CS 4641 PROJECT 1

Name: CHANG Yingshan
Student ID: 903457645
GT Account: ychang363

Problem Description

1. Letter Recognition

➢ Objective

The objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 images. 16 numerical attributes were measured for each image, which were then scaled to fit into a range of integer values from 0 through 15.

➢ Attribute overview

i. Letter capital letter (26 values from A to Z) —— The correct label
ii. x-box: horizontal position of box
iii. y-box: vertical position of box
iv. width: width of box
v. height: height of box
vi. onpixel: total # on pixels
vii. x-bar: mean x of on pixels in box
viii. y-bar: mean y of on pixels in box
ix. x-var: mean x variance of on pixels in box
x. y-var: mean y variance of on pixels in box
xi. xy-corr: mean x y correlation of on pixels in box
xii. x^2y-bar: mean of x * x * y
xiii. xy^2-bar: mean of x * y * y
xiv. x-edge: mean edge count left to right
xv. x-edge-y-corr: correlation of x-edge with y
xvi. y-edge: mean edge count bottom to top (integer)
xvii. y-edge-x-corr: correlation of y-edge with x

➢ Why interesting

Letter recognition is a common problem in the field of computer vision. The strategy usually used is to consider the brightness of each pixel as “features” and consider an image as a “2D feature map”. Then convolutional neuron network can be applied to extract “patterns” of each image. Image of the same letter should share similar “patterns” and that is what the algorithm is supposed to learn. However, in this dataset, some additional processing work is carried out on the primitive data so that each image is represented by 15 attributes instead of a 2D map of pixels. In the later steps of training and predicting, the algorithm will also totally depend on these 15 generated attributes without looking

back at the original image.

- **Training/validation/testing dataset**

  The whole dataset has 20,000 items. For this assignment, I trained on the first 16,000 items and then used the resulting model to predict the letter category for the remaining 4000 items. Furthermore, the training data is split into training and validation dataset (splitting ratio: 80:20). For each training iteration, the model is trained on 80% of the training data and a validation score is calculated on the remaining 20%. I tried to tune parameters of each ML algorithm according to the average of validation score.

- **Source**

  Creator: David J. Slate

  Odesta Corporation; 1890 Maple Ave; Suite 115; Evanston, IL 60201

  Donor: David J. Slate (dave '@' math.nwu.edu) (708) 491-3867

  URL: http://archive.ics.uci.edu/ml/datasets/Letter+Recognition

2. Titanic

- **Objective**

  The sinking of Titanic is one of the most infamous shipwrecks in history. One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others. This is a binary classification problem aiming at analyzing what sorts of people were likely to survive the tragedy.

- **Attribute overview**

  i. Survival: correct label, 0-No, 1-Yes
  ii. Pclass: ticket class
  iii. Sex: sex of the passenger
  iv. Age: age of the passenger
  v. Sibsp: # of siblings/spouses aboard the Titanic
  vi. Parch: # of parents/children aboard the Titanic
  ~~vii. Ticket: ticket number~~
  viii. Fare: passenger fare
  ~~ix. Cabin: cabin number~~
  x. Embarked: port of embarkation (C=Cherbourg, Q=Queenstown, S=Southampton)
  xi. FamilySize: SibSp + Parch + 1
  xii. IsAlone: if FamilySize
  xiii. Child: binary attribute. child =1 if the passenger is under 18; child = 0 otherwise
  xiv. Female_adult: binary attribute. Female_adult = 1 if the passenger is a female adult; Female_adult = 0 otherwise.
  xv. Male_adult: binary attribute. Male_adult = 1 if the passenger is a male adult; Male_adult = 0 otherwise.

  Note: For the seventh and ninth attribute, there were so many empty entries in the dataset (nearly 70% of them are null values). Therefore, I did not use them for both training and testing. Moreover, the last five attributes are constructed from other existing attributes. To be specific, FamilySize and IsAlone are constructed from SibSp and Parch, while child,

Female_adult and Male_adult are generated from sex and age, which indicate the identity of the passenger.

- **Training/validation/testing dataset**

  This classification problem is from Kaggle competition. The training dataset, with 891 items, is directly downloaded from Kaggle website, which is split into training and validation dataset (splitting ratio: 80:20). For each training iteration, the model is trained on 80% of the training data and a validation score is calculated on the remaining 20%. After finding the optimal parameters according to validation score, the model is trained on the whole training dataset and predictions are made by the resulting model. Since Kaggle does not provide the ground truth for testing data, I used the score generated by Kaggle as testing score after I uploaded the predictions.

