An Empirical Comparison of Supervised Learning Algorithms

Futian Zhang f6zhang@ucsd.edu
Department of Cognitive Science and Department of Computer Science
University of California,
San Diego, CA 92093, USA

Abstract

More and more machine learning algorithms have come out in the last few decades. Choosing the correct algorithm is critical in practice. This paper compares the performance of three different machine learning algorithms under three different binary classification problems. The machine learning algorithms that we use are: SVMs, logistic regression, and random forest. We use accuracy scores to measure the performance of different algorithms.

1. Introduction

This paper is a replication of previous work of A Empirical Comparison of Supervised Learning Algorithms (Caruana & NiculescuMizil, 2006). Their work compared the performance of a large number of algorithms across many machine learning problems. In this paper, we replicate three algorithms with three different problems that are used in their work.

Machine learning algorithms are used in many areas to solve different problems. These problems differ in sample size, number of attributes etc. Some algorithms may perform well in some problems, but poorly in other problems. For example, SVM is useful to solve problems with high dimensional dataset, while KNN is efficient for solving low dimensional dataset. Therefore, it is necessary to evaluate different algorithms across different problems.

This paper evaluates the accuracy of three algorithms, SVM, logistic regression, and random forest across three problems provided by the UCI Repository. The algorithms are provided by the sklearn package of python. The problems vary in number of attributes, percentage of positive, etc. We convert all problems into binary classification problems.

2. Method

2.1. Learning algorithms

We tune different hyperparameters for different algorithms to get the best model of that algorithm for the training set, then we use that model for testing. This would get us the best performance of that algorithm for each problem. This section shows which hyperparameters we tune for each algorithm.

SVMs: we use the package sklearn.SVM. We tune the hyperparameters of kernel, degree, gamma, and C, including linear, polynomial degree 2&3, and rbf with gamma value of {0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1,2}. The regularization parameter value is 10^i where i is the integers between [-7,3]. We have in total 11*11 = 121 different settings.

Logistic regression (LR): We use package sklearn.linear_model.LogisticRegression. The regularization parameter value is 10^i where i is the integers between [-8,4]. We also tune for without regularization parameter, which is the case of a very large regularization value (sklearn does not have an option for turning off regularization).

Random forest (RF): We use sklearn.ensemble.RandomForestClassifier. The forest has 1024 trees. And we tune max_features with {1,2,4,6,8,12,16,20}.

With SVM and LR, we scale the continued values in the dataset into 0 mean and 1 variance. For RF, we don't scale the attributes. In total, we have more than 120 models for each trial of each problem.

2.2 Performance matrix

We only use accuracy score (ACC), or the percentage of the predicted value is correct, to evaluate the performance of the algorithm. We perform 3 trials for each algorithm and problem pair. The accuracy score is the average of 3 trials.

2.3 Dataset

We use 3 binary classification problems: ADULT, COV_TYPE, LETTER, from the UCI Repository (Blake & Merz, 1998). For each problem, we take 5000 samples as the training set and the rest are the test set. For ADULT, it has 14 attributes, and 25% of data is positive. Some of them are categorized data, we change them into binary data, which has 105 attributes in total. For COV_TYPE, it is a categorization problem, we turn it into a classification problem by setting the largest class as the positive case and the rest as the negative case. 36% of the data is positive. For LETTER, we also turn the categorization problem into classification problem by setting letter 'O' as the positive case and other as negative case. Only 3% of the data is positive.

3. Experiment

For each problem, we randomly select 5000 samples from the dataset and perform a 5-fold cross validation on each training set. Each time, we have 4000 samples for training and 1000 samples for validation. Then we compute the average score for each hyperparameter and the highest score would be the best model. We use the best model to predict on the rest of the dataset. This gives us the ACC score for that trial for the algorithm and problem pair. We then compute the average of 3 trials for each algorithm and problem pair to get the ACC score for each algorithm and problem pair. This is the performance metrics to indicate the performance of that algorithm on each problem (Table 1) and the average performance of all algorithms (Table 2). See the Appendix 1(Table 4) for the raw test score.

Table 1: Average ACC score for each algorithm on each problem

Model	ADULT	COV	LETTER
RF	**0.850**	**0.822**	0.988
SVM	0.849*	0.805	**0.993**
LR	0.845	0.752	0.963

Annotation: T-test score between algorithms (T-test score are in appendix 2:Table 5):

ADULT:

RF-SVM ($p=0.851$) shows no significant difference across two algorithms ($p>0.05$).

SVM-LR ($p=0.003$) and LR-RF ($p=0.044$) shows significant differences across two algorithms ($p<0.05$).

COV:

RF-SVM ($p<0.01$), SVM-LR ($p<0.01$), LR-RF ($p<0.01$) all show significant differences across two algorithms ($p<0.05$).

LETTER

RF-SVM ($p=0.033$), SVM-LR ($p<0.01$), LR-RF ($p<0.01$) all show significant differences across two algorithms ($p<0.05$).

Table 2: Average ACC score for each algorithm

Model	ACC
RF	**0.887**
SVM	0.882*
LR	0.853*

Annotation: T-test score between algorithms (T-test score are in appendix 2: Table 5):

RF-SVM ($p=0.912$), SVM-LR ($p<0.488$), LR-RF ($p<0.523$) all show no significant differences across two algorithms ($p<0.05$).

In the above tables, the algorithm with the best performance would be mark **bold**. And the algorithms whose t-test result shows that it has no significant difference with the highest performance's one would be marked with *.

We observe the higher |positive rate - negative rate| a problem has, the higher the average performance of that problem is. Therefore, these algorithms would generally perform well in LETTER, since it has only a 3% positive rate.

