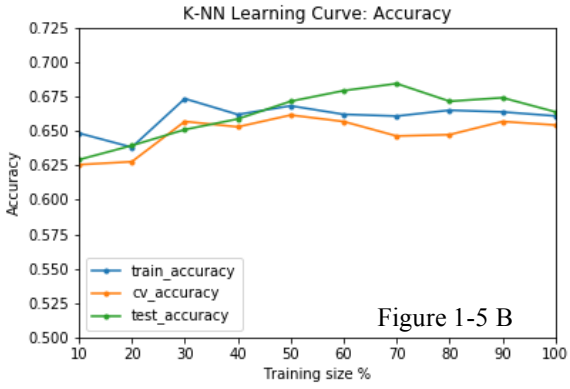


Figure 1-5 B

Dataset 2, classification of page blocks.

Data preprocessing.

The dataset has 5472 samples in five classes: 4913 samples in Class 1, 329 in Class 2, 28 in Class 3, 87 in Class 4 and 115 in Class 5. This is a very unbalanced multiclass dataset. To prevent the Class 1 dominating the classification score calculation, Class 1 was under-sampled to 10% of the original size (491 samples). Even after balancing, the size of Class 3 still much smaller than other classes, but that is the interesting part, I would like to see whether I can find proper classifiers to classify this five classes with dramatically different size. Then, the balanced dataset with 1050 samples was split into train set and test set at 80:20 ratio. In the train set, there are 401 samples in Class 1, 257 samples in Class 2, 21 samples in Class 3, 69 samples in Class 4 and 92 samples in

Class 5. The values of ten numeric attributes of the training set were scaled to zero mean and unit variance by the StandardScaler (in the sklearn.preprocessing). The mean and standard deviation are computed on training set, and then reapply the same transformation on the testing set. During the hyperparameter selection and optimization, k-fold cross-validation is performed only within the training set. The test set is never touched during hyperparameter selection and optimization, but only used for the final test.

Decision Tree.

I firstly train the sklearn DecisionTreeClassifier with the training set with all default hyperparameters, without any pruning. I got a very deep tree with tree depth 16 and 129 tree nodes. The weighted f1 score of training set is 0.9988 and the f1 score of test set is 0.9150. The test score is already good, but the decision tree is pretty deep, some overfitting may exist. To overcome the overfitting, I performed some pre-pruning, such as limit the tree max-depth and minimum number of samples on each leaf (hyperparameter "min_samples_leaf").

Considering the trade-off between bias and variance, the max tree depth should be set at 8, as shown in the Figure 2-1A. Using max_depth = 8, when min_samples_leaf ≥ 5, the cross-validation accuracy decreases as the "min_samples_leaf" value increases (See Figure 2-1B). This behavior is not surprising for this dataset, since the two smallest class only has 21 (Class 3) and 69 samples (Class 4) in the training set. If the samples on each leaf is forced at a large value, then the classes with small number of samples will be merged into the leaves classified as other classes and thus cause a bigger error. Considering the balance between bias and variance, the "min_samples_leaf" = 3 is chosen for this classifier.