- **Source**

  https://www.kaggle.com/c/titanic

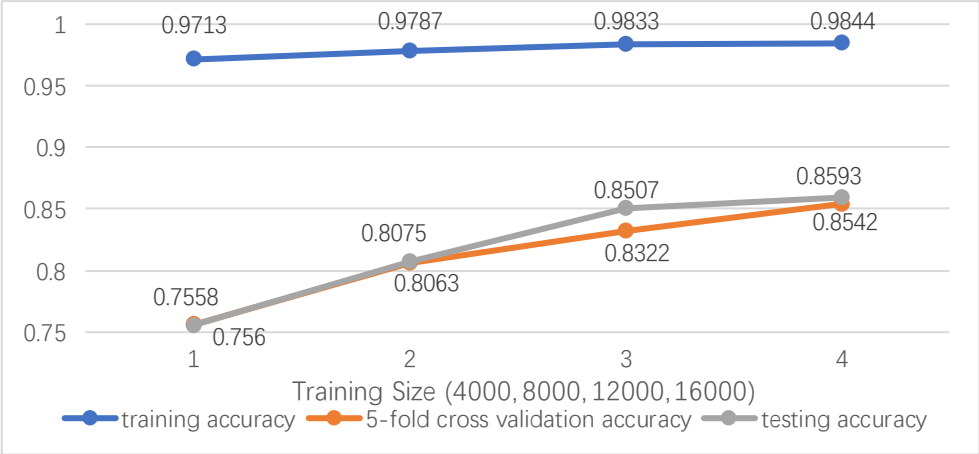
Analysis

1. Decision Tree

- **Performance as a function of training size**

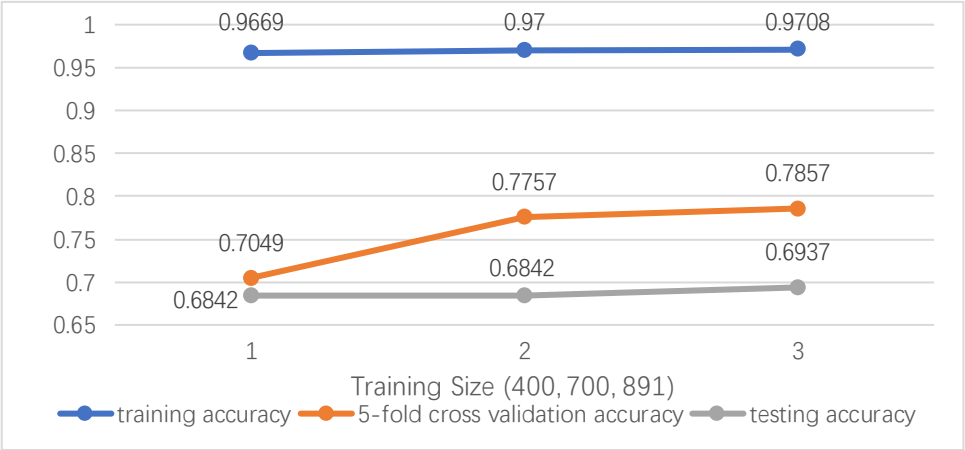
  i. Letter recognition

  - Parameters: min_samples_split=3, criterion='gini', max_depth=50



  ii. Titanic

  - Parameters: min_samples_split=3, criterion='gini', max_depth=50



iii. Analysis

In both problems, the overall performances are better with large training size, since more data provides the models with more clues. Since decision tree is not a very powerful learner, it is less likely to overfit. For powerful learner, it is still possible that the training accuracy decreases as training size increases.