Also, random forest performs the best among all three algorithms in terms of mean score. SVM is the second and LR is the third. However, the three algorithms all show no significant difference across any two of them. And SVM has a significantly higher score in LETTER problems than RF. Therefore, the best algorithm depends on the problem.

The runtime of three algorithms are different. Logistic regression is the fastest, random forest is the second, and SVM is the last because it has to run a lot of hyperparameters. Also, SVM gets slower when C is larger. Random forest gets slower when max_feature gets larger. Logistic regression is fast regardless of C value. If we want to get the outcome as fast as possible, we should use logistic regression.

Therefore, there is no algorithm that is the best among the three. It depends on the problem and the expected runtime.

In addition to test set performance, we also calculate the mean training set performance (Table 3). We use average accuracy scores to represent the performance. As the table indicates, the mean training set performance is higher than the test set performance. This indicates the existence of generalization error. However, the random forest has relatively high generalization error such that the training set performance is 1. SVM has some generalization error while logistic regression has almost no generalization error, which means that the training set performance is almost equal to the test set performance. In LETTER, the logistic regression test score is even higher than the training score. We think this is due to chance, but it shows how low the generalization error it has.

Table 3: Average training set score for each algorithm for each problem

Model	ADULT	COV	LETTER
RF	1	1	1
SVM	0.873	0.902	0.999
LR	0.849	0.757	0.962

4. Conclusion

Overall, random forest seems to be the best algorithm among the three. SVM is the second and logistical regression is the last. However, the performance varies on different problems. For example, random forest is better than SVM on COV, but worse than SVM on LETTER. This result matches the outcome of CMN006 (Caruana & NiculescuMizil, 2006). We also find that logistic regression takes the shortest time to run, random forest is the second and SVM is the slowest.

Bonus point: I did run time analysis.

Appendix

1. Table 4: Raw test set score for each algorithm on each problem

Model	ADULT			COV			LETTER		
RF	0.851	0.851	0.846	0.822	0.822	0.824	0.989	0.986	0.990
SVM	0.850	0.848	0.850	0.804	0.807	0.805	0.994	0.991	0.993
LR	0.844	0.847	0.845	0.752	0.749	0.753	0.962	0.963	0.962

2. Table 5: p-value for each pair of algorithms in Table 1 and Table 2

Model	ADULT	COV	LETTER	Mean
RF-SVM	0.8507	<0.0001	0.0333	0.9120
SVM-LR	0.0031	<0.0001	<0.0001	0.4883
LR-RF	0.0442	<0.0001	<0.0001	0.5227

Note: I used the raw raw data (with about 10 decimal places) to compute the above data, not the ones in the table 4 because it has already been rounded to 3 decimal places.

References

Blake, C., & Merz, C. (1998). UCI repository of machine learning databases.

Caruana R, Niculescu-Mizil A (2006) An empirical comparison of supervised learning algorithms. In: *Proceedings of the 23rd international conference on machine learning*, ICML ’06. ACM, New York, NY, USA, pp 161–168

In [1]:
```
# import all require packages
import numpy as np
import pandas as pd
import random
from sklearn import preprocessing
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
# Reading the data set and making all categorical data to binary
names = ["A" + str(s) for s in range(15)]
adult_data = pd.read_csv("adult.data", header=None, names=names)
adult_data = pd.get_dummies(adult_data)

# Drop all rows with "?" data and the negation of y
for i in names:
    if adult_data.columns.contains(i+ "_ ?"):
        adult_data = adult_data[adult_data[i+ "_ ?"] != 1 ]
        adult_data = adult_data.drop(columns=[i+ "_ ?"])
adult_data = adult_data.drop(columns=["A14_ <=50K"])
```

In [3]:
```
# Do random forest in the first place because we don't scale data in random forest
from sklearn.ensemble import RandomForestClassifier

# Function to calculate average which will be used later
def average(l):
    return sum(l) / len(l)

# For random forest, all possible choice of hyperparameters are listed here
max_features_list =  [1,2,4,6,8,12,16,20]

# For record accuracy score for each trial
trial_acc_score = []
training_acc_score = []

# For each trial
for trial in range(3):
    model_list_whole = []
    acc_list_whole = []

    # Randomly select 5000 samples
    sample_list = []
    for i in range(5000):
        new_value = random.randint(0, len(adult_data)-1)
        while new_value in sample_list:
            new_value = random.randint(0, len(adult_data)-1)
        sample_list.append(new_value)

    # Divide the 5000 sample into 5-fold
    x = adult_data.iloc[sample_list, 0:105]
    y = adult_data.iloc[sample_list, [105]]
    n_folds = 5
    n_per_fold = int(len(x)/n_folds)
    fold_index = list(np.arange(5))
    x_fold = []
    y_fold = []
    for i in fold_index:
        x_fold.append(x[i*n_per_fold:(i*n_per_fold)+n_per_fold])
        y_fold.append(y[i*n_per_fold:(i*n_per_fold)+n_per_fold])

    # For each fold of training and testing, divide them into training and testing set
    for i in range(n_folds):
        x_test = x_fold[i]
        y_test = y_fold[i]
        fold_index = list(np.arange(5))
        fold_index.remove(i)
        x_train = np.empty((0,105))
        y_train = np.empty((0,1))
        for j in fold_index:
            x_train = np.append(x_train,x_fold[j],0)
            y_train = np.append(y_train,y_fold[j],0)

        model_list = []
        acc_list = []

        # For each hyperparameter
        for max_features in max_features_list:
```

```
            # Set up the model, train the data, and test on test set
            model = RandomForestClassifier(n_estimators=1024, max_features=max_features)
            model.fit(x_train, y_train)
            y_predict = model.predict(x_test)
            acc = accuracy_score(y_test, y_predict)

            # Record the model and accuracy score
            model_list.append(model)
            acc_list.append(acc)

        # Record each score into a 2-d array
        model_list_whole.append(model_list)
        acc_list_whole.append(acc_list)

    # Find out the best model and train all sample data
    acc_average_list = list(map(average, zip(*acc_list_whole)))
    max_index = acc_average_list.index(max(acc_average_list))
    best_model = model_list_whole[0][max_index]
    best_model.fit(x, y)

    # Get the accuracy on the training data
    y_train_predict = best_model.predict(x)
    acc = accuracy_score(y, y_train_predict)
    training_acc_score.append(acc)
    print("Training ACC score: ", acc)

    # Test the accuracy on all rest of data, record the accuracy score for this trial
    x_test = adult_data.drop(adult_data.index[sample_list]).iloc[:, 0:105]
    y_test = adult_data.drop(adult_data.index[sample_list]).iloc[:, [105]]
    y_predict = best_model.predict(x_test)
    acc = accuracy_score(y_test, y_predict)
    trial_acc_score.append(acc)
    print("Test ACC score: ", acc)
```

```
Training ACC score:  1.0
Test ACC score:  0.8514426516175185
Training ACC score:  1.0
Test ACC score:  0.8509259995230903
Training ACC score:  1.0
Test ACC score:  0.8464351005484461
```