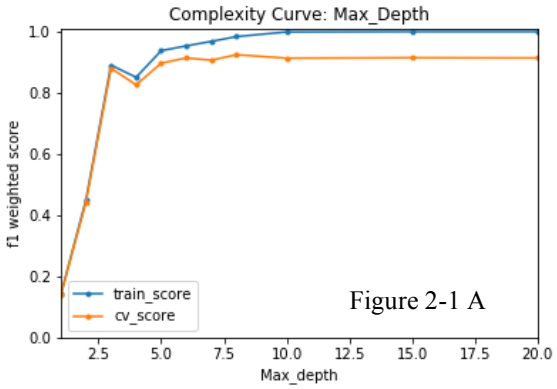


Figure 2-1 A

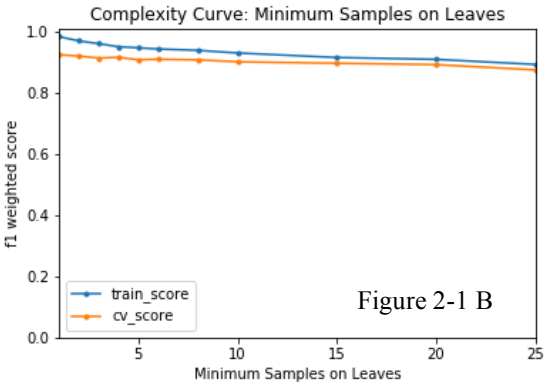


Figure 2-1 B

With this optimized condition and the classifier fitted with all training data (840 samples), the classification accuracy on the test set for individual class are: 0.8778 for Class 1, 0.9444 for Class 2, 1.0 for Class 3, 1.0 for Class 4 and 1.0 for Class 5. Learning curves of f1 score against training size are shown in Figure 2-1 C. The training score fluctuates between 0.9254 and 0.9686 in the full range of training size, and the cross-validation score also shows stable values between 0.8562~ 0.9183. However, the test score is quite low when training size < 60%. With training size > 60 %, the test score (0.8849~ 0.9252) converges with the cross-validation score. The reason for the large difference between train and test score with small training samples is: when training size < 60%, only less than 500 samples are used to build a decision tree with depth 8, so the decision tree is overfitted. If the size of the dataset can be a few times larger than the current size and with uniformly distributed number of samples in each class, the training and test score curves will converge even better.

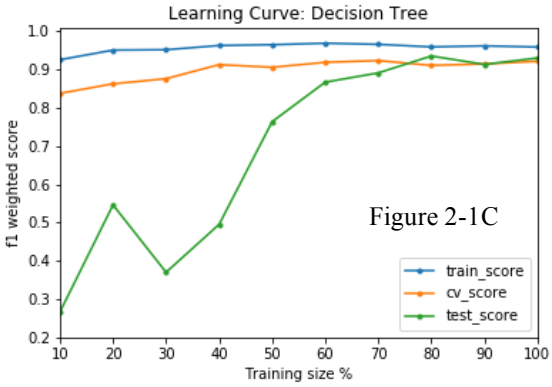


Figure 2-1C

Neural Network.

The MLPClassifier in the sklearn package was used for the neural network classification. Using RandomizedSearchCV to find the combination of hidden layer sizes ("hidden_layer_sizes"), activation functions ("activation"), solvers for weight optimization ("solver"), L2 penalty parameter ("alpha") and batch sizes ("batch_size"), the results indicate solver='lbfgs', activation='identity' always generate good performance, alpha = 0.1, but "batch_size" do not affect the cross-validation score too much.

Using solver='lbfgs', activation='identity', alpha =0.1 and default batch_size, the curve of the training and cross-validation f1 scores against the max iterations (max_iter) was plotted in Figure 2-2A. The training score increases as the max_iter increases when max_iter < 250, and then keeps constant. The cross-validation score reaches the maximum with max_iter = 200, suggesting this is the best point balancing bias and variance. With the tuned hyperparameters and with a single hidden layer, the f1 score at different hidden layer size (the number of neurons in the single hidden layer) was studies. As shown in Figure 2-2B, the train score increases as neuron number increases when hidden_layer_size < 25; the score slowly decreases as the hidden_layer_size increases when hidden_layer_size > 25.