- **Performance of DT using different pruning strategies and splitting criterion**

$$Gini: Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

$$Entropy: H(E) = -\sum_{j=1}^{c} p_j \log p_j$$

Splitting criterion	Min impurity decrease	Training accuracy	Validation accuracy	Testing accuracy	Total running time
'gini'	0	1	0.8560	0.8618	1.37
	0.000005	1	0.8575	0.8650	1.44
	0.00001	1	0.8573	0.8623	1.79
	0.001	0.7204	0.7203	0.6908	1.76
'entropy	0	1	0.8648	0.8680	1.90
	0.000005	1	0.8637	0.8690	1.92
	0.00001	0.9453	0.8651	0.8693	1.89
	0.001	0.8799	0.8197	0.8165	1.98

i. Splitting criterion

When splitting a node into several child nodes, we should choose the best attribute based on the splitting criterion. For 'gini', the best attribute selected for splitting is the one that induces the greatest decrease of gini impurity after splitting. For 'entropy', the best attribute selected for splitting is the one that lead to the largest information gain (entropy before splitting – entropy after splitting). As can be seen from the table, the performance of the two criteria are generally comparable, while the running time of 'gini' is slightly shorter, since it does not require computationally intensive logarithmic functions.

ii. Pruning

My pruning strategy is setting a boundary for 'min_impurity_decrease'. If the amount of impurity decrease induced by a split is lower than the threshold, the node will not be split.

The weighted impurity decrease equation is the following:

N_t / N * (impurity - N_t_R / N_t * right_impurity- N_t_L / N_t * left_impurity

(N is the total number of samples, N_t is the number of samples at the current node,

N_t_L is the number of samples in the left child, and N_t_R is the number of samples in the right child.)

No matter what splitting criterion is used, appropriate level of pruning is desirable since it does help improving the classifier's generalizability. However, aggressive pruning results in underfitting.

2. Neural Network

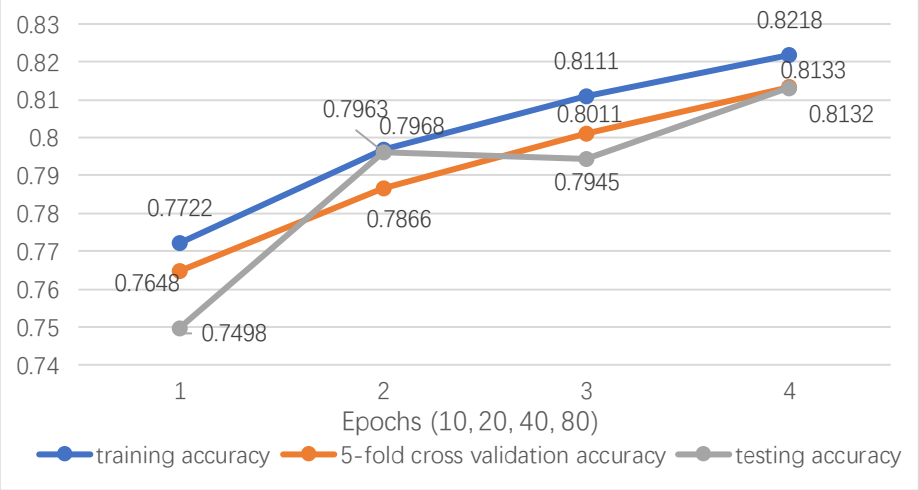
➢ Performance as a function of iterations

i. Letter recognition

- Network structure:

  Dense(output_dim=16, input_dim=15, init='random_uniform', activation='tanh')

  Dense(output_dim=26, activation='softmax')

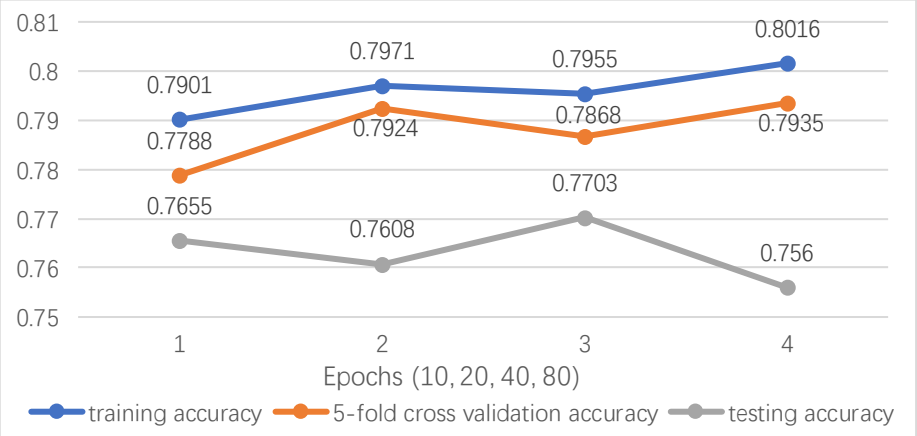


ii. Titanic

- Network structure

  Dense(output_dim=16,input_dim=11,init='random_uniform',activation='sigmoid')

  Dense(output_dim=1, activation='sigmoid')



iii. Analysis

As number of epochs grows, the non-linear function simulated by the neural network will be pushed closer and closer to the true classification boundary. When the training set is large (in the letter recognition problem), overfitting is not revealed. However, when the

training set is small (in the Titanic problem), it starts to overfit when number of epochs reaches 80, which results in a lower testing score.