In [4]:

```
# The final average accuracy score
rf_acc = average(trial_acc_score)
rf_acc
```

Out[4]: 0.8496012505630183

In [5]:

```
# normalize all numerical columns with mean 0 and var 1
for col in ["A0","A2","A4","A10","A11","A12"]:
    adult_data[col] = adult_data[col].astype(float)
    adult_data[col] = preprocessing.scale(adult_data[[col]])
```

In [6]:
```
# Now do SVM
from sklearn import svm

# For SVM, all possible choice of hyperparameters are listed here
C_list = [10**i for i in range(-7,4)]
kernel_list = ['linear', 'poly', 'rbf']
width_list = [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1,2]

# For record accuracy score for each trial
trial_acc_score = []
training_acc_score = []

# For each trial
for trial in range(3):
    model_list_whole = []
    acc_list_whole = []

    # Randomly select 5000 samples
    sample_list = []
    for i in range(5000):
        new_value = random.randint(0, len(adult_data)-1)
        while new_value in sample_list:
            new_value = random.randint(0, len(adult_data)-1)
        sample_list.append(new_value)

    # Divide the 5000 sample into 5-fold
    x = adult_data.iloc[sample_list, 0:105]
    y = adult_data.iloc[sample_list, [105]]
    n_folds = 5
    n_per_fold = int(len(x)/n_folds)
    fold_index = list(np.arange(5))
    x_fold = []
    y_fold = []
    for i in fold_index:
        x_fold.append(x[i*n_per_fold:(i*n_per_fold)+n_per_fold])
        y_fold.append(y[i*n_per_fold:(i*n_per_fold)+n_per_fold])

    # For each fold of training and testing, divide them into training and testing set
    for i in range(n_folds):
        x_test = x_fold[i]
        y_test = y_fold[i]
        fold_index = list(np.arange(5))
        fold_index.remove(i)
        x_train = np.empty((0,105))
        y_train = np.empty((0,1))
        for j in fold_index:
            x_train = np.append(x_train,x_fold[j],0)
            y_train = np.append(y_train,y_fold[j],0)

        model_list = []
        acc_list = []

        # For each hyperparameter
        for C in C_list:
            for kernel in kernel_list:
                if kernel == "poly":
```

```
                # For "poly" kernel, the degree could be 2 or 3
                for degree in [2,3]:

                    # Set up the model, train the data, and test on test set
                    model = svm.SVC(C=C, kernel=kernel, degree=degree, gamma="scale")
                    model.fit(x_train, y_train)
                    y_predict = model.predict(x_test)
                    acc = accuracy_score(y_test, y_predict)

                    # Record the model and accuracy score
                    model_list.append(model)
                    acc_list.append(acc)

            elif kernel == 'rbf':
                for width in width_list:
                    # Set up the model, train the data, and test on test set
                    model = svm.SVC(C=C, kernel=kernel, gamma=width)
                    model.fit(x_train, y_train)
                    y_predict = model.predict(x_test)
                    acc = accuracy_score(y_test, y_predict)

                    # Record the model and accuracy score
                    model_list.append(model)
                    acc_list.append(acc)
            else:

                # Set up the model, train the data, and test on test set
                model = svm.SVC(C=C, kernel=kernel, gamma="scale")
                model.fit(x_train, y_train)
                y_predict = model.predict(x_test)
                acc = accuracy_score(y_test, y_predict)

                # Record the model and accuracy score
                model_list.append(model)
                acc_list.append(acc)

        # Record each score into a 2-d array
        model_list_whole.append(model_list)
        acc_list_whole.append(acc_list)

    # Find out the best model and train all sample data
    acc_average_list = list(map(average, zip(*acc_list_whole)))
    max_index = acc_average_list.index(max(acc_average_list))
    best_model = model_list_whole[0][max_index]
    best_model.fit(x, y)

    # Get the accuracy on the training data
    y_train_predict = best_model.predict(x)
    acc = accuracy_score(y, y_train_predict)
    training_acc_score.append(acc)
    print("Training ACC score: ", acc)

    # Test the accuracy on all rest of data, record the accuracy score for this trial
    x_test = adult_data.drop(adult_data.index[sample_list]).iloc[:, 0:105]
    y_test = adult_data.drop(adult_data.index[sample_list]).iloc[:, [105]]
    y_predict = best_model.predict(x_test)
    acc = accuracy_score(y_test, y_predict)
```

```
    trial_acc_score.append(acc)
    print("Test ACC score: ", acc)
```

```
Training ACC score:  0.8656
Test ACC score:  0.8497337254590255
Training ACC score:  0.8878
Test ACC score:  0.848382481519752
Training ACC score:  0.8642
Test ACC score:  0.8496939829902234
```