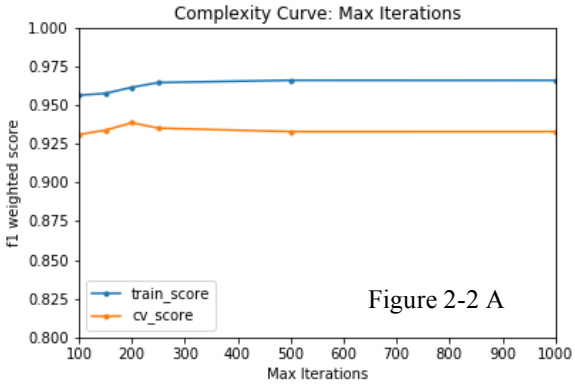


Figure 2-2 A

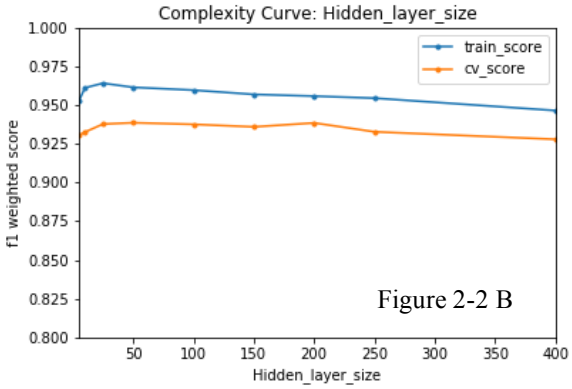


Figure 2-2 B

I note that classifiers with more than one hidden layers were also evaluated, but multiple hidden layers cause lower cross-validation f1 weighted scores and higher train scores, which is a sign of overfitting. Therefore, a single hidden layer with 50 neurons in it is the best classifier. With all the optimized parameters, the test accuracy for individual classes are: 0.9333 for Class 1, 0.9861 for Class 2, 1.0 for Class 3, 0.9444 for Class 4 and 1.0 for Class 5, with the total weighted f1 score 0.9620. The learning curves are shown in Figure 2-2 C. The test score gets close the training score after 50% training size, and the test score and train score converge tightly with training size at 80% or higher.

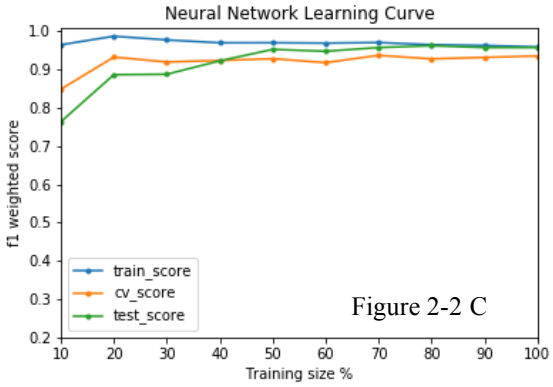


Figure 2-2 C

Boosting.

AdaBoostClassifier in the sklearn package was used to address this problem. After gridsearching the combination between n_estimators, boosting algorithm and the decision tree max_depth, I found that the SAMME.R boosting and learning rate =1.0 always provide stable performance, and the tree max_depth has to be equal to 4 or bigger.

Next, I studied the cross-validation score with a series of max_depth in the base decision tree estimator. When max_depth ≥ 4, there is almost no gain in the classification performance, as shown in Figure 2-3A. With decision tree max_depth = 4, the relation between value of n_estimator and the training and cross-validation f1 score was studied and is plotted in Figure 2-3B. When the n_estimators ≥ 50, the f1 score does not change much as n_estimators value increases. With the classifier trained by all the training samples with tree max_depth = 4 and n_estimators = 100, the test score for individual classes are: 0.9111 for Class 1, 0.9583 for Class 2, 1.0 for Class 3, 0.8889 for Class 4 and 0.9565 for Class 5, with the total weighted f1 score 0.9335.