➢ **Performance of NN with different number of nodes in the hidden layer**

For the letter recognition problem, I explored the influence of hidden_size on the learning outcome. The summary of results is presented below.

Hidden_size	Training accuracy	Validation accuracy	Testing accuracy	Total running time (s)
20	0.8232	0.8126	0.8160	98
300	0.9586	0.9394	0.9335	107
500	0.9570	0.9384	0.9530	115
800	0.9480	0.9312	0.9477	132

As a measure of the network complexity, it turned out that neural network with small hidden_size tend to underfit. Increasing hidden_size gives the network more power to simulate the non-linear function. As a result, both training and testing accuracy rises. However, there is also an optimal point, beyond which the network starts to overfit and the testing score drops.

3. Boosting

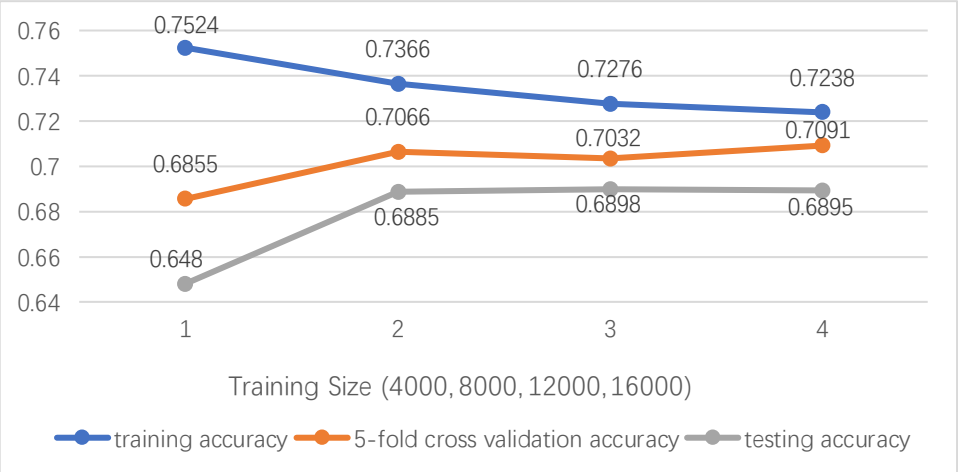
Algorithm: adaptive boosting

Weak learner: decision tree

➢ **Performance as a function of training size**

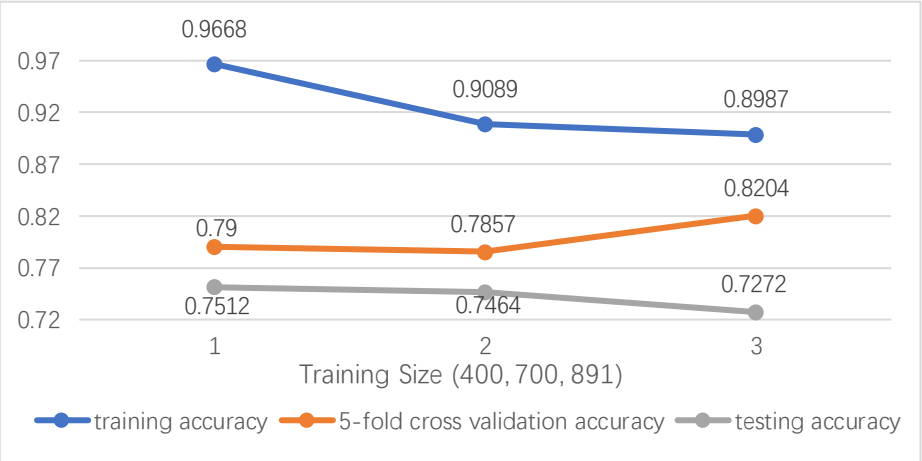
i. Letter recognition

- Parameters: # of estimators=300, learning rate=0.6



ii. Titanic

- Parameters: # of estimators=300, learning rate=0.6



iii. Analysis

For both classification problems, the training accuracy decreases as training size grows. One possible explanation might be that more aggressive pruning (max_depth=2, min_samples_split=3, min_samples_leaf=3) prevents the model from exactly fitting the training data. Therefore, the training accuracy is compromised in order to the generalizability of the model.