In [7]:
```
# The final average accuracy score
svm_acc = average(trial_acc_score)
svm_acc
```

Out[7]:
```
0.8492700633230003
```

In [8]:

```
# Now do the same for logistic regression
from sklearn.linear_model import LogisticRegression

# For logistic regression, all possible choice of hyperparameters are listed here
# Note that a very large number of C is for no regularization
C_list = [10**i for i in range(-8,5)] + [10**10]

# For record accuracy score for each trial
trial_acc_score = []
training_acc_score = []

# For each trial
for trial in range(3):
    model_list_whole = []
    acc_list_whole = []

    # Randomly select 5000 samples
    sample_list = []
    for i in range(5000):
        new_value = random.randint(0, len(adult_data)-1)
        while new_value in sample_list:
            new_value = random.randint(0, len(adult_data)-1)
        sample_list.append(new_value)

    # Divide the 5000 sample into 5-fold
    x = adult_data.iloc[sample_list, 0:105]
    y = adult_data.iloc[sample_list, [105]]
    n_folds = 5
    n_per_fold = int(len(x)/n_folds)
    fold_index = list(np.arange(5))
    x_fold = []
    y_fold = []
    for i in fold_index:
        x_fold.append(x[i*n_per_fold:(i*n_per_fold)+n_per_fold])
        y_fold.append(y[i*n_per_fold:(i*n_per_fold)+n_per_fold])

    # For each fold of training and testing, divide them into training and testing set
    for i in range(n_folds):
        x_test = x_fold[i]
        y_test = y_fold[i]
        fold_index = list(np.arange(5))
        fold_index.remove(i)
        x_train = np.empty((0,105))
        y_train = np.empty((0,1))
        for j in fold_index:
            x_train = np.append(x_train,x_fold[j],0)
            y_train = np.append(y_train,y_fold[j],0)

        model_list = []
        acc_list = []

        # For each hyperparameter
        for C in C_list:

            # Set up the model, train the data, and test on test set
            model = LogisticRegression(C=C)
```

```
            model.fit(x_train, y_train)
            y_predict = model.predict(x_test)
            acc = accuracy_score(y_test, y_predict)

            # Record the model and accuracy score
            model_list.append(model)
            acc_list.append(acc)

        # Record each score into a 2-d array
        model_list_whole.append(model_list)
        acc_list_whole.append(acc_list)

    # Find out the best model and train all sample data
    acc_average_list = list(map(average, zip(*acc_list_whole)))
    max_index = acc_average_list.index(max(acc_average_list))
    best_model = model_list_whole[0][max_index]
    best_model.fit(x, y)

    # Get the accuracy on the training data
    y_train_predict = best_model.predict(x)
    acc = accuracy_score(y, y_train_predict)
    training_acc_score.append(acc)
    print("Training ACC score: ", acc)

    # Test the accuracy on all rest of data, record the accuracy score for this trial
    x_test = adult_data.drop(adult_data.index[sample_list]).iloc[:, 0:105]
    y_test = adult_data.drop(adult_data.index[sample_list]).iloc[:, [105]]
    y_predict = best_model.predict(x_test)
    acc = accuracy_score(y_test, y_predict)
    trial_acc_score.append(acc)
    print("Test ACC score: ", acc)
```

```
Training ACC score:  0.8502
Test ACC score:  0.8437326126698991
Training ACC score:  0.8544
Test ACC score:  0.8456799936412049
Training ACC score:  0.853
Test ACC score:  0.8447261743899531
```

In [ ]:

```
# The final average accuracy score
log_acc = average(trial_acc_score)
log_acc
```

Out[9]: 0.8447129269003524

In [ ]:

```
# Clear the data before to prevent misuse
adult_data = []

# Now let us work on cov type
cov_data = pd.read_csv("covtype.data", header=None)

# Now make the most index positive and the rest negative
cov_data[54][cov_data[54] != 2] = 0
cov_data[54][cov_data[54] == 2] = 1
```