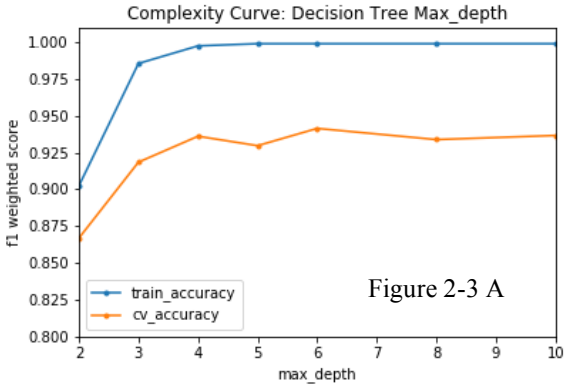


Figure 2-3 A

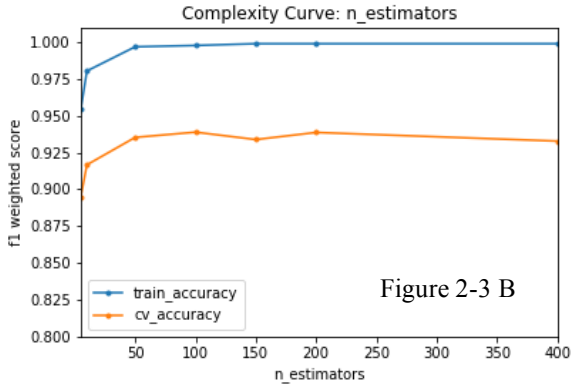
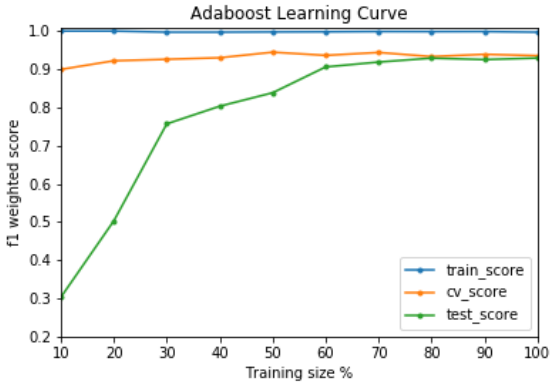


Figure 2-3 B



As shown in Figure 2-3 C, with different training size the train score stays higher than 0.9970, but the cross-validation score always below the training score by 0.0533~0.1000 difference. The test learning curve converges with the cross-validation learning curve after 70% of the train samples. The behaviors of these learning curves of adaBoosting are similar as the those of decision tree.

Support vector machines (SVM).

Firstly, I grid-searched the hypermeters of sklearn.svm.SVC classifier, including: kernel functions ('linear', 'poly', 'rbf' and 'sigmoid'), decision function approaches ("one-vs-one" or "one-vs-rest"), class_weight, C and gamma. The "one-vs-one" and "one-vs-rest" give the same result, but the kernel function 'linear' performs best, followed by 'rbf.

Next, I use 'rbf' as the kernel function and explore the penalty parameter C of error term and the gamma which defines how much influence a single training example has, by cross-validation in sklearn. The best hyperparameter combination is {'C': 100, 'gamma': 0.1}, considering both the cross-validation score and train score, as shown in the Figure 2-4 A. With this optimized condition and the classifier fitted with all training data (840 samples), the classification accuracy on the test set for individual class are: 0.8889 for Class 1, 0.9722 for Class 2, 0.8571 for Class 3, 1.0 for Class 4 and 0.9565 for Class 5.

Next, I explore the support vector classification with linear kernel function. Similar as for the dataset 1, I use the sklearn LinearSVC classifier and studied the penalty terms 'l1' and 'l2' with gridSearch on parameter C and max_iter. I found out that the l1 and l2 penalty has similar effect for the cross-validation score, but l2 penalty when combining with hypermeters {'loss': 'hinge', 'C': 10, 'max_iter' > 1000} works slightly better than l1 penalty. Then, a complexity curve of scores against different C parameters is obtained by using the LinearSVC with 'l2', 'hinge', and max_iter = 2000. The best C parameter should be 10, as shown in the Figure 2-4 B. With this optimized condition and the classifier fitted with all training data (840 samples), the classification accuracy on the test set for individual class are: 0.9222 for Class 1, 0.9305 for Class 2, 1.0 for Class 3, 1.0 for Class 4 and 0.9565 for Class 5.