➢ Performance of Adaboost with different number of estimators

For the Adaboost algorithm, I changed number of estimators for the purpose of boosting performance. The results are summarized below. (parameters for pruning: max_depth=8, min_samples_split=3, min_samples_leaf=3)

number of estimators	Training accuracy	Validation accuracy	Testing accuracy	Total running time (s)
100	0.9961	0.9345	0.9320	55
300	0.9982	0.9436	0.9490	122
500	0.9987	0.9463	0.9475	193

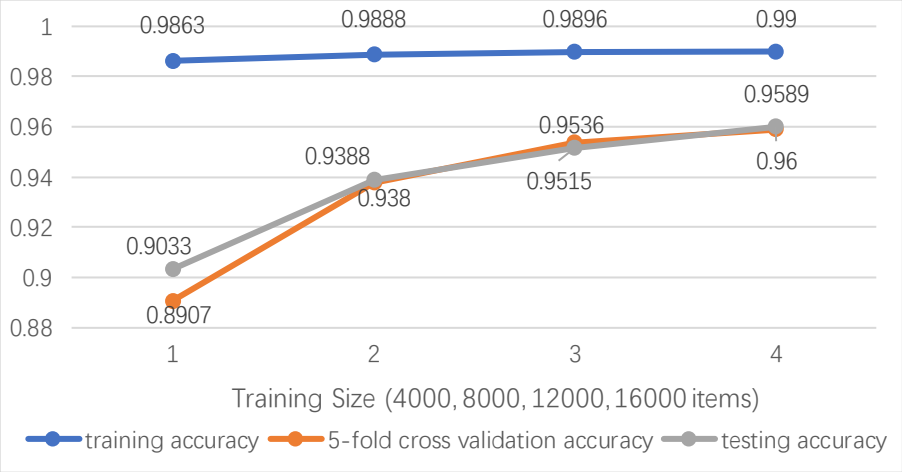
The number of estimators controls the maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early. It is easy to conclude from the table that increasing number of estimators is helpful for improving the learning outcome. However, so many estimators are not desirable since it leads to a sharp increase in running time.

4. Support Vector Machine

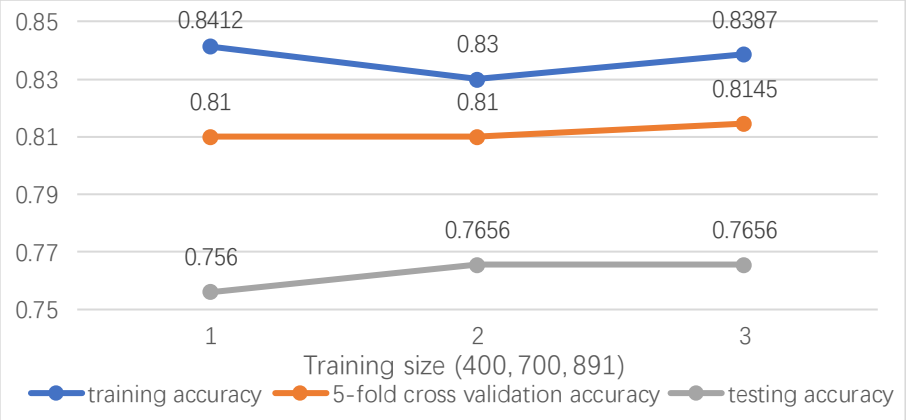
➢ Performance as a function of training size

i. Letter recognition

- Parameters: decision_function_shape =’ovr’, kernel=’rbf’



ii. Titanic



iii. Analysis

It is clear that for both classification problems, as the training size increases, performance of the model is improved. This observation is in accordance with common sense that the more data provided, the easier for the classifier to learn the correct classification criteria.

➢ **Performance of SVM with different parameters**

For the letter recognition problem, I executed the program several times with different parameters. The results are summarized in the following table.

parameters	Training accuracy	Validation accuracy	Testing accuracy	Total running time (s)
Kernel='rbf'	0.9900	0.9589	0.9600	103
Kernel='poly'; Degree=3	0.9900	0.9350	0.9318	62
Kernel='poly'; Degree=5	1	0.9316	0.9326	57
Kernel='poly'; Degree=7	1	0.9295	0.9265	54
Kernel='linear'	0.8540	0.8334	0.8075	68

I used different kernels, and for the polynomial kernel, I also used various degrees to compare the performance of each model. Three insightful conclusions can be drawn from the table.

i. 'rbf' kernel has the best performance, but the training time is also the greatest. The slowness could be attributed to the complex computation of the kernel function.

ii. 'linear' kernel has the worst performance, which indicates that the data would better be classified by a non-linear classifier.

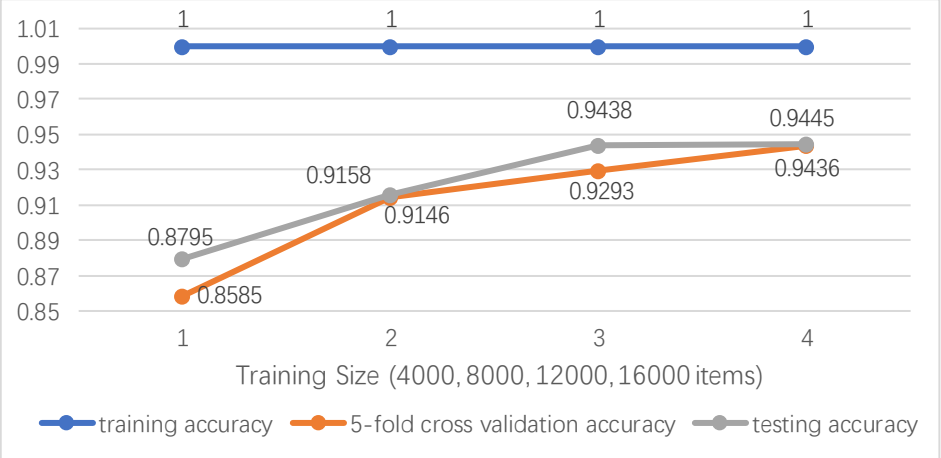
iii. 'poly' kernel also generates satisfactory results. However, as the number of degrees increases, the training accuracy increases to almost 100%, while validation and testing accuracy rise at first and then drop. One possible explanation is that, as the number of degree grows, it is easier to find a hyperplane that correctly separates the data points, but it is also more prone to overfit when the data points are mapped to a high dimension space and, thus, become too sparse, which undermines the overall performance.

5. K-Nearest-Neighbors

➢ Performance as a function of training size

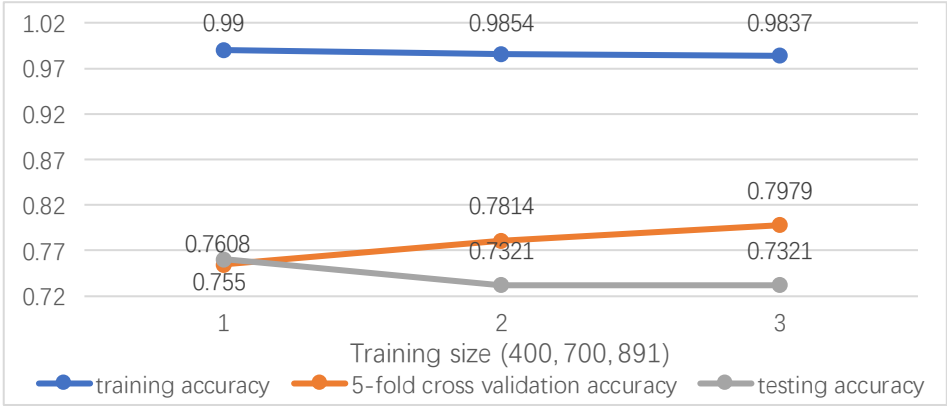
i. Letter recognition

- Parameter: # of neighbors=4



ii. Titanic

- Parameter: # of neighbors = 4



iii. Analysis

For the letter recognition problem, the algorithm produced results as expected: performance is positively correlated with training size. However, it is a little bit weird that for the Titanic problem, the model trained on the smallest dataset got the highest testing score. The imbalance of the dataset might account for this issue. For instance, the first 400 items in the training set are somehow more 'similar' to the items in the testing set. Anyway,

it is important to note that it is not generally true that small training size is likely to result in better performance.

- **Performance of KNN with different parameters**

For the letter recognition problem, different parameters are tried and the performances are evaluated according to validation score. The results are as follows.

weight	# of neighbors	Training accuracy	Validation accuracy	Testing accuracy	Total running time(s)
'distance'	1	1	0.9409	0.9455	5.67
	2	1	0.9400	0.9448	7.21
	3	1	0.9409	0.9438	8.83
	4	1	0.9431	0.9445	9.65
	5	1	0.9405	0.9420	10.27
'uniform'	1	1	0.9414	0.9455	5.28
	2	0.9691	0.9217	0.9270	7.32
	3	0.9739	0.9349	0.9378	8.70
	4	0.9666	0.9327	0.9320	9.22
	5	0.9651	0.9334	0.9320	9.63

The values of 'k' varied from 1 to 5. Plus, in the first five trials, the influence of each neighbor on the prediction result is based on the distance between the neighbor and the new data point. In the last five trials, all neighbors have the same influence on the prediction result.

The summary of knn classifier is twofold. For one thing, looking at more neighbors while classifying a new data point does not always lead to higher accuracy, because neighbors not very close to the new data point should carry less significance. Particularly for the 'uniform' case, including more neighbors results in greater performance decline. For another thing, the more neighbors needed, the longer the total running time. The running time for in 'distance' case is slightly longer than that in 'uniform' case, probably due to the extra effort of distance calculation.

Summary

The table below provides a brief summary of five algorithms for the letter recognition problem.

Algorithm	The best training accuracy	The best testing accuracy	Approximate running time (range)
DT	1	0.8693	1-2s
NN	0.9586	0.9530	90-180s (depends on # of epochs and hidden size)
Boosting	0.9987	0.9490	50-180s (depends on # of estimators)
SVM	1	0.9600	50-100s
KNN	1	0.9455	5-10s

As can be seen, SVM achieves the highest accuracy, followed by NN. Adaboost and KNN also produce good results, while decision tree seems to be the worst classifier for this problem.

Both NN and SVM are capable of simulate non-linear relationships. In neural network, nonlinearity is introduced by non-linear activation functions. the combination of different neurons makes it possible to approximate all kinds of functions. SVM solves non-linear classification problem by projecting data points into higher dimension spaces, in which they can be separated by linear classifiers. When looking back at the data points in lower dimension spaces, it 'seems' that they are separated by non-linear classifiers.

Adaboost also did a good job because it ensembles hundreds of weak learners (decision trees in this case) together to make predictions. Although each weak learner is far less accurate and only makes little contribution, the real strength lies in 'unity'. Additionally, because each weak learner does not need to be very accurate, pruning can be done more aggressively to ensure that the resulting model also performs well on unseen data.

With regard to KNN, to be honest, I am surprised at the high accuracy it achieves. KNN is a really simple algorithm that only looks at all data points while making predictions on the new data without actual 'training process'. Therefore, it is very fast compared with other algorithms that need iterative training or intensive computation. Although this method has multiple defects, the shortcomings have not yet been revealed for this problem. One possible drawback would be that the distance between two points, which determines the influence of a neighbor on the classification result, may not actually represent 'similarity'. In other words, the truly 'similar' objects may be far away from each other in the data space. Nevertheless, this concern does not matter much at least in this case.

Decision tree has relatively poor testing results even though its training accuracy almost reaches 100%. It is a sign of overfitting, but when I tried to do more pruning or set early stopping criteria, the testing accuracy does not rise correspondingly. Perhaps the number of attributes is a limitation for the performance of decision tree. Since only 15 attributes are available and repeatedly using the same attributes is not allowed, the decision tree will quickly run out of attributes.

Lastly, it is also important to note the trade-off between running time and performance. To achieve better performance, intensive computation or iterative training are usually unavoidable. Thus, it is hard to define which algorithm is best without referring to the specific requirements of specific tasks.

Reference

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
https://scikit-learn.org/stable/modules/svm.html
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
https://www.kaggle.com/fffjay/titanic-survivor-predict/notebook
https://medium.com/@rishabhjain_22692/decision-trees-it-begins-here-93ff54ef134
https://blog.csdn.net/am290333566/article/details/81187562#23__CART_159