In [ ]:
```
# Do random forest in the first place because we don't scale data in random forest
from sklearn.ensemble import RandomForestClassifier

# Function to calculate average which will be used later
def average(l):
    return sum(l) / len(l)

# For random forest, all possible choice of hyperparameters are listed here
max_features_list =  [1,2,4,6,8,12,16,20]

# For record accuracy score for each trial
trial_acc_score = []
training_acc_score = []

# For each trial
for trial in range(3):
    model_list_whole = []
    acc_list_whole = []

    # Randomly select 5000 samples
    sample_list = []
    for i in range(5000):
        new_value = random.randint(0, len(cov_data)-1)
        while new_value in sample_list:
            new_value = random.randint(0, len(cov_data)-1)
        sample_list.append(new_value)

    # Divide the 5000 sample into 5-fold
    x = cov_data.iloc[sample_list, 0:54]
    y = cov_data.iloc[sample_list, [54]]
    n_folds = 5
    n_per_fold = int(len(x)/n_folds)
    fold_index = list(np.arange(5))
    x_fold = []
    y_fold = []
    for i in fold_index:
        x_fold.append(x[i*n_per_fold:(i*n_per_fold)+n_per_fold])
        y_fold.append(y[i*n_per_fold:(i*n_per_fold)+n_per_fold])

    # For each fold of training and testing, divide them into training and testing set
    for i in range(n_folds):
        x_test = x_fold[i]
        y_test = y_fold[i]
        fold_index = list(np.arange(5))
        fold_index.remove(i)
        x_train = np.empty((0,54))
        y_train = np.empty((0,1))
        for j in fold_index:
            x_train = np.append(x_train,x_fold[j],0)
            y_train = np.append(y_train,y_fold[j],0)

        model_list = []
        acc_list = []

        # For each hyperparameter
        for max_features in max_features_list:
```

```
            # Set up the model, train the data, and test on test set
            model = RandomForestClassifier(n_estimators=1024, max_features=max_features)
            model.fit(x_train, y_train)
            y_predict = model.predict(x_test)
            acc = accuracy_score(y_test, y_predict)

            # Record the model and accuracy score
            model_list.append(model)
            acc_list.append(acc)

        # Record each score into a 2-d array
        model_list_whole.append(model_list)
        acc_list_whole.append(acc_list)

    # Find out the best model and train all sample data
    acc_average_list = list(map(average, zip(*acc_list_whole)))
    max_index = acc_average_list.index(max(acc_average_list))
    best_model = model_list_whole[0][max_index]
    best_model.fit(x, y)

    # Get the accuracy on the training data
    y_train_predict = best_model.predict(x)
    acc = accuracy_score(y, y_train_predict)
    training_acc_score.append(acc)
    print("Training ACC score: ", acc)

    # Test the accuracy on all rest of data, record the accuracy score for this trial
    x_test = cov_data.drop(cov_data.index[sample_list]).iloc[:, 0:54]
    y_test = cov_data.drop(cov_data.index[sample_list]).iloc[:, [54]]
    y_predict = best_model.predict(x_test)
    acc = accuracy_score(y_test, y_predict)
    trial_acc_score.append(acc)
    print("Test ACC score: ", acc)
```

```
Training ACC score:  1.0
Test ACC score:  0.8217676020638459
Training ACC score:  1.0
Test ACC score:  0.8215349680214995
Training ACC score:  1.0
Test ACC score:  0.8240227634146511
```

In [ ]:

```
# The final average accuracy score
rf_acc = average(trial_acc_score)
rf_acc
```

Out[12]:

```
0.822441777833332
```

In [ ]:

```
# normalize all numerical columns with mean 0 and var 1
for col in range(0,10):
    cov_data[col] = cov_data[col].astype(float)
    cov_data[col] = preprocessing.scale(cov_data[[col]])
```

In [ ]:
```
# Now do SVM
from sklearn import svm

# For SVM, all possible choice of hyperparameters are listed here
C_list = [10**i for i in range(-7,4)]
kernel_list = ['linear', 'poly', 'rbf']

# For record accuracy score for each trial
trial_acc_score = []
training_acc_score = []

# For each trial
for trial in range(3):
    model_list_whole = []
    acc_list_whole = []

    # Randomly select 5000 samples
    sample_list = []
    for i in range(5000):
        new_value = random.randint(0, len(cov_data)-1)
        while new_value in sample_list:
            new_value = random.randint(0, len(cov_data)-1)
        sample_list.append(new_value)

    # Divide the 5000 sample into 5-fold
    x = cov_data.iloc[sample_list, 0:54]
    y = cov_data.iloc[sample_list, [54]]
    n_folds = 5
    n_per_fold = int(len(x)/n_folds)
    fold_index = list(np.arange(5))
    x_fold = []
    y_fold = []
    for i in fold_index:
        x_fold.append(x[i*n_per_fold:(i*n_per_fold)+n_per_fold])
        y_fold.append(y[i*n_per_fold:(i*n_per_fold)+n_per_fold])

    # For each fold of training and testing, divide them into training and testing set
    for i in range(n_folds):
        x_test = x_fold[i]
        y_test = y_fold[i]
        fold_index = list(np.arange(5))
        fold_index.remove(i)
        x_train = np.empty((0,54))
        y_train = np.empty((0,1))
        for j in fold_index:
            x_train = np.append(x_train,x_fold[j],0)
            y_train = np.append(y_train,y_fold[j],0)

        model_list = []
        acc_list = []

        # For each hyperparameter
        for C in C_list:
            for kernel in kernel_list:
                if kernel == "poly":
                    # For "poly" kernel, the degree could be 2 or 3
```

```
                    for degree in [2,3]:

                        # Set up the model, train the data, and test on test set
                        model = svm.SVC(C=C, kernel=kernel, degree=degree, gamma="auto")
                        model.fit(x_train, y_train)
                        y_predict = model.predict(x_test)
                        acc = accuracy_score(y_test, y_predict)

                        # Record the model and accuracy score
                        model_list.append(model)
                        acc_list.append(acc)
                elif kernel == 'rbf':
                    for width in width_list:
                        # Set up the model, train the data, and test on test set
                        model = svm.SVC(C=C, kernel=kernel, gamma=width)
                        model.fit(x_train, y_train)
                        y_predict = model.predict(x_test)
                        acc = accuracy_score(y_test, y_predict)

                        # Record the model and accuracy score
                        model_list.append(model)
                        acc_list.append(acc)
                else:

                    # Set up the model, train the data, and test on test set
                    model = svm.SVC(C=C, kernel=kernel, gamma="scale")
                    model.fit(x_train, y_train)
                    y_predict = model.predict(x_test)
                    acc = accuracy_score(y_test, y_predict)

                    # Record the model and accuracy score
                    model_list.append(model)
                    acc_list.append(acc)

        # Record each score into a 2-d array
        model_list_whole.append(model_list)
        acc_list_whole.append(acc_list)

    # Find out the best model and train all sample data
    acc_average_list = list(map(average, zip(*acc_list_whole)))
    max_index = acc_average_list.index(max(acc_average_list))
    best_model = model_list_whole[0][max_index]
    best_model.fit(x, y)

    # Get the accuracy on the training data
    y_train_predict = best_model.predict(x)
    acc = accuracy_score(y, y_train_predict)
    training_acc_score.append(acc)
    print("Training ACC score: ", acc)

    # Test the accuracy on all rest of data, record the accuracy score for this trial
    x_test = cov_data.drop(cov_data.index[sample_list]).iloc[:, 0:54]
    y_test = cov_data.drop(cov_data.index[sample_list]).iloc[:, [54]]
    y_predict = best_model.predict(x_test)
    acc = accuracy_score(y_test, y_predict)
    trial_acc_score.append(acc)
    print("Test ACC score: ", acc)
```

```
Training ACC score:  0.8874
Test ACC score:  0.8038617251029493
Training ACC score:  0.9314
Test ACC score:  0.806865134754137
Training ACC score:  0.886
Test ACC score:  0.8048200384714207
```