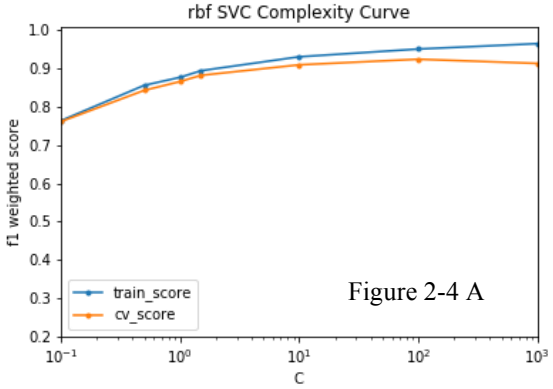


Figure 2-4 A

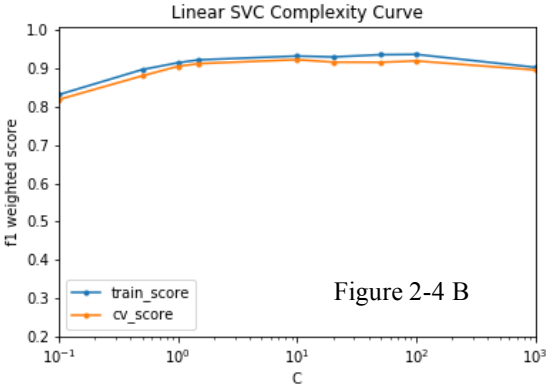


Figure 2-4 B

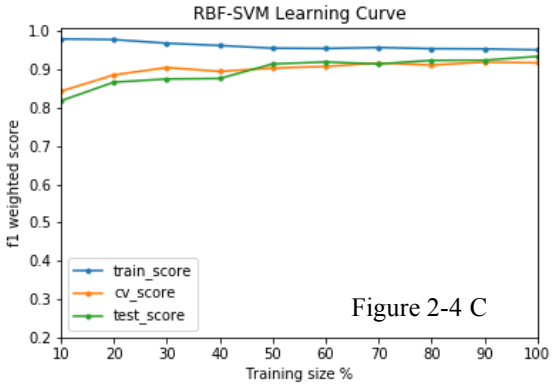


Figure 2-4 C

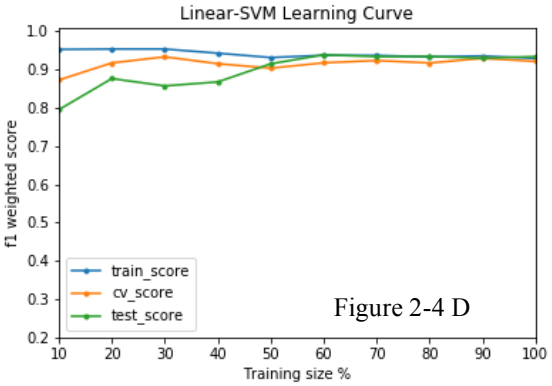


Figure 2-4 D

As shown in Figure 2-4 C and Figure 2-4 D, the learning curves of the test score and training score of linear kernel SVM converge faster and tighter than those of the 'rbf' SVM, indicating the linear SVM is more robust than rbf SVM for this dataset, even with only 50% training samples.

k-nearest neighbors.

Similar as dataset 1, I firstly grid-searched the weight functions "uniform" or "distance" and a series of "n_neighbors" values (the k of k-nearest neighbors) for this dataset. The "distance" weight function performs better than the uniform weight function for this unbalanced multiclass classification problem. Next, I plot the complexity curve of f1 weighted score against the n_neighbors (k). As shown in the Figure 2-5A, the training score (=0.9988) does not change much as the k value varies, but when k > 10 cross-validation accuracy decreases as the k value increases. This behavior is not surprising for this classification problem, because this is an unbalanced dataset with five classes. The two smallest class only has 21 and 69 samples in the training set. If the k value is larger than 10, it is quite easy to include samples of other classes, and thus result in a worse classification for the small classes.