In [ ]:

```
# The final average accuracy score
svm_acc = average(trial_acc_score)
svm_acc
```

Out[15]:

```
0.8051822994428357
```

In [ ]:
```
# Now do the same for logistic regression
from sklearn.linear_model import LogisticRegression

# For logistic regression, all possible choice of hyperparameters are listed here
C_list = [10**i for i in range(-8,5)] + [10**10]

# For record accuracy score for each trial
trial_acc_score = []
training_acc_score = []

# For each trial
for trial in range(3):
    model_list_whole = []
    acc_list_whole = []

    # Randomly select 5000 samples
    sample_list = []
    for i in range(5000):
        new_value = random.randint(0, len(cov_data)-1)
        while new_value in sample_list:
            new_value = random.randint(0, len(cov_data)-1)
        sample_list.append(new_value)

    # Divide the 5000 sample into 5-fold
    x = cov_data.iloc[sample_list, 0:54]
    y = cov_data.iloc[sample_list, [54]]
    n_folds = 5
    n_per_fold = int(len(x)/n_folds)
    fold_index = list(np.arange(5))
    x_fold = []
    y_fold = []
    for i in fold_index:
        x_fold.append(x[i*n_per_fold:(i*n_per_fold)+n_per_fold])
        y_fold.append(y[i*n_per_fold:(i*n_per_fold)+n_per_fold])

    # For each fold of training and testing, divide them into training and testing set
    for i in range(n_folds):
        x_test = x_fold[i]
        y_test = y_fold[i]
        fold_index = list(np.arange(5))
        fold_index.remove(i)
        x_train = np.empty((0,54))
        y_train = np.empty((0,1))
        for j in fold_index:
            x_train = np.append(x_train,x_fold[j],0)
            y_train = np.append(y_train,y_fold[j],0)

        model_list = []
        acc_list = []

        # For each hyperparameter
        for C in C_list:

            # Set up the model, train the data, and test on test set
            model = LogisticRegression(C=C)
            model.fit(x_train, y_train)
```

```
            y_predict = model.predict(x_test)
            acc = accuracy_score(y_test, y_predict)

            # Record the model and accuracy score
            model_list.append(model)
            acc_list.append(acc)

        # Record each score into a 2-d array
        model_list_whole.append(model_list)
        acc_list_whole.append(acc_list)

    # Find out the best model and train all sample data
    acc_average_list = list(map(average, zip(*acc_list_whole)))
    max_index = acc_average_list.index(max(acc_average_list))
    best_model = model_list_whole[0][max_index]
    best_model.fit(x, y)

    # Get the accuracy on the training data
    y_train_predict = best_model.predict(x)
    acc = accuracy_score(y, y_train_predict)
    training_acc_score.append(acc)
    print("Training ACC score: ", acc)

    # Test the accuracy on all rest of data, record the accuracy score for this trial
    x_test = cov_data.drop(cov_data.index[sample_list]).iloc[:, 0:54]
    y_test = cov_data.drop(cov_data.index[sample_list]).iloc[:, [54]]
    y_predict = best_model.predict(x_test)
    acc = accuracy_score(y_test, y_predict)
    trial_acc_score.append(acc)
    print("Test ACC score: ", acc)
```

```
Training ACC score:  0.756
Test ACC score:  0.7518558641139421
Training ACC score:  0.7528
Test ACC score:  0.7494843857419637
Training ACC score:  0.7608
Test ACC score:  0.7532169468691624
```

In [ ]:

```
# The final average accuracy score
log_acc = average(trial_acc_score)
log_acc
```

Out[17]: 0.7515190655750228

In [2]:

```
# Clear the data before to prevent misuse
cov_data = []

# Now let us work on cov type
letter_data = pd.read_csv("letter-recognition.data", header=None)

# Now make the most index positive and the rest negative
letter_data[0][letter_data[0] != 'O'] = 0
letter_data[0][letter_data[0] == 'O'] = 1
letter_data[0] = letter_data[0].astype('int')
```

In [4]:
```
# Do random forest in the first place because we don't scale data in random forest
from sklearn.ensemble import RandomForestClassifier

# For random forest, all possible choice of hyperparameters are listed here
max_features_list =  [1,2,4,6,8,12,16]

# Function to calculate average which will be used later
def average(l):
    return sum(l) / len(l)

# For record accuracy score for each trial
trial_acc_score = []
training_acc_score = []

# For each trial
for trial in range(3):
    model_list_whole = []
    acc_list_whole = []

    # Randomly select 5000 samples
    sample_list = []
    for i in range(5000):
        new_value = random.randint(0, len(letter_data)-1)
        while new_value in sample_list:
            new_value = random.randint(0, len(letter_data)-1)
        sample_list.append(new_value)

    # Divide the 5000 sample into 5-fold
    x = letter_data.iloc[sample_list, 1:17]
    y = letter_data.iloc[sample_list, [0]]
    n_folds = 5
    n_per_fold = int(len(x)/n_folds)
    fold_index = list(np.arange(5))
    x_fold = []
    y_fold = []
    for i in fold_index:
        x_fold.append(x[i*n_per_fold:(i*n_per_fold)+n_per_fold])
        y_fold.append(y[i*n_per_fold:(i*n_per_fold)+n_per_fold])

    # For each fold of training and testing, divide them into training and testing set
    for i in range(n_folds):
        x_test = x_fold[i]
        y_test = y_fold[i]
        fold_index = list(np.arange(5))
        fold_index.remove(i)
        x_train = np.empty((0,16))
        y_train = np.empty((0,1))
        for j in fold_index:
            x_train = np.append(x_train,x_fold[j],0)
            y_train = np.append(y_train,y_fold[j],0)

        model_list = []
        acc_list = []

        # For each hyperparameter
        for max_features in max_features_list:
```

```
            # Set up the model, train the data, and test on test set
            model = RandomForestClassifier(n_estimators=1024, max_features=max_features)
            model.fit(x_train, y_train)
            y_predict = model.predict(x_test)
            acc = accuracy_score(y_test, y_predict)

            # Record the model and accuracy score
            model_list.append(model)
            acc_list.append(acc)

        # Record each score into a 2-d array
        model_list_whole.append(model_list)
        acc_list_whole.append(acc_list)

    # Find out the best model and train all sample data
    acc_average_list = list(map(average, zip(*acc_list_whole)))
    max_index = acc_average_list.index(max(acc_average_list))
    best_model = model_list_whole[0][max_index]
    best_model.fit(x, y)

    # Get the accuracy on the training data
    y_train_predict = best_model.predict(x)
    acc = accuracy_score(y, y_train_predict)
    training_acc_score.append(acc)
    print("Training ACC score: ", acc)

    # Test the accuracy on all rest of data, record the accuracy score for this trial
    x_test = letter_data.drop(letter_data.index[sample_list]).iloc[:, 1:17]
    y_test = letter_data.drop(letter_data.index[sample_list]).iloc[:, [0]]
    y_predict = best_model.predict(x_test)
    acc = accuracy_score(y_test, y_predict)
    trial_acc_score.append(acc)
    print("Test ACC score: ", acc)
```

```
Training ACC score:  1.0
Test ACC score:  0.9891333333333333
Training ACC score:  1.0
Test ACC score:  0.9858
Training ACC score:  1.0
Test ACC score:  0.9895333333333334
```

In [5]:
```
# The final average accuracy score
rf_acc = average(trial_acc_score)
rf_acc
```

Out[5]: 0.9881555555555556

In [3]:
```
# normalize all numerical columns with mean 0 and var 1
for col in range(1,17):
    letter_data[col] = letter_data[col].astype(float)
    letter_data[col] = preprocessing.scale(letter_data[[col]])
```

In [4]:
```
# Now do SVM
from sklearn import svm

# For SVM, all possible choice of hyperparameters are listed here
C_list = [10**i for i in range(-7,4)]
kernel_list = ['linear', 'poly', 'rbf']
width_list = [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1,2]

# Function to calculate average which will be used later
def average(l):
    return sum(l) / len(l)

# For record accuracy score for each trial
trial_acc_score = []
training_acc_score = []

# For each trial
for trial in range(3):
    model_list_whole = []
    acc_list_whole = []

    # Randomly select 5000 samples
    sample_list = []
    for i in range(5000):
        new_value = random.randint(0, len(letter_data)-1)
        while new_value in sample_list:
            new_value = random.randint(0, len(letter_data)-1)
        sample_list.append(new_value)

    # Divide the 5000 sample into 5-fold
    x = letter_data.iloc[sample_list, 1:17]
    y = letter_data.iloc[sample_list, [0]]
    n_folds = 5
    n_per_fold = int(len(x)/n_folds)
    fold_index = list(np.arange(5))
    x_fold = []
    y_fold = []
    for i in fold_index:
        x_fold.append(x[i*n_per_fold:(i*n_per_fold)+n_per_fold])
        y_fold.append(y[i*n_per_fold:(i*n_per_fold)+n_per_fold])

    # For each fold of training and testing, divide them into training and testing set
    for i in range(n_folds):
        x_test = x_fold[i]
        y_test = y_fold[i]
        fold_index = list(np.arange(5))
        fold_index.remove(i)
        x_train = np.empty((0,16))
        y_train = np.empty((0,1))
        for j in fold_index:
            x_train = np.append(x_train,x_fold[j],0)
            y_train = np.append(y_train,y_fold[j],0)

        model_list = []
        acc_list = []
```

```
        # For each hyperparameter
        for C in C_list:
            for kernel in kernel_list:
                if kernel == "poly":
                    # For "poly" kernel, the degree could be 2 or 3
                    for degree in [2,3]:

                        # Set up the model, train the data, and test on test set
                        model = svm.SVC(C=C, kernel=kernel, degree=degree, gamma="auto")
                        model.fit(x_train, y_train)
                        y_predict = model.predict(x_test)
                        acc = accuracy_score(y_test, y_predict)

                        # Record the model and accuracy score
                        model_list.append(model)
                        acc_list.append(acc)

                elif kernel == 'rbf':
                    for width in width_list:
                        # Set up the model, train the data, and test on test set
                        model = svm.SVC(C=C, kernel=kernel, gamma=width)
                        model.fit(x_train, y_train)
                        y_predict = model.predict(x_test)
                        acc = accuracy_score(y_test, y_predict)

                        # Record the model and accuracy score
                        model_list.append(model)
                        acc_list.append(acc)
                else:
                    # Set up the model, train the data, and test on test set
                    model = svm.SVC(C=C, kernel=kernel, max_iter = 10000)
                    model.fit(x_train, y_train)
                    y_predict = model.predict(x_test)
                    acc = accuracy_score(y_test, y_predict)

                    # Record the model and accuracy score
                    model_list.append(model)
                    acc_list.append(acc)

        # Record each score into a 2-d array
        model_list_whole.append(model_list)
        acc_list_whole.append(acc_list)

    # Find out the best model and train all sample data
    acc_average_list = list(map(average, zip(*acc_list_whole)))
    max_index = acc_average_list.index(max(acc_average_list))
    best_model = model_list_whole[0][max_index]
    best_model.fit(x, y)

    # Get the accuracy on the training data
    y_train_predict = best_model.predict(x)
    acc = accuracy_score(y, y_train_predict)
    training_acc_score.append(acc)
    print("Training ACC score: ", acc)

    # Test the accuracy on all rest of data, record the accuracy score for this trial
    x_test = letter_data.drop(letter_data.index[sample_list]).iloc[:, 1:17]
```

```
    y_test = letter_data.drop(letter_data.index[sample_list]).iloc[:, [0]]
    y_predict = best_model.predict(x_test)
    acc = accuracy_score(y_test, y_predict)
    trial_acc_score.append(acc)
    print("Test ACC score: ", acc)
```

```
Training ACC score:  0.9996
Test ACC score:  0.9944
Training ACC score:  0.9988
Test ACC score:  0.9913333333333333
Training ACC score:  0.999
Test ACC score:  0.9928666666666667
```