With this optimized condition (k =5, "distance" weight) and the classifier fitted with all training data (840 samples), the classification accuracy on the test set for individual class are: 0.9111 for Class 1, 0.9583 for Class 2, 0.8571 for Class 3, 0.8888 for Class 4 and 0.9565 for Class 5.

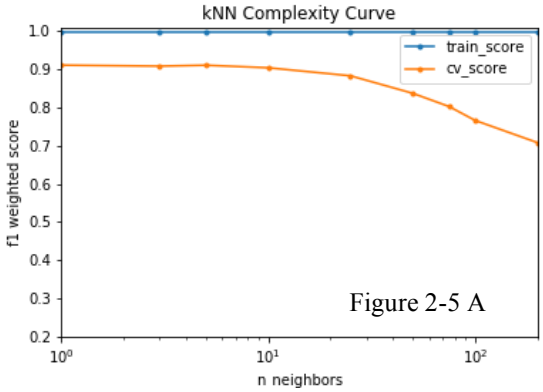


Figure 2-5 A

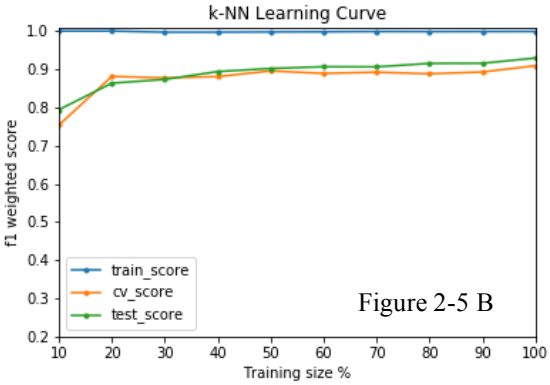


Figure 2-5 B

Learning curves are shown in Figure 2-5B. The train score does not change much for the full range of train size, with only a small drop (from 1.0000 to 0.9988) as the training size increases. The cross-validation score and test score slowly increases as the train size increases. Since only 840 samples were used for the 100% training set, we cannot see the convergence of training and test curves. If the size of the dataset can grow 5 five times larger with uniformly distributed number of samples in each class, the training and test accuracy curves will converge.

Conclusion

Dataset 1, safe/risky loan classification.

The training and test accuracy with 100% training samples obtained by the five learning algorithms are summarized in Table 1. The training accuracy from the five algorithms are very similar, ranging from 0.6607 (by k-NN) to 0.6789 (by SVM with linear kernel). The test accuracy has larger differences, but still the k-NN resulting in the lowest accuracy 0.6637 and the SVM (linear kernel) resulting in the highest accuracy at 0.6932. For all the five learning algorithms, the training and test learning curves converge well with 40% of the training samples or more. Although the k-NN algorithm gives the least training and test accuracy, it is the only algorithm that makes the training and test learning curves converge with any training size (from 10% to 100%).

Using properly tuned hyperparameters such as max_depth of decision tree, neuron numbers in the hidden layers of neural network, number of weak learners of boosting, penalty parameter of SVM and the k value of k-NN, we can obtain classifiers with low variance with the current training size. However, no single learning algorithm is able to provide a high accuracy model. Moreover, from the complexity curves of each algorithm, we can see that classifiers with relatively low complexity show better performance on this dataset. The possible explanations for these findings are:

1) The data in the dataset is noisy, so no matter what algorithms are used they cannot achieve classifiers with high accuracy.
2) The nature of the problem itself. The problem is to predict whether a loan is safe using many types of personal information of the borrower, but we are trying to train the classifier using other people's information and loan results rather than the borrower. However, different people have different life styles and living cost. Two persons have similar annual income and the same credit history may have completely different saving amount and investment strategy, which is not reflected by the attributes of the dataset we have.

Table 1. The training and test accuracy of each learning algorithm for Dataset 1.

	Train Score	Test score
Decision Tree	0.6735	0.6701
Neural Network	0.6775	0.6893
Boosting	0.6771	0.6765
SVM ('rbf')	0.6722	0.6637
SVM(linear)	0.6789	0.6932
k-NN	0.6607	0.6637

Dataset 2, classification of page blocks.

The training and test f1 score with 100% training samples obtained by the five learning algorithms are summarized in Table 2. The neural network classifier is the best classifier for this multiclass classification problem, whose overall training score is 0.9589 and overall test score is 0.9620, and the test accuracy for each individual class is higher than 0.93. Observing the learning curves, we can see that only the training and test curves of neural network and SVM classifiers converge together. The classifiers from other three algorithms are all slightly overfitted with the 840 training samples. The boosting and k-NN have the most significant overfitting, with about 0.07 difference between the training score and the test score.

Table 2. The training and test score of each learning algorithm for Dataset 2.

	Training Score	Test Score	Test Accuracy				
	Total weighted	Total weighted	Class 1	Class 2	Class 3	Class 4	Class 5
Decision Tree	0.9586	0.9299	0.8778	0.9444	1.0000	1.0000	1.0000
Neural Network	0.9589	0.9620	0.9333	0.9861	1.0000	0.9444	1.0000
Boosting	0.9982	0.9335	0.9111	0.9583	1.0000	0.8889	0.9565
SVM ('rbf')	0.9511	0.9342	0.8889	0.9722	0.8571	1.0000	0.9565
SVM (Linear)	0.9306	0.9382	0.9222	0.9305	1.0000	1.0000	0.9565
k-NN	0.9988	0.9293	0.9111	0.9583	0.8571	0.8889	0.9565

The reason why decision tree, boosting (with decision tree as base learner) and k-NN have overfitting for this classification problem is: The Class 3 has very few samples (21 in training set, 7 in test set) comparing to other classes (401 in Class 1, 257 in Class 2, 69 in Class 4 and 92 in

Class 5). To be able to correctly classify this small class, the minimum number of samples on decision tree leaves cannot be too big. Therefore, I cannot apply strong pre-pruning for the decision tree as well as for the base decision tree in boosting, and thus the result classifiers cause high variance for the prediction of the larger classes. Similar story for the k-NN classifier, large k value decrease the prediction accuracy for the small classes, so small k value (k=5) has to be used. With such a small k value, the variance of the classifier is typically large. If there are more samples of the small classes added into the dataset, the decision tree, boosting and k-NN methods will perform better.

Dataset 1 vs. Dataset 2.

These two classification problems are quite different. Dataset 1 is a binary classification problem that contains numerical and categorical attributes, but the existing attributes might not be sufficient to predict the target at high accuracy. The Dataset 2 is a multiclass classification problem that contains ten numerical attributes very well correlated with the prediction target, but have very unbalanced target classes. Because of the different nature of these two datasets, the learning processes of the two datasets favor different model complexity. The dataset 1 favors low complexity models to be able to generalize better. For example, the decision tree for this dataset 1 performs best with max_depth equal to 4 and with more than 50 samples on each leaf, and the adaboosting using the decision tree with max_depth =1 and n_estimators = 10 best balance the bias and variance. More complex decision tree and adaboosting classifiers cause serious overfitting, as shown in Figure 1-1A and 1-3A. The SVM classifier with the linear kernel function works slightly better than other techniques on dataset 1, because this classifier tries to provide a simple decision boundary that maximizes the margins between two classes. However, the dataset 2 favors classifiers with moderate complexity, because it has five classes that need more complex classifiers to classify. The decision tree for this dataset 2 performs best with max_depth equal to 8 and with 3 samples on each leaf, and the boosting performs best using the decision tree with max_depth =4 and n_estimators = 100. The performance of neural network classifier on dataset 2 is outstanding compared to other four learning techniques, but the neural network classifier on dataset 1 performs similarly as other techniques, because a complex network system won't improve the learning accuracy too much on a noisy dataset.