In [5]:

```
# The final average accuracy score
svm_acc = average(trial_acc_score)
svm_acc
```

Out[5]: 0.9928666666666666

In [6]:
```
# Now do the same for logistic regression
from sklearn.linear_model import LogisticRegression

# Function to calculate average which will be used later
def average(l):
    return sum(l) / len(l)

# For logistic regression, all possible choice of hyperparameters are listed here
C_list = [10**i for i in range(-8,5)] + [10**10]

# For record accuracy score for each trial
trial_acc_score = []
training_acc_score = []

# For each trial
for trial in range(3):
    model_list_whole = []
    acc_list_whole = []

    # Randomly select 5000 samples
    sample_list = []
    for i in range(5000):
        new_value = random.randint(0, len(letter_data)-1)
        while new_value in sample_list:
            new_value = random.randint(0, len(letter_data)-1)
        sample_list.append(new_value)

    # Divide the 5000 sample into 5-fold
    x = letter_data.iloc[sample_list, 1:17]
    y = letter_data.iloc[sample_list, [0]]
    n_folds = 5
    n_per_fold = int(len(x)/n_folds)
    fold_index = list(np.arange(5))
    x_fold = []
    y_fold = []
    for i in fold_index:
        x_fold.append(x[i*n_per_fold:(i*n_per_fold)+n_per_fold])
        y_fold.append(y[i*n_per_fold:(i*n_per_fold)+n_per_fold])

    # For each fold of training and testing, divide them into training and testing set
    for i in range(n_folds):
        x_test = x_fold[i]
        y_test = y_fold[i]
        fold_index = list(np.arange(5))
        fold_index.remove(i)
        x_train = np.empty((0,16))
        y_train = np.empty((0,1))
        for j in fold_index:
            x_train = np.append(x_train,x_fold[j],0)
            y_train = np.append(y_train,y_fold[j],0)

        model_list = []
        acc_list = []

        # For each hyperparameter
        for C in C_list:
```

```
            # Set up the model, train the data, and test on test set
            model = LogisticRegression(C=C)
            model.fit(x_train, y_train)
            y_predict = model.predict(x_test)
            acc = accuracy_score(y_test, y_predict)

            # Record the model and accuracy score
            model_list.append(model)
            acc_list.append(acc)

        # Record each score into a 2-d array
        model_list_whole.append(model_list)
        acc_list_whole.append(acc_list)

    # Find out the best model and train all sample data
    acc_average_list = list(map(average, zip(*acc_list_whole)))
    max_index = acc_average_list.index(max(acc_average_list))
    best_model = model_list_whole[0][max_index]
    best_model.fit(x, y)

    # Get the accuracy on the training data
    y_train_predict = best_model.predict(x)
    acc = accuracy_score(y, y_train_predict)
    training_acc_score.append(acc)
    print("Training ACC score: ", acc)

    # Test the accuracy on all rest of data, record the accuracy score for this trial
    x_test = letter_data.drop(letter_data.index[sample_list]).iloc[:, 1:17]
    y_test = letter_data.drop(letter_data.index[sample_list]).iloc[:, [0]]
    y_predict = best_model.predict(x_test)
    acc = accuracy_score(y_test, y_predict)
    trial_acc_score.append(acc)
    print("Test ACC score: ", acc)
```

```
Training ACC score:  0.9634
Test ACC score:  0.962
Training ACC score:  0.9604
Test ACC score:  0.963
Training ACC score:  0.9622
Test ACC score:  0.9624
```

In [7]:

```
# The final average accuracy score
log_acc = average(trial_acc_score)
log_acc
```

Out[7]: 0.9624666666666667

In [ ]: