Shadman Noor

A12960909

COGS 118A Final Project

Abstract:

This paper uses various different classification techniques on three different datasets and performs analysis on them. The classification techniques that are used are K-Nearest Neighbors, Random Forest, and Logistic Regression. The three datasets that were analyzed are the ADULT dataset, the COV_TYPE dataset and the LETTER dataset from the UCI Repository. A key aspect of my analysis was looking at error scores to access the performance of these different supervised learning algorithms on the datasets.

Introduction:

The study that I have conducted through the analysis of the three datasets, ADULT, COV_TYPE, and LETTER, demonstrate a comparison between the three analysis methods: K-Nearest Neighbors, Logistic Regression, and Random Forest empirically. Since the field of Machine Learning is gaining importance in our world rapidly, more and more new techniques are evolving to perform data analysis. As such, the importance of empirical studies to analyze different methods of analysis are very important. In this study I chose to use ‘accuracy’ as the main error metric for the evaluation of the three different models. The study finds that for different datasets the performance of the algorithms varies. Even though Random Forest performs the best amongst the three different models on average, it doesn’t perform better than the other models on all three of the datasets.

Methods:

The main domain of exploration for the different algorithms were the hyperparameters. We ran our models on the same data with different hyperparameters and found out the best hyperparameter for prediction on the dataset. The models were then trained on the datasets using those best hyperparameters and predictions were made on the test set. The next section of the paper holds information about all the different parameters that were explored in the study for the three algorithms.

K-Nearest Neighbors: For K-Nearest Neighbors we tried 25 different values of K spaced log-evenly across 1 to 500. We weighted the K-Nearest Neighbors algorithm with distance.

Random Forest: We set the Random Forest algorithm to have 1024 trees and we considered different sizes of the feature set at each split. For the ADULT dataset and the COVER_TYPE dataset the sizes of the feature set considered were 1, 2, 4, 6, 8, 12, 16 or 20, but for the LETTER dataset we only considered the sizes to be 1, 2, 4, 6, 8, 12, 16 because LETTER dataset only had 16 features in total.

Logistic Regression: For Logistic Regression we varied the regularization parameter, C by factors of 10 from 10^-8 to 10^4. The solver was set to 'liblinear.'

Accuracy score was chosen as the performance metric for the study . The three algorithms were trained using the best hyperparameters and predictions were made on the test set. Test accuracy was calculated after that and compiled for evaluation of the algorithms. The accuracy score ranges from 0 to 1, where 1 is perfectly accurate and 0 means there is no accuracy.

The datasets that were chosen are ADULT dataset, COV_TYPE dataset and LETTER dataset, which were all obtained from the UCI Repository. For the ADULT dataset, one hot encoding

was used to encode all the categorical entries in the dataset and the labels were changed from ‘ >50k’ and ‘ <=50k’ to 1 and 0. For the COV_TYPE dataset we calculated the most common cover type as 1 and set all the other types as 0. For the LETTER dataset we set letters from A to M to 1 and the rest of the letters as 0. This way all three problems were turned into binary problems.

Five-fold cross validation was used to split the data. A training set size of 5000 was randomly selected for all nine trials while the rest of the samples in the dataset was reserved for test set. All the features of the dataset were standardized using standard scalar so that they have mean of 0 and standard deviation of 1. Three trials were performed on each dataset where all three models were used in each trial.

Experiment:

Table 1 represents the average test set accuracy score from all nine trials across all three datasets. Each cell in the table will be the average test set accuracy score of an algorithm over 3 trials of each of 3 datasets (mean of 9 test set scores).

Table 1

	Average Test Set Accuracy
KNN	0.8401040116266226
Random Forest	0.8725895555807401
Logistic Regression	0.7734688218187457

Similarly, Table 2 reports the average training set accuracy score from all nine trials across all three datasets where each cell in the table will be the average test set accuracy score of an algorithm over 3 trials of each of 3 datasets.

Table 2

	Average Train Set Accuracy
KNN	0.9808888888888888
Random Forest	1.0
Logistic Regression	0.776688888888889

T-test was also chosen to compare the models. T-tests were used with arrays of accuracy scores of each algorithm over three trials to see how similar the performances were between them. After doing three trials on all three datasets, arrays of accuracy scores across all nine trials were used to compare the models with t-tests as well. The p-values for all the algorithms for all nine trails across all three datasets is shown in table 3.

Table 3

	P-value for all nine trials
KNN and Random Forest	0.29296952937840315
Random Forest and Logistic Regression	0.0017340131831185953
Logistic Regression and KNN	0.036380095762982935

From the above tables, the best model overall seems to be Random Forest, which slightly outperforms K-Nearest Neighbors. The poorest performance is shown by Logistic Regression.

Table 4 shows the datasets as the columns, and the algorithms as rows. Each cell is the average test set accuracy over three trials of each dataset.

Table 4

	ADULT Test	COV_TYPE Test	LETTER Test
KNN	0.8217682749119043	0.7717437599679636	0.9268
Random Forest	0.8470444850700792	0.82261307056103	0.9481111111111112
Logistic Regression	0.8440637999099171	0.7520537766574308	0.7242888888888889

Similarly, Table 5 shows the average train set accuracy over three trials of each dataset.

Table 5

	ADULT Train	COV_TYPE Train	LETTER Train
KNN	0.9426666666666667	1.0	1.0
Random Forest	1.0	1.0	1.0
Logistic Regression	0.8498666666666667	0.7543333333333333	0.7258666666666667

To compare the model's performance with each other, T-test was also performed on the test accuracy scores generated by the three algorithms on each dataset. Table 6 reports the p-values for each of the combination of the algorithms on each dataset.

Table 6

	ADULT p-value	COV_TYPE p-value	LETTER p-value
KNN and Random Forest	0.00048468528084998807	2.692397259865767e-06	0.003838995866673888
Random Forest and Logistic Regression	0.2434617939743712	3.702247880570312e-06	3.6725201869652128e-09
Logistic Regression and KNN	0.00037013662334016575	0.0010668985208018861	5.618046883959578e-07

Again, from the accuracy scores of each of the algorithms show us that random forest was overall the best model on all three datasets, while KNN came as a close second, leaving Logistic Regression to have the poorest performance.

For secondary results, this paper reports raw test set accuracy scores across all trials on all of the datasets. Table 7 shows the raw test scores for all three algorithms for each dataset.

Table 7

	ADULT Test	COV_TYPE Test	LETTER Test
KNN	0.8237024083936094	0.7697964625736964	0.9283333333333333
	0.8230267864239726	0.7713051116990618	0.9318666666666666
	0.8185756299181305	0.7741297056311327	0.9202
Random Forest	0.8501311501470471	0.8223422428699402	0.9492
	0.8433351879818775	0.8233127087630119	0.9485333333333333
	0.8471902074556872	0.8221842600501379	0.9466
Logistic Regression	0.8424608536682299	0.7504895731338931	0.7258666666666667
	0.8433351879818775	0.7497447969833961	0.7236666666666667
	0.846395358079644	0.755926959855003	0.7233333333333334

Conclusion:

The importance of studying the performance of different supervised learning analysis algorithms in today's world is important. Machine Learning is proving to be a big part of our future and as such exploring and learning which algorithms work best on what kind of datasets is very useful. The experiment reported in this paper analyzed the performance of three different learning algorithms, namely K-Nearest Neighbors, Random Forest, and Logistic Regression, on three datasets based on the 'accuracy' performance metric. Overall, the best performance was shown

by Random Forest, beating the performance of KNN narrowly. The poorest performance was observed from Logistic Regression in our analysis.

References / Acknowledgements:

This paper was inspired by Rich Caruana and Alexandru Niculescu-Mizil's paper: "An Empirical Comparison of Supervised Learning Algorithms."

I thank Professor Jason Fleischer from University of California San Diego for his guidance and help with a lot of the code taught through his model selection lecture during his COGS 118A class in the Fall of 2020.

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn import datasets
from sklearn.metrics import accuracy_score
from pandas import read_csv
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [2]:
```
headers = ['age',
           'wrk-cls',
           'fnlwgt',
           'edu-lvl',
           'edu-num',
           'marriage',
           'occupation',
           'relationship',
           'race',
           'sex',
           'cap-gain',
           'cap-loss',
           'hr-per-wk',
           'country',
           'income']

df = pd.read_csv('adult.data', header = None, names = headers, na_values = ' ?')
```

In [3]:
```
df = df.dropna()
```

In [4]:
```
# Replacing the labels with 0's and 1's
numeric = df['income'].tolist()
for i, j in enumerate(numeric):
    if j == ' >50K':
        numeric[i] = 1
    else:
        numeric[i] = 0

df['income'] = numeric
```

In [5]:
```
# Seperating features and labels
X_p = pd.get_dummies(df.iloc[:, :-1])
y_p = df.iloc[:, -1]
```

In [6]:
```
# accuracy vectors for train set
logreg_accuracy_train = []
randforest_accuracy_train = []
knn_accuracy_train = []
```

In [7]:
```
# accuracy vectors for test set
logreg_accuracy = []
randforest_accuracy = []
knn_accuracy = []
```

Trial 1

In [8]:
```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedKFold

# take all our penguin data, and reserve 20% of it for testing
X_train, X_test, y_train, y_test = train_test_split(X_p, y_p,
                                                    train_size=5000,
                                                    random_state=12345,
                                                    stratify=y_p)


# Initializing Classifiers
clf1 = KNeighborsClassifier()
clf2 = RandomForestClassifier(n_estimators = 1024)
clf3 = LogisticRegression(solver='liblinear')

# Declaring parameters
K_list = np.array([n*20 for n in range(1,26)])
F_list = np.array([1, 2, 4, 6, 8, 12, 16, 20])
C_list = np.array([10**(-8), 10**(-7), 10**(-6), 10**(-5), 10**(-4), 10**(-3), 10**(-2), 10**(-1),
                   10**(0), 10**(1), 10**(2), 10**(3), 10**(4)])

# Building the pipelines
pipe1 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf1)])

pipe2 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf2)])

pipe3 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf3)])


# Setting up the parameter grids
param_grid1 = [{'classifier__weights': ['uniform', 'distance'],
                'classifier__n_neighbors': K_list}]

param_grid2 = [{'classifier__max_features': F_list}]

param_grid3 = [{'classifier__C': C_list}]
```

```
# Setting up multiple GridSearchCV objects, 1 for each algorithm
gridcvs = {}

for pgrid, est, name in zip((param_grid1, param_grid2, param_grid3),
                            (pipe1, pipe2, pipe3),
                            ('KNN', 'RandomForest', 'Logistic')):
    gcv = GridSearchCV(estimator=est,
                       param_grid=pgrid,
                       scoring='accuracy',
                       n_jobs=3,
                       cv=5, # just 2-fold inner loop, i.e. train/test
                       verbose=0,
                       return_train_score=True,
                       refit='accuracy')
    gridcvs[name] = gcv
```

In [9]:
```
%%time
# ^^ this handy Jupyter magic times the execution of the cell for you

cv_scores = {name: [] for name, gs_est in gridcvs.items()}

skfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

# The outer loop for algorithm selection
c = 1
for outer_train_idx, outer_valid_idx in skfold.split(X_train,y_train):
    for name, gs_est in sorted(gridcvs.items()):
        print('outer fold %d/5 | tuning %-8s' % (c, name), end='')

        # The inner loop for hyperparameter tuning
        gs_est.fit(X_train.iloc[outer_train_idx], y_train.iloc[outer_train_idx])
        y_pred = gs_est.predict(X_train.iloc[outer_valid_idx])
        acc = accuracy_score(y_true=y_train.iloc[outer_valid_idx], y_pred=y_pred)
        print(' | inner ACC %.2f%% | outer ACC %.2f%%' %
              (gs_est.best_score_ * 100, acc * 100))
        cv_scores[name].append(acc)

    c += 1
```

```
outer fold 1/5 | tuning KNN      | inner ACC 82.80% | outer ACC 82.90%
outer fold 1/5 | tuning Logistic | inner ACC 84.08% | outer ACC 84.80%
outer fold 1/5 | tuning RandomForest | inner ACC 85.17% | outer ACC 86.20%
outer fold 2/5 | tuning KNN      | inner ACC 82.83% | outer ACC 82.40%
outer fold 2/5 | tuning Logistic | inner ACC 84.47% | outer ACC 84.00%
outer fold 2/5 | tuning RandomForest | inner ACC 85.52% | outer ACC 84.60%
outer fold 3/5 | tuning KNN      | inner ACC 82.80% | outer ACC 82.50%
outer fold 3/5 | tuning Logistic | inner ACC 84.35% | outer ACC 84.10%
outer fold 3/5 | tuning RandomForest | inner ACC 85.60% | outer ACC 84.30%
outer fold 4/5 | tuning KNN      | inner ACC 83.05% | outer ACC 82.10%
outer fold 4/5 | tuning Logistic | inner ACC 84.10% | outer ACC 83.60%
outer fold 4/5 | tuning RandomForest | inner ACC 85.38% | outer ACC 85.50%
outer fold 5/5 | tuning KNN      | inner ACC 82.88% | outer ACC 83.40%
outer fold 5/5 | tuning Logistic | inner ACC 84.25% | outer ACC 84.30%
outer fold 5/5 | tuning RandomForest | inner ACC 85.28% | outer ACC 84.80%
CPU times: user 51.8 s, sys: 4.93 s, total: 56.8 s
Wall time: 1h 3min 10s
```

In [10]:
```
# Looking at the results
for name in cv_scores:
    print('%-8s | outer CV acc. %.2f%% +\- %.3f' % (
          name, 100 * np.mean(cv_scores[name]), 100 * np.std(cv_scores[name])))
print()
for name in cv_scores:
    print('{} best parameters'.format(name), gridcvs[name].best_params_)
```

```
KNN      | outer CV acc. 82.66% +\- 0.450
RandomForest | outer CV acc. 85.08% +\- 0.685
Logistic | outer CV acc. 84.16% +\- 0.393

KNN best parameters {'classifier__n_neighbors': 80, 'classifier__weights': 'distance'}
RandomForest best parameters {'classifier__max_features': 12}
Logistic best parameters {'classifier__C': 0.1}
```

In [11]:
```
# Fitting a model to the whole training set
# using the "best" KNN algorithm
best_algo = gridcvs['KNN']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('Accuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['KNN'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
knn_accuracy.append(test_acc)
knn_accuracy_train.append(train_acc)

# using the "best" RandomForest algorithm
best_algo = gridcvs['RandomForest']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['RandomForest'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
randforest_accuracy.append(test_acc)
randforest_accuracy_train.append(train_acc)

# using the "best" Logistic algorithm
best_algo = gridcvs['Logistic']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))


print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['Logistic'].best_params_)
```

```
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
logreg_accuracy.append(test_acc)
logreg_accuracy_train.append(train_acc)
```

```
Accuracy 83.10% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 80, 'classifier__weights': 'distance'}
Training Accuracy: 100.00%
Test Accuracy: 82.37%

Accuracy 85.76% (average over CV test folds)
Best Parameters: {'classifier__max_features': 16}
Training Accuracy: 100.00%
Test Accuracy: 85.01%

Accuracy 84.36% (average over CV test folds)
Best Parameters: {'classifier__C': 1.0}
Training Accuracy: 85.28%
Test Accuracy: 84.25%
```

In [12]:
```
print(knn_accuracy[0])
print(randforest_accuracy[0])
print(logreg_accuracy[0])
```

```
0.8237024083936094
0.8501311501470471
0.8424608536682299
```

Trial 2

In [13]:
```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedKFold

# take all our penguin data, and reserve 20% of it for testing
X_train, X_test, y_train, y_test = train_test_split(X_p, y_p,
                                                    train_size=5000,
                                                    random_state=54321,
                                                    stratify=y_p)


# Initializing Classifiers
clf1 = KNeighborsClassifier()
clf2 = RandomForestClassifier(n_estimators = 1024)
clf3 = LogisticRegression(solver='liblinear')

# Declaring parameters
K_list = np.array([n*20 for n in range(1,26)])
F_list = np.array([1, 2, 4, 6, 8, 12, 16, 20])
C_list = np.array([10**(-8), 10**(-7), 10**(-6), 10**(-5), 10**(-4), 10**(-3), 10**(-2), 10**(-1),
                   10**(0), 10**(1), 10**(2), 10**(3), 10**(4)])

# Building the pipelines
pipe1 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf1)])

pipe2 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf2)])

pipe3 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf3)])


# Setting up the parameter grids
param_grid1 = [{'classifier__weights': ['uniform', 'distance'],
                'classifier__n_neighbors': K_list}]

param_grid2 = [{'classifier__max_features': F_list}]

param_grid3 = [{'classifier__C': C_list}]
```

```
# Setting up multiple GridSearchCV objects, 1 for each algorithm
gridcvs = {}

for pgrid, est, name in zip((param_grid1, param_grid2, param_grid3),
                            (pipe1, pipe2, pipe3),
                            ('KNN', 'RandomForest', 'Logistic')):
    gcv = GridSearchCV(estimator=est,
                       param_grid=pgrid,
                       scoring='accuracy',
                       n_jobs=3,
                       cv=5, # just 2-fold inner loop, i.e. train/test
                       verbose=0,
                       return_train_score=True,
                       refit='accuracy')
    gridcvs[name] = gcv
```

In [14]:
```
%%time
# ^^ this handy Jupyter magic times the execution of the cell for you

cv_scores = {name: [] for name, gs_est in gridcvs.items()}

skfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

# The outer loop for algorithm selection
c = 1
for outer_train_idx, outer_valid_idx in skfold.split(X_train,y_train):
    for name, gs_est in sorted(gridcvs.items()):
        print('outer fold %d/5 | tuning %-8s' % (c, name), end='')

        # The inner loop for hyperparameter tuning
        gs_est.fit(X_train.iloc[outer_train_idx], y_train.iloc[outer_train_idx])
        y_pred = gs_est.predict(X_train.iloc[outer_valid_idx])
        acc = accuracy_score(y_true=y_train.iloc[outer_valid_idx], y_pred=y_pred)
        print(' | inner ACC %.2f%% | outer ACC %.2f%%' %
              (gs_est.best_score_ * 100, acc * 100))
        cv_scores[name].append(acc)

    c += 1
```

```
outer fold 1/5 | tuning KNN      | inner ACC 82.88% | outer ACC 83.10%
outer fold 1/5 | tuning Logistic | inner ACC 84.78% | outer ACC 84.40%
outer fold 1/5 | tuning RandomForest | inner ACC 85.35% | outer ACC 84.90%
outer fold 2/5 | tuning KNN      | inner ACC 83.28% | outer ACC 82.10%
outer fold 2/5 | tuning Logistic | inner ACC 84.40% | outer ACC 85.10%
outer fold 2/5 | tuning RandomForest | inner ACC 85.62% | outer ACC 84.80%
outer fold 3/5 | tuning KNN      | inner ACC 83.23% | outer ACC 82.70%
outer fold 3/5 | tuning Logistic | inner ACC 84.40% | outer ACC 85.20%
outer fold 3/5 | tuning RandomForest | inner ACC 84.95% | outer ACC 85.90%
outer fold 4/5 | tuning KNN      | inner ACC 83.35% | outer ACC 82.60%
outer fold 4/5 | tuning Logistic | inner ACC 84.62% | outer ACC 83.70%
outer fold 4/5 | tuning RandomForest | inner ACC 85.75% | outer ACC 84.50%
outer fold 5/5 | tuning KNN      | inner ACC 83.12% | outer ACC 81.30%
outer fold 5/5 | tuning Logistic | inner ACC 84.55% | outer ACC 85.10%
outer fold 5/5 | tuning RandomForest | inner ACC 85.45% | outer ACC 85.20%
CPU times: user 49.1 s, sys: 4.63 s, total: 53.7 s
Wall time: 1h 2min 29s
```

In [15]:
```
# Looking at the results
for name in cv_scores:
    print('%-8s | outer CV acc. %.2f%% +\- %.3f' % (
          name, 100 * np.mean(cv_scores[name]), 100 * np.std(cv_scores[name])))
print()
for name in cv_scores:
    print('{} best parameters'.format(name), gridcvs[name].best_params_)
```

```
KNN      | outer CV acc. 82.36% +\- 0.618
RandomForest | outer CV acc. 85.06% +\- 0.476
Logistic | outer CV acc. 84.70% +\- 0.576

KNN best parameters {'classifier__n_neighbors': 20, 'classifier__weights': 'distance'}
RandomForest best parameters {'classifier__max_features': 20}
Logistic best parameters {'classifier__C': 0.1}
```

In [16]:
```
# Fitting a model to the whole training set
# using the "best" KNN algorithm
best_algo = gridcvs['KNN']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('Accuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['KNN'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
knn_accuracy.append(test_acc)
knn_accuracy_train.append(train_acc)

# using the "best" RandomForest algorithm
best_algo = gridcvs['RandomForest']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['RandomForest'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
randforest_accuracy.append(test_acc)
randforest_accuracy_train.append(train_acc)

# using the "best" Logistic algorithm
best_algo = gridcvs['Logistic']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))


print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['Logistic'].best_params_)
```

```
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
logreg_accuracy.append(test_acc)
logreg_accuracy_train.append(train_acc)
```

```
Accuracy 83.06% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 60, 'classifier__weights': 'distance'}
Training Accuracy: 100.00%
Test Accuracy: 82.30%

Accuracy 85.64% (average over CV test folds)
Best Parameters: {'classifier__max_features': 20}
Training Accuracy: 100.00%
Test Accuracy: 84.38%

Accuracy 84.78% (average over CV test folds)
Best Parameters: {'classifier__C': 0.1}
Training Accuracy: 85.48%
Test Accuracy: 84.33%
```

In [17]:
```
print(knn_accuracy[1])
print(randforest_accuracy[1])
print(logreg_accuracy[1])
```

```
0.8230267864239726
0.8438120976075034
0.8433351879818775
```

Trial 3

In [18]:
```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedKFold

# take all our penguin data, and reserve 20% of it for testing
X_train, X_test, y_train, y_test = train_test_split(X_p, y_p,
                                                    train_size=5000,
                                                    random_state=13245,
                                                    stratify=y_p)


# Initializing Classifiers
clf1 = KNeighborsClassifier()
clf2 = RandomForestClassifier(n_estimators = 1024)
clf3 = LogisticRegression(solver='liblinear')

# Declaring parameters
K_list = np.array([n*20 for n in range(1,26)])
F_list = np.array([1, 2, 4, 6, 8, 12, 16, 20])
C_list = np.array([10**(-8), 10**(-7), 10**(-6), 10**(-5), 10**(-4), 10**(-3), 10**(-2), 10**(-1),
                   10**(0), 10**(1), 10**(2), 10**(3), 10**(4)])

# Building the pipelines
pipe1 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf1)])

pipe2 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf2)])

pipe3 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf3)])


# Setting up the parameter grids
param_grid1 = [{'classifier__weights': ['uniform', 'distance'],
                'classifier__n_neighbors': K_list}]

param_grid2 = [{'classifier__max_features': F_list}]

param_grid3 = [{'classifier__C': C_list}]
```

```
# Setting up multiple GridSearchCV objects, 1 for each algorithm
gridcvs = {}

for pgrid, est, name in zip((param_grid1, param_grid2, param_grid3),
                            (pipe1, pipe2, pipe3),
                            ('KNN', 'RandomForest', 'Logistic')):
    gcv = GridSearchCV(estimator=est,
                       param_grid=pgrid,
                       scoring='accuracy',
                       n_jobs=3,
                       cv=5, # just 2-fold inner loop, i.e. train/test
                       verbose=0,
                       return_train_score=True,
                       refit='accuracy')
    gridcvs[name] = gcv
```

In [19]:
```
%%time
# ^^ this handy Jupyter magic times the execution of the cell for you

cv_scores = {name: [] for name, gs_est in gridcvs.items()}

skfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

# The outer loop for algorithm selection
c = 1
for outer_train_idx, outer_valid_idx in skfold.split(X_train,y_train):
    for name, gs_est in sorted(gridcvs.items()):
        print('outer fold %d/5 | tuning %-8s' % (c, name), end='')

        # The inner loop for hyperparameter tuning
        gs_est.fit(X_train.iloc[outer_train_idx], y_train.iloc[outer_train_idx])
        y_pred = gs_est.predict(X_train.iloc[outer_valid_idx])
        acc = accuracy_score(y_true=y_train.iloc[outer_valid_idx], y_pred=y_pred)
        print(' | inner ACC %.2f%% | outer ACC %.2f%%' %
              (gs_est.best_score_ * 100, acc * 100))
        cv_scores[name].append(acc)

    c += 1
```

```
outer fold 1/5 | tuning KNN      | inner ACC 81.95% | outer ACC 80.80%
outer fold 1/5 | tuning Logistic | inner ACC 83.20% | outer ACC 84.20%
outer fold 1/5 | tuning RandomForest | inner ACC 84.20% | outer ACC 84.10%
outer fold 2/5 | tuning KNN      | inner ACC 81.73% | outer ACC 82.60%
outer fold 2/5 | tuning Logistic | inner ACC 83.10% | outer ACC 83.60%
outer fold 2/5 | tuning RandomForest | inner ACC 84.08% | outer ACC 84.30%
outer fold 3/5 | tuning KNN      | inner ACC 81.62% | outer ACC 83.40%
outer fold 3/5 | tuning Logistic | inner ACC 83.12% | outer ACC 84.60%
outer fold 3/5 | tuning RandomForest | inner ACC 83.45% | outer ACC 85.00%
outer fold 4/5 | tuning KNN      | inner ACC 81.95% | outer ACC 80.60%
outer fold 4/5 | tuning Logistic | inner ACC 83.47% | outer ACC 82.70%
outer fold 4/5 | tuning RandomForest | inner ACC 84.17% | outer ACC 84.10%
outer fold 5/5 | tuning KNN      | inner ACC 82.05% | outer ACC 81.60%
outer fold 5/5 | tuning Logistic | inner ACC 83.38% | outer ACC 82.80%
outer fold 5/5 | tuning RandomForest | inner ACC 83.90% | outer ACC 83.60%
CPU times: user 49.2 s, sys: 4.99 s, total: 54.1 s
Wall time: 1h 2min 40s
```

In [20]:
```
# Looking at the results
for name in cv_scores:
    print('%-8s | outer CV acc. %.2f%% +\- %.3f' % (
          name, 100 * np.mean(cv_scores[name]), 100 * np.std(cv_scores[name])))
print()
for name in cv_scores:
    print('{} best parameters'.format(name), gridcvs[name].best_params_)
```

```
KNN      | outer CV acc. 81.80% +\- 1.066
RandomForest | outer CV acc. 84.22% +\- 0.453
Logistic | outer CV acc. 83.58% +\- 0.749

KNN best parameters {'classifier__n_neighbors': 60, 'classifier__weights': 'distance'}
RandomForest best parameters {'classifier__max_features': 8}
Logistic best parameters {'classifier__C': 0.1}
```

In [21]:
```
# Fitting a model to the whole training set
# using the "best" KNN algorithm
best_algo = gridcvs['KNN']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('Accuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['KNN'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
knn_accuracy.append(test_acc)
knn_accuracy_train.append(train_acc)

# using the "best" RandomForest algorithm
best_algo = gridcvs['RandomForest']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['RandomForest'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
randforest_accuracy.append(test_acc)
randforest_accuracy_train.append(train_acc)

# using the "best" Logistic algorithm
best_algo = gridcvs['Logistic']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))


print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['Logistic'].best_params_)
```

```
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
logreg_accuracy.append(test_acc)
logreg_accuracy_train.append(train_acc)
```

```
Accuracy 82.06% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 40, 'classifier__weights': 'uniform'}
Training Accuracy: 82.80%
Test Accuracy: 81.86%

Accuracy 83.82% (average over CV test folds)
Best Parameters: {'classifier__max_features': 12}
Training Accuracy: 100.00%
Test Accuracy: 84.72%

Accuracy 83.38% (average over CV test folds)
Best Parameters: {'classifier__C': 1.0}
Training Accuracy: 84.20%
Test Accuracy: 84.64%
```

In [22]:
```
print(knn_accuracy[2])
print(randforest_accuracy[2])
print(logreg_accuracy[2])
```

```
0.8185756299181305
0.8471902074556872
0.846395358079644
```

In [23]:
```
#report average train accuracy per classifier (with best parameter)
print("Average KNN Test Accuracy: ", np.mean(knn_accuracy_train))
print("Average Random Forest Test Accuracy: ", np.mean(randforest_accuracy_train))
print("Average Logistic Regression Test Accuracy: ", np.mean(logreg_accuracy_train))
```

```
Average KNN Test Accuracy:  0.9426666666666667
Average Random Forest Test Accuracy:  1.0
Average Logistic Regression Test Accuracy:  0.8498666666666667
```

In [24]:
```
#report average test accuracy per classifier (with best parameter)
print("Average KNN Test Accuracy: ", np.mean(knn_accuracy))
print("Average Random Forest Test Accuracy: ", np.mean(randforest_accuracy))
print("Average Logistic Regression Test Accuracy: ", np.mean(logreg_accuracy))
```

```
Average KNN Test Accuracy:  0.8217682749119043
Average Random Forest Test Accuracy:  0.8470444850700792
Average Logistic Regression Test Accuracy:  0.8440637999099171
```

T - test for ADULT dataset

In [25]:
```
from scipy import stats
import numpy as np
```

In [26]:
```
# T-test between the different algorithms

knn_forest = stats.ttest_ind(knn_accuracy, randforest_accuracy)

forest_logistic = stats.ttest_ind(randforest_accuracy, logreg_accuracy)

logistic_knn = stats.ttest_ind(logreg_accuracy, knn_accuracy)
```

In [27]:
```
# Results of the T-tests

print('KNN and RandomForest: ', knn_forest)
print('\nRandomForest and Logistic Regression: ', forest_logistic)
print('\nLogistic Regression and KNN: ', logistic_knn)
```

```
KNN and RandomForest:  Ttest_indResult(statistic=-10.389223178576424, pvalue=0.00048468528084998807)

RandomForest and Logistic Regression:  Ttest_indResult(statistic=1.3668281006611518, pvalue=0.243461793974371
2)

Logistic Regression and KNN:  Ttest_indResult(statistic=11.135200471769094, pvalue=0.00037013662334016575)
```

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn import datasets
from sklearn.metrics import accuracy_score
from pandas import read_csv
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [2]:
```
headers = ['Elevation', 'Aspect', 'Slope',
       'Horizontal_Distance_To_Hydrology', 'Vertical_Distance_To_Hydrology',
       'Horizontal_Distance_To_Roadways', 'Hillshade_9am', 'Hillshade_Noon',
       'Hillshade_3pm', 'Horizontal_Distance_To_Fire_Points',
       'Wilderness_Area1', 'Wilderness_Area2', 'Wilderness_Area3',
       'Wilderness_Area4', 'Soil_Type1', 'Soil_Type2', 'Soil_Type3',
       'Soil_Type4', 'Soil_Type5', 'Soil_Type6', 'Soil_Type7', 'Soil_Type8',
       'Soil_Type9', 'Soil_Type10', 'Soil_Type11', 'Soil_Type12',
       'Soil_Type13', 'Soil_Type14', 'Soil_Type15', 'Soil_Type16',
       'Soil_Type17', 'Soil_Type18', 'Soil_Type19', 'Soil_Type20',
       'Soil_Type21', 'Soil_Type22', 'Soil_Type23', 'Soil_Type24',
       'Soil_Type25', 'Soil_Type26', 'Soil_Type27', 'Soil_Type28',
       'Soil_Type29', 'Soil_Type30', 'Soil_Type31', 'Soil_Type32',
       'Soil_Type33', 'Soil_Type34', 'Soil_Type35', 'Soil_Type36',
       'Soil_Type37', 'Soil_Type38', 'Soil_Type39', 'Soil_Type40',
       'Cover_Type']

df = pd.read_csv('covtype.data', header = None, names = headers, na_values = ' ?')
```

In [3]:
```
df = df.dropna()
```

In [4]:
```
# Finding the most common cover type
numeric = df['Cover_Type'].tolist()
def most_common(numeric):
    return max(set(numeric), key = numeric.count)

print(most_common(numeric))
```

```
2
```

In [5]:
```
# Replacing the labels with 0's and 1's
for i, j in enumerate(numeric):
    if j == 2:
        numeric[i] = 1
    else:
        numeric[i] = 0

df['Cover_Type'] = numeric
```

In [6]:
```
# Seperating features and labels
X_p = df.iloc[:, :-1]
y_p = df.iloc[:, -1]
```

In [7]:
```
# accuracy vectors for train set
logreg_accuracy_train = []
randforest_accuracy_train = []
knn_accuracy_train = []
```

In [8]:
```
# accuracy vectors for test set
logreg_accuracy = []
randforest_accuracy = []
knn_accuracy = []
```

Trial 1

In [9]:
```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedKFold

# take all our penguin data, and reserve 20% of it for testing
X_train, X_test, y_train, y_test = train_test_split(X_p, y_p,
                                                    train_size=5000,
                                                    random_state=12345,
                                                    stratify=y_p)


# Initializing Classifiers
clf1 = KNeighborsClassifier()
clf2 = RandomForestClassifier(n_estimators = 1024)
clf3 = LogisticRegression(solver='liblinear', multi_class='auto')

# Declaring parameters
K_list = np.array([n*20 for n in range(1,26)])
F_list = np.array([1, 2, 4, 6, 8, 12, 16, 20])
C_list = np.array([10**(-8), 10**(-7), 10**(-6), 10**(-5), 10**(-4), 10**(-3), 10**(-2), 10**(-1),
                   10**(0), 10**(1), 10**(2), 10**(3), 10**(4)])

# Building the pipelines
pipe1 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf1)])

pipe2 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf2)])

pipe3 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf3)])


# Setting up the parameter grids
param_grid1 = [{'classifier__weights': ['uniform', 'distance'],
                'classifier__n_neighbors': K_list}]

param_grid2 = [{'classifier__max_features': F_list}]

param_grid3 = [{'classifier__C': C_list}]
```

```
# Setting up multiple GridSearchCV objects, 1 for each algorithm
gridcvs = {}

for pgrid, est, name in zip((param_grid1, param_grid2, param_grid3),
                            (pipe1, pipe2, pipe3),
                            ('KNN', 'RandomForest', 'Logistic')):
    gcv = GridSearchCV(estimator=est,
                       param_grid=pgrid,
                       scoring='accuracy',
                       n_jobs=3,
                       cv=5, # just 2-fold inner loop, i.e. train/test
                       verbose=0,
                       return_train_score=True,
                       refit='accuracy')
    gridcvs[name] = gcv
```

In [10]:
```
%%time
# ^^ this handy Jupyter magic times the execution of the cell for you

cv_scores = {name: [] for name, gs_est in gridcvs.items()}

skfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

# The outer loop for algorithm selection
c = 1
for outer_train_idx, outer_valid_idx in skfold.split(X_train,y_train):
    for name, gs_est in sorted(gridcvs.items()):
        print('outer fold %d/5 | tuning %-8s' % (c, name), end='')

        # The inner loop for hyperparameter tuning
        gs_est.fit(X_train.iloc[outer_train_idx], y_train.iloc[outer_train_idx])
        y_pred = gs_est.predict(X_train.iloc[outer_valid_idx])
        acc = accuracy_score(y_true=y_train.iloc[outer_valid_idx], y_pred=y_pred)
        print(' | inner ACC %.2f%% | outer ACC %.2f%%' %
              (gs_est.best_score_ * 100, acc * 100))
        cv_scores[name].append(acc)

    c += 1
```

```
outer fold 1/5 | tuning KNN      | inner ACC 76.47% | outer ACC 78.62%
outer fold 1/5 | tuning Logistic | inner ACC 74.97% | outer ACC 73.73%
outer fold 1/5 | tuning RandomForest | inner ACC 81.70% | outer ACC 80.52%
outer fold 2/5 | tuning KNN      | inner ACC 75.24% | outer ACC 80.22%
outer fold 2/5 | tuning Logistic | inner ACC 73.84% | outer ACC 77.12%
outer fold 2/5 | tuning RandomForest | inner ACC 80.70% | outer ACC 83.52%
outer fold 3/5 | tuning KNN      | inner ACC 76.92% | outer ACC 76.10%
outer fold 3/5 | tuning Logistic | inner ACC 74.70% | outer ACC 74.60%
outer fold 3/5 | tuning RandomForest | inner ACC 81.45% | outer ACC 82.00%
outer fold 4/5 | tuning KNN      | inner ACC 77.41% | outer ACC 73.17%
outer fold 4/5 | tuning Logistic | inner ACC 74.83% | outer ACC 73.17%
outer fold 4/5 | tuning RandomForest | inner ACC 81.78% | outer ACC 79.28%
outer fold 5/5 | tuning KNN      | inner ACC 76.31% | outer ACC 76.28%
outer fold 5/5 | tuning Logistic | inner ACC 74.31% | outer ACC 74.77%
outer fold 5/5 | tuning RandomForest | inner ACC 81.08% | outer ACC 83.48%
CPU times: user 56.6 s, sys: 4.93 s, total: 1min 1s
Wall time: 49min 22s
```

In [11]:
```
# Looking at the results
for name in cv_scores:
    print('%-8s | outer CV acc. %.2f%% +\- %.3f' % (
          name, 100 * np.mean(cv_scores[name]), 100 * np.std(cv_scores[name])))
print()
for name in cv_scores:
    print('{} best parameters'.format(name), gridcvs[name].best_params_)
```

```
KNN      | outer CV acc. 76.88% +\- 2.404
RandomForest | outer CV acc. 81.76% +\- 1.662
Logistic | outer CV acc. 74.68% +\- 1.354

KNN best parameters {'classifier__n_neighbors': 20, 'classifier__weights': 'distance'}
RandomForest best parameters {'classifier__max_features': 4}
Logistic best parameters {'classifier__C': 10.0}
```

In [12]:
```
# Fitting a model to the whole training set
# using the "best" KNN algorithm
best_algo = gridcvs['KNN']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('Accuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['KNN'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
knn_accuracy.append(test_acc)
knn_accuracy_train.append(train_acc)

# using the "best" RandomForest algorithm
best_algo = gridcvs['RandomForest']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['RandomForest'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
randforest_accuracy.append(test_acc)
randforest_accuracy_train.append(train_acc)

# using the "best" Logistic algorithm
best_algo = gridcvs['Logistic']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))


print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['Logistic'].best_params_)
```

```
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
logreg_accuracy.append(test_acc)
logreg_accuracy_train.append(train_acc)
```

```
Accuracy 77.50% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 20, 'classifier__weights': 'distance'}
Training Accuracy: 100.00%
Test Accuracy: 76.98%

Accuracy 82.00% (average over CV test folds)
Best Parameters: {'classifier__max_features': 16}
Training Accuracy: 100.00%
Test Accuracy: 82.23%

Accuracy 74.36% (average over CV test folds)
Best Parameters: {'classifier__C': 0.1}
Training Accuracy: 75.02%
Test Accuracy: 75.05%
```

In [13]:
```
print(knn_accuracy[0])
print(randforest_accuracy[0])
print(logreg_accuracy[0])
```

```
0.7697964625736964
0.8223422428699402
0.7504895731338931
```

Trial 2

In [14]:
```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedKFold

# take all our penguin data, and reserve 20% of it for testing
X_train, X_test, y_train, y_test = train_test_split(X_p, y_p,
                                                    train_size=5000,
                                                    random_state=54321,
                                                    stratify=y_p)


# Initializing Classifiers
clf1 = KNeighborsClassifier()
clf2 = RandomForestClassifier(n_estimators = 1024)
clf3 = LogisticRegression(solver='liblinear', multi_class='auto')

# Declaring parameters
K_list = np.array([n*20 for n in range(1,26)])
F_list = np.array([1, 2, 4, 6, 8, 12, 16, 20])
C_list = np.array([10**(-8), 10**(-7), 10**(-6), 10**(-5), 10**(-4), 10**(-3), 10**(-2), 10**(-1),
                   10**(0), 10**(1), 10**(2), 10**(3), 10**(4)])

# Building the pipelines
pipe1 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf1)])

pipe2 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf2)])

pipe3 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf3)])


# Setting up the parameter grids
param_grid1 = [{'classifier__weights': ['uniform', 'distance'],
                'classifier__n_neighbors': K_list}]

param_grid2 = [{'classifier__max_features': F_list}]

param_grid3 = [{'classifier__C': C_list}]
```

```
# Setting up multiple GridSearchCV objects, 1 for each algorithm
gridcvs = {}

for pgrid, est, name in zip((param_grid1, param_grid2, param_grid3),
                            (pipe1, pipe2, pipe3),
                            ('KNN', 'RandomForest', 'Logistic')):
    gcv = GridSearchCV(estimator=est,
                       param_grid=pgrid,
                       scoring='accuracy',
                       n_jobs=3,
                       cv=5, # just 2-fold inner loop, i.e. train/test
                       verbose=0,
                       return_train_score=True,
                       refit='accuracy')
    gridcvs[name] = gcv
```

In [15]:
```
%%time
# ^^ this handy Jupyter magic times the execution of the cell for you

cv_scores = {name: [] for name, gs_est in gridcvs.items()}

skfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

# The outer loop for algorithm selection
c = 1
for outer_train_idx, outer_valid_idx in skfold.split(X_train,y_train):
    for name, gs_est in sorted(gridcvs.items()):
        print('outer fold %d/5 | tuning %-8s' % (c, name), end='')

        # The inner loop for hyperparameter tuning
        gs_est.fit(X_train.iloc[outer_train_idx], y_train.iloc[outer_train_idx])
        y_pred = gs_est.predict(X_train.iloc[outer_valid_idx])
        acc = accuracy_score(y_true=y_train.iloc[outer_valid_idx], y_pred=y_pred)
        print(' | inner ACC %.2f%% | outer ACC %.2f%%' %
              (gs_est.best_score_ * 100, acc * 100))
        cv_scores[name].append(acc)

    c += 1
```

```
outer fold 1/5 | tuning KNN      | inner ACC 76.24% | outer ACC 74.13%
outer fold 1/5 | tuning Logistic | inner ACC 75.22% | outer ACC 74.13%
outer fold 1/5 | tuning RandomForest | inner ACC 81.87% | outer ACC 82.32%
outer fold 2/5 | tuning KNN      | inner ACC 76.09% | outer ACC 76.22%
outer fold 2/5 | tuning Logistic | inner ACC 75.42% | outer ACC 75.62%
outer fold 2/5 | tuning RandomForest | inner ACC 81.52% | outer ACC 82.72%
outer fold 3/5 | tuning KNN      | inner ACC 75.33% | outer ACC 77.80%
outer fold 3/5 | tuning Logistic | inner ACC 75.38% | outer ACC 76.30%
outer fold 3/5 | tuning RandomForest | inner ACC 81.00% | outer ACC 82.20%
outer fold 4/5 | tuning KNN      | inner ACC 76.58% | outer ACC 74.87%
outer fold 4/5 | tuning Logistic | inner ACC 76.11% | outer ACC 74.07%
outer fold 4/5 | tuning RandomForest | inner ACC 81.95% | outer ACC 80.38%
outer fold 5/5 | tuning KNN      | inner ACC 75.61% | outer ACC 78.68%
outer fold 5/5 | tuning Logistic | inner ACC 74.86% | outer ACC 76.28%
outer fold 5/5 | tuning RandomForest | inner ACC 81.05% | outer ACC 83.28%
CPU times: user 1min 14s, sys: 4.44 s, total: 1min 18s
Wall time: 48min 39s
```

In [16]:
```
# Looking at the results
for name in cv_scores:
    print('%-8s | outer CV acc. %.2f%% +\- %.3f' % (
          name, 100 * np.mean(cv_scores[name]), 100 * np.std(cv_scores[name])))
print()
for name in cv_scores:
    print('{} best parameters'.format(name), gridcvs[name].best_params_)
```

```
KNN      | outer CV acc. 76.34% +\- 1.713
RandomForest | outer CV acc. 82.18% +\- 0.976
Logistic | outer CV acc. 75.28% +\- 0.994

KNN best parameters {'classifier__n_neighbors': 20, 'classifier__weights': 'distance'}
RandomForest best parameters {'classifier__max_features': 16}
Logistic best parameters {'classifier__C': 1.0}
```

In [17]:
```
# Fitting a model to the whole training set
# using the "best" KNN algorithm
best_algo = gridcvs['KNN']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('Accuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['KNN'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
knn_accuracy.append(test_acc)
knn_accuracy_train.append(train_acc)

# using the "best" RandomForest algorithm
best_algo = gridcvs['RandomForest']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['RandomForest'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
randforest_accuracy.append(test_acc)
randforest_accuracy_train.append(train_acc)

# using the "best" Logistic algorithm
best_algo = gridcvs['Logistic']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))


print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['Logistic'].best_params_)
```

```
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
logreg_accuracy.append(test_acc)
logreg_accuracy_train.append(train_acc)
```

```
Accuracy 76.16% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 20, 'classifier__weights': 'distance'}
Training Accuracy: 100.00%
Test Accuracy: 77.13%

Accuracy 82.14% (average over CV test folds)
Best Parameters: {'classifier__max_features': 12}
Training Accuracy: 100.00%
Test Accuracy: 82.33%

Accuracy 75.36% (average over CV test folds)
Best Parameters: {'classifier__C': 10.0}
Training Accuracy: 76.14%
Test Accuracy: 74.97%
```

In [18]:
```
print(knn_accuracy[1])
print(randforest_accuracy[1])
print(logreg_accuracy[1])
```

```
0.7713051116990618
0.8233127087630119
0.7497447969833961
```

Trial 3

In [19]:
```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedKFold

# take all our penguin data, and reserve 20% of it for testing
X_train, X_test, y_train, y_test = train_test_split(X_p, y_p,
                                                    train_size=5000,
                                                    random_state=13245,
                                                    stratify=y_p)


# Initializing Classifiers
clf1 = KNeighborsClassifier()
clf2 = RandomForestClassifier(n_estimators = 1024)
clf3 = LogisticRegression(solver='liblinear', multi_class='auto')

# Declaring parameters
K_list = np.array([n*20 for n in range(1,26)])
F_list = np.array([1, 2, 4, 6, 8, 12, 16, 20])
C_list = np.array([10**(-8), 10**(-7), 10**(-6), 10**(-5), 10**(-4), 10**(-3), 10**(-2), 10**(-1),
                   10**(0), 10**(1), 10**(2), 10**(3), 10**(4)])

# Building the pipelines
pipe1 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf1)])

pipe2 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf2)])

pipe3 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf3)])


# Setting up the parameter grids
param_grid1 = [{'classifier__weights': ['uniform', 'distance'],
                'classifier__n_neighbors': K_list}]

param_grid2 = [{'classifier__max_features': F_list}]

param_grid3 = [{'classifier__C': C_list}]
```

```
# Setting up multiple GridSearchCV objects, 1 for each algorithm
gridcvs = {}

for pgrid, est, name in zip((param_grid1, param_grid2, param_grid3),
                            (pipe1, pipe2, pipe3),
                            ('KNN', 'RandomForest', 'Logistic')):
    gcv = GridSearchCV(estimator=est,
                       param_grid=pgrid,
                       scoring='accuracy',
                       n_jobs=3,
                       cv=5, # just 2-fold inner loop, i.e. train/test
                       verbose=0,
                       return_train_score=True,
                       refit='accuracy')
    gridcvs[name] = gcv
```

In [20]:
```
%%time
# ^^ this handy Jupyter magic times the execution of the cell for you

cv_scores = {name: [] for name, gs_est in gridcvs.items()}

skfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

# The outer loop for algorithm selection
c = 1
for outer_train_idx, outer_valid_idx in skfold.split(X_train,y_train):
    for name, gs_est in sorted(gridcvs.items()):
        print('outer fold %d/5 | tuning %-8s' % (c, name), end='')

        # The inner loop for hyperparameter tuning
        gs_est.fit(X_train.iloc[outer_train_idx], y_train.iloc[outer_train_idx])
        y_pred = gs_est.predict(X_train.iloc[outer_valid_idx])
        acc = accuracy_score(y_true=y_train.iloc[outer_valid_idx], y_pred=y_pred)
        print(' | inner ACC %.2f%% | outer ACC %.2f%%' %
              (gs_est.best_score_ * 100, acc * 100))
        cv_scores[name].append(acc)

    c += 1
```

```
outer fold 1/5 | tuning KNN      | inner ACC 75.29% | outer ACC 74.13%
outer fold 1/5 | tuning Logistic | inner ACC 75.29% | outer ACC 71.73%
outer fold 1/5 | tuning RandomForest | inner ACC 80.40% | outer ACC 78.32%
outer fold 2/5 | tuning KNN      | inner ACC 75.94% | outer ACC 76.02%
outer fold 2/5 | tuning Logistic | inner ACC 74.37% | outer ACC 75.82%
outer fold 2/5 | tuning RandomForest | inner ACC 80.77% | outer ACC 81.42%
outer fold 3/5 | tuning KNN      | inner ACC 75.52% | outer ACC 75.20%
outer fold 3/5 | tuning Logistic | inner ACC 74.65% | outer ACC 72.20%
outer fold 3/5 | tuning RandomForest | inner ACC 80.95% | outer ACC 79.60%
outer fold 4/5 | tuning KNN      | inner ACC 75.28% | outer ACC 76.58%
outer fold 4/5 | tuning Logistic | inner ACC 74.53% | outer ACC 75.38%
outer fold 4/5 | tuning RandomForest | inner ACC 80.70% | outer ACC 81.08%
outer fold 5/5 | tuning KNN      | inner ACC 74.78% | outer ACC 77.78%
outer fold 5/5 | tuning Logistic | inner ACC 73.68% | outer ACC 75.68%
outer fold 5/5 | tuning RandomForest | inner ACC 79.98% | outer ACC 83.38%
CPU times: user 50.5 s, sys: 5.07 s, total: 55.6 s
Wall time: 46min 8s
```

In [21]:
```
# Looking at the results
for name in cv_scores:
    print('%-8s | outer CV acc. %.2f%% +\- %.3f' % (
          name, 100 * np.mean(cv_scores[name]), 100 * np.std(cv_scores[name])))
print()
for name in cv_scores:
    print('{} best parameters'.format(name), gridcvs[name].best_params_)
```

```
KNN      | outer CV acc. 75.94% +\- 1.235
RandomForest | outer CV acc. 80.76% +\- 1.715
Logistic | outer CV acc. 74.16% +\- 1.805

KNN best parameters {'classifier__n_neighbors': 20, 'classifier__weights': 'distance'}
RandomForest best parameters {'classifier__max_features': 4}
Logistic best parameters {'classifier__C': 0.1}
```

In [22]:
```
# Fitting a model to the whole training set
# using the "best" KNN algorithm
best_algo = gridcvs['KNN']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('Accuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['KNN'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
knn_accuracy.append(test_acc)
knn_accuracy_train.append(train_acc)

# using the "best" RandomForest algorithm
best_algo = gridcvs['RandomForest']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['RandomForest'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
randforest_accuracy.append(test_acc)
randforest_accuracy_train.append(train_acc)

# using the "best" Logistic algorithm
best_algo = gridcvs['Logistic']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))


print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['Logistic'].best_params_)
```

```
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
logreg_accuracy.append(test_acc)
logreg_accuracy_train.append(train_acc)
```

```
Accuracy 76.00% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 20, 'classifier__weights': 'distance'}
Training Accuracy: 100.00%
Test Accuracy: 77.41%

Accuracy 81.34% (average over CV test folds)
Best Parameters: {'classifier__max_features': 6}
Training Accuracy: 100.00%
Test Accuracy: 82.22%

Accuracy 74.64% (average over CV test folds)
Best Parameters: {'classifier__C': 100.0}
Training Accuracy: 75.14%
Test Accuracy: 75.59%
```

In [23]:
```
print(knn_accuracy[2])
print(randforest_accuracy[2])
print(logreg_accuracy[2])
```

```
0.7741297056311327
0.8221842600501379
0.755926959855003
```

In [24]:
```
#report average train accuracy per classifier (with best parameter)
print("Average KNN Test Accuracy: ", np.mean(knn_accuracy_train))
print("Average Random Forest Test Accuracy: ", np.mean(randforest_accuracy_train))
print("Average Logistic Regression Test Accuracy: ", np.mean(logreg_accuracy_train))
```

```
Average KNN Test Accuracy:  1.0
Average Random Forest Test Accuracy:  1.0
Average Logistic Regression Test Accuracy:  0.7543333333333333
```

In [25]:
```
#report average test accuracy per classifier (with best parameter)
print("Average KNN Test Accuracy: ", np.mean(knn_accuracy))
print("Average Random Forest Test Accuracy: ", np.mean(randforest_accuracy))
print("Average Logistic Regression Test Accuracy: ", np.mean(logreg_accuracy))
```

```
Average KNN Test Accuracy:  0.7717437599679636
Average Random Forest Test Accuracy:  0.82261307056103
Average Logistic Regression Test Accuracy:  0.7520537766574308
```

T - test for COV_TYPE dataset

In [26]:
```
from scipy import stats
import numpy as np
```

In [27]:
```
# T-test between the different algorithms

knn_forest = stats.ttest_ind(knn_accuracy, randforest_accuracy)

forest_logistic = stats.ttest_ind(randforest_accuracy, logreg_accuracy)

logistic_knn = stats.ttest_ind(logreg_accuracy, knn_accuracy)
```

In [28]:
```
# Results of the T-tests

print('KNN and RandomForest: ', knn_forest)
print('\nRandomForest and Logistic Regression: ', forest_logistic)
print('\nLogistic Regression and KNN: ', logistic_knn)
```

```
KNN and RandomForest:  Ttest_indResult(statistic=-38.59381360746943, pvalue=2.692397259865767e-06)

RandomForest and Logistic Regression:  Ttest_indResult(statistic=35.63298684574943, pvalue=3.702247880570312e
-06)

Logistic Regression and KNN:  Ttest_indResult(statistic=-8.465804406727974, pvalue=0.0010668985208018861)
```

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn import datasets
from sklearn.metrics import accuracy_score
from pandas import read_csv
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [2]:
```
headers = ['lettr', 'x-box', 'y-box', 'width', 'high', 'onpix', 'x-bar', 'y-bar', 'x2bar', 'y2bar', 'xybar',
'x2ybr', 'xy2br',
           'x-ege', 'xegvy', 'y-ege', 'yegvx']

df = pd.read_csv('letter-recognition.data', header = None, names = headers, na_values = ' ?')
```

In [3]:
```
df = df.dropna()
```

In [4]:
```
# Replacing the labels with 0's and 1's
for_numeric = ['A','B','C','D','E','F','G','H','I','J','K','L','M']

numeric = df['lettr'].tolist()

for i, j in enumerate(numeric):
    if j in for_numeric :
        numeric[i] = 1
    else:
        numeric[i] = 0

df['lettr'] = numeric
```

In [5]:
```
# Seperating features and labels
X_p = df.iloc[:, 1:]
y_p = df.iloc[:, 0]
```

In [6]:
```
# accuracy vectors for train set
logreg_accuracy_train = []
randforest_accuracy_train = []
knn_accuracy_train = []
```

In [7]:
```
# accuracy vectors for test set
logreg_accuracy = []
randforest_accuracy = []
knn_accuracy = []
```

In [8]:
```
%%time
import warnings
warnings.filterwarnings('ignore')
```

```
CPU times: user 21 µs, sys: 9 µs, total: 30 µs
Wall time: 35.3 µs
```

Trial 1

In [9]:
```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedKFold

# take all our penguin data, and reserve 20% of it for testing
X_train, X_test, y_train, y_test = train_test_split(X_p, y_p,
                                                    train_size=5000,
                                                    random_state=12345,
                                                    stratify=y_p)


# Initializing Classifiers
clf1 = KNeighborsClassifier()
clf2 = RandomForestClassifier(n_estimators = 1024)
clf3 = LogisticRegression(solver='liblinear', multi_class='auto')

# Declaring parameters
K_list = np.array([n*20 for n in range(1,26)])
F_list = np.array([1, 2, 4, 6, 8, 12, 16])
C_list = np.array([10**(-8), 10**(-7), 10**(-6), 10**(-5), 10**(-4), 10**(-3), 10**(-2), 10**(-1),
                   10**(0), 10**(1), 10**(2), 10**(3), 10**(4)])

# Building the pipelines
pipe1 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf1)])

pipe2 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf2)])

pipe3 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf3)])


# Setting up the parameter grids
param_grid1 = [{'classifier__weights': ['uniform', 'distance'],
                'classifier__n_neighbors': K_list}]

param_grid2 = [{'classifier__max_features': F_list}]

param_grid3 = [{'classifier__C': C_list}]
```

```
# Setting up multiple GridSearchCV objects, 1 for each algorithm
gridcvs = {}

for pgrid, est, name in zip((param_grid1, param_grid2, param_grid3),
                            (pipe1, pipe2, pipe3),
                            ('KNN', 'RandomForest', 'Logistic')):
    gcv = GridSearchCV(estimator=est,
                       param_grid=pgrid,
                       scoring='accuracy',
                       n_jobs=3,
                       cv=5, # just 2-fold inner loop, i.e. train/test
                       verbose=0,
                       return_train_score=True,
                       refit='accuracy')
    gridcvs[name] = gcv
```

In [10]:
```
%%time
# ^^ this handy Jupyter magic times the execution of the cell for you

cv_scores = {name: [] for name, gs_est in gridcvs.items()}

skfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

# The outer loop for algorithm selection
c = 1
for outer_train_idx, outer_valid_idx in skfold.split(X_train,y_train):
    for name, gs_est in sorted(gridcvs.items()):
        print('outer fold %d/5 | tuning %-8s' % (c, name), end='')

        # The inner loop for hyperparameter tuning
        gs_est.fit(X_train.iloc[outer_train_idx], y_train.iloc[outer_train_idx])
        y_pred = gs_est.predict(X_train.iloc[outer_valid_idx])
        acc = accuracy_score(y_true=y_train.iloc[outer_valid_idx], y_pred=y_pred)
        print(' | inner ACC %.2f%% | outer ACC %.2f%%' %
              (gs_est.best_score_ * 100, acc * 100))
        cv_scores[name].append(acc)

    c += 1
```

```
outer fold 1/5 | tuning KNN      | inner ACC 89.90% | outer ACC 91.00%
outer fold 1/5 | tuning Logistic | inner ACC 72.72% | outer ACC 69.70%
outer fold 1/5 | tuning RandomForest | inner ACC 92.60% | outer ACC 93.50%
outer fold 2/5 | tuning KNN      | inner ACC 90.18% | outer ACC 89.10%
outer fold 2/5 | tuning Logistic | inner ACC 72.40% | outer ACC 72.90%
outer fold 2/5 | tuning RandomForest | inner ACC 92.97% | outer ACC 92.20%
outer fold 3/5 | tuning KNN      | inner ACC 89.83% | outer ACC 91.20%
outer fold 3/5 | tuning Logistic | inner ACC 72.20% | outer ACC 74.20%
outer fold 3/5 | tuning RandomForest | inner ACC 92.97% | outer ACC 94.60%
outer fold 4/5 | tuning KNN      | inner ACC 89.03% | outer ACC 91.90%
outer fold 4/5 | tuning Logistic | inner ACC 72.17% | outer ACC 74.50%
outer fold 4/5 | tuning RandomForest | inner ACC 92.58% | outer ACC 93.90%
outer fold 5/5 | tuning KNN      | inner ACC 89.53% | outer ACC 89.00%
outer fold 5/5 | tuning Logistic | inner ACC 72.75% | outer ACC 71.20%
outer fold 5/5 | tuning RandomForest | inner ACC 93.23% | outer ACC 92.90%
CPU times: user 43.8 s, sys: 4.15 s, total: 47.9 s
Wall time: 27min 49s
```

In [11]:
```
# Looking at the results
for name in cv_scores:
    print('%-8s | outer CV acc. %.2f%% +\- %.3f' % (
          name, 100 * np.mean(cv_scores[name]), 100 * np.std(cv_scores[name])))
print()
for name in cv_scores:
    print('{} best parameters'.format(name), gridcvs[name].best_params_)
```

```
KNN      | outer CV acc. 90.44% +\- 1.174
RandomForest | outer CV acc. 93.42% +\- 0.823
Logistic | outer CV acc. 72.50% +\- 1.821

KNN best parameters {'classifier__n_neighbors': 20, 'classifier__weights': 'distance'}
RandomForest best parameters {'classifier__max_features': 4}
Logistic best parameters {'classifier__C': 1.0}
```

In [12]:
```
# Fitting a model to the whole training set
# using the "best" KNN algorithm
best_algo = gridcvs['KNN']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('Accuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['KNN'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
knn_accuracy.append(test_acc)
knn_accuracy_train.append(train_acc)

# using the "best" RandomForest algorithm
best_algo = gridcvs['RandomForest']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['RandomForest'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
randforest_accuracy.append(test_acc)
randforest_accuracy_train.append(train_acc)

# using the "best" Logistic algorithm
best_algo = gridcvs['Logistic']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))


print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['Logistic'].best_params_)
```

```
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
logreg_accuracy.append(test_acc)
logreg_accuracy_train.append(train_acc)
```

```
Accuracy 91.02% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 20, 'classifier__weights': 'distance'}
Training Accuracy: 100.00%
Test Accuracy: 92.83%

Accuracy 93.74% (average over CV test folds)
Best Parameters: {'classifier__max_features': 8}
Training Accuracy: 100.00%
Test Accuracy: 94.92%

Accuracy 72.48% (average over CV test folds)
Best Parameters: {'classifier__C': 0.1}
Training Accuracy: 72.30%
Test Accuracy: 72.59%
```

In [13]:
```
print(knn_accuracy[0])
print(randforest_accuracy[0])
print(logreg_accuracy[0])
```

```
0.9283333333333333
0.9492
0.7258666666666667
```

Trial 2

In [14]:
```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedKFold

# take all our penguin data, and reserve 20% of it for testing
X_train, X_test, y_train, y_test = train_test_split(X_p, y_p,
                                                    train_size=5000,
                                                    random_state=5432,
                                                    stratify=y_p)


# Initializing Classifiers
clf1 = KNeighborsClassifier()
clf2 = RandomForestClassifier(n_estimators = 1024)
clf3 = LogisticRegression(solver='liblinear', multi_class='auto')

# Declaring parameters
K_list = np.array([n*20 for n in range(1,26)])
F_list = np.array([1, 2, 4, 6, 8, 12, 16])
C_list = np.array([10**(-8), 10**(-7), 10**(-6), 10**(-5), 10**(-4), 10**(-3), 10**(-2), 10**(-1),
                   10**(0), 10**(1), 10**(2), 10**(3), 10**(4)])

# Building the pipelines
pipe1 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf1)])

pipe2 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf2)])

pipe3 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf3)])


# Setting up the parameter grids
param_grid1 = [{'classifier__weights': ['uniform', 'distance'],
                'classifier__n_neighbors': K_list}]

param_grid2 = [{'classifier__max_features': F_list}]

param_grid3 = [{'classifier__C': C_list}]
```

```
# Setting up multiple GridSearchCV objects, 1 for each algorithm
gridcvs = {}

for pgrid, est, name in zip((param_grid1, param_grid2, param_grid3),
                            (pipe1, pipe2, pipe3),
                            ('KNN', 'RandomForest', 'Logistic')):
    gcv = GridSearchCV(estimator=est,
                       param_grid=pgrid,
                       scoring='accuracy',
                       n_jobs=3,
                       cv=5, # just 2-fold inner loop, i.e. train/test
                       verbose=0,
                       return_train_score=True,
                       refit='accuracy')
    gridcvs[name] = gcv
```

In [15]:
```
%%time
# ^^ this handy Jupyter magic times the execution of the cell for you

cv_scores = {name: [] for name, gs_est in gridcvs.items()}

skfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

# The outer loop for algorithm selection
c = 1
for outer_train_idx, outer_valid_idx in skfold.split(X_train,y_train):
    for name, gs_est in sorted(gridcvs.items()):
        print('outer fold %d/5 | tuning %-8s' % (c, name), end='')

        # The inner loop for hyperparameter tuning
        gs_est.fit(X_train.iloc[outer_train_idx], y_train.iloc[outer_train_idx])
        y_pred = gs_est.predict(X_train.iloc[outer_valid_idx])
        acc = accuracy_score(y_true=y_train.iloc[outer_valid_idx], y_pred=y_pred)
        print(' | inner ACC %.2f%% | outer ACC %.2f%%' %
              (gs_est.best_score_ * 100, acc * 100))
        cv_scores[name].append(acc)

    c += 1
```

```
outer fold 1/5 | tuning KNN      | inner ACC 90.12% | outer ACC 92.90%
outer fold 1/5 | tuning Logistic | inner ACC 73.10% | outer ACC 72.30%
outer fold 1/5 | tuning RandomForest | inner ACC 93.40% | outer ACC 95.70%
outer fold 2/5 | tuning KNN      | inner ACC 90.00% | outer ACC 93.00%
outer fold 2/5 | tuning Logistic | inner ACC 72.52% | outer ACC 74.70%
outer fold 2/5 | tuning RandomForest | inner ACC 93.33% | outer ACC 93.90%
outer fold 3/5 | tuning KNN      | inner ACC 90.70% | outer ACC 91.20%
outer fold 3/5 | tuning Logistic | inner ACC 72.85% | outer ACC 73.10%
outer fold 3/5 | tuning RandomForest | inner ACC 93.50% | outer ACC 93.60%
outer fold 4/5 | tuning KNN      | inner ACC 90.10% | outer ACC 90.20%
outer fold 4/5 | tuning Logistic | inner ACC 73.85% | outer ACC 70.10%
outer fold 4/5 | tuning RandomForest | inner ACC 93.50% | outer ACC 92.70%
outer fold 5/5 | tuning KNN      | inner ACC 90.60% | outer ACC 89.50%
outer fold 5/5 | tuning Logistic | inner ACC 72.58% | outer ACC 74.90%
outer fold 5/5 | tuning RandomForest | inner ACC 93.08% | outer ACC 92.90%
CPU times: user 46.5 s, sys: 4.02 s, total: 50.5 s
Wall time: 29min 47s
```

In [16]:
```
# Looking at the results
for name in cv_scores:
    print('%-8s | outer CV acc. %.2f%% +\- %.3f' % (
          name, 100 * np.mean(cv_scores[name]), 100 * np.std(cv_scores[name])))
print()
for name in cv_scores:
    print('{} best parameters'.format(name), gridcvs[name].best_params_)
```

```
KNN      | outer CV acc. 91.36% +\- 1.407
RandomForest | outer CV acc. 93.76% +\- 1.065
Logistic | outer CV acc. 73.02% +\- 1.755

KNN best parameters {'classifier__n_neighbors': 20, 'classifier__weights': 'distance'}
RandomForest best parameters {'classifier__max_features': 6}
Logistic best parameters {'classifier__C': 1.0}
```

In [17]:
```
# Fitting a model to the whole training set
# using the "best" KNN algorithm
best_algo = gridcvs['KNN']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('Accuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['KNN'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
knn_accuracy.append(test_acc)
knn_accuracy_train.append(train_acc)

# using the "best" RandomForest algorithm
best_algo = gridcvs['RandomForest']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['RandomForest'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
randforest_accuracy.append(test_acc)
randforest_accuracy_train.append(train_acc)

# using the "best" Logistic algorithm
best_algo = gridcvs['Logistic']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))


print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['Logistic'].best_params_)
```

```
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
logreg_accuracy.append(test_acc)
logreg_accuracy_train.append(train_acc)
```

```
Accuracy 91.56% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 20, 'classifier__weights': 'distance'}
Training Accuracy: 100.00%
Test Accuracy: 93.19%

Accuracy 94.06% (average over CV test folds)
Best Parameters: {'classifier__max_features': 4}
Training Accuracy: 100.00%
Test Accuracy: 94.85%

Accuracy 72.84% (average over CV test folds)
Best Parameters: {'classifier__C': 10.0}
Training Accuracy: 73.06%
Test Accuracy: 72.37%
```

In [18]:
```
print(knn_accuracy[1])
print(randforest_accuracy[1])
print(logreg_accuracy[1])
```

```
0.9318666666666666
0.9485333333333333
0.7236666666666667
```

Trial 3

In [19]:
```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedKFold

# take all our penguin data, and reserve 20% of it for testing
X_train, X_test, y_train, y_test = train_test_split(X_p, y_p,
                                                    train_size=5000,
                                                    random_state=13245,
                                                    stratify=y_p)


# Initializing Classifiers
clf1 = KNeighborsClassifier()
clf2 = RandomForestClassifier(n_estimators = 1024)
clf3 = LogisticRegression(solver='liblinear', multi_class='auto')

# Declaring parameters
K_list = np.array([n*20 for n in range(1,26)])
F_list = np.array([1, 2, 4, 6, 8, 12, 16])
C_list = np.array([10**(-8), 10**(-7), 10**(-6), 10**(-5), 10**(-4), 10**(-3), 10**(-2), 10**(-1),
                   10**(0), 10**(1), 10**(2), 10**(3), 10**(4)])

# Building the pipelines
pipe1 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf1)])

pipe2 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf2)])

pipe3 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf3)])


# Setting up the parameter grids
param_grid1 = [{'classifier__weights': ['uniform', 'distance'],
                'classifier__n_neighbors': K_list}]

param_grid2 = [{'classifier__max_features': F_list}]

param_grid3 = [{'classifier__C': C_list}]
```

```
# Setting up multiple GridSearchCV objects, 1 for each algorithm
gridcvs = {}

for pgrid, est, name in zip((param_grid1, param_grid2, param_grid3),
                            (pipe1, pipe2, pipe3),
                            ('KNN', 'RandomForest', 'Logistic')):
    gcv = GridSearchCV(estimator=est,
                       param_grid=pgrid,
                       scoring='accuracy',
                       n_jobs=3,
                       cv=5, # just 2-fold inner loop, i.e. train/test
                       verbose=0,
                       return_train_score=True,
                       refit='accuracy')
    gridcvs[name] = gcv
```

In [20]:
```
%%time
# ^^ this handy Jupyter magic times the execution of the cell for you

cv_scores = {name: [] for name, gs_est in gridcvs.items()}

skfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

# The outer loop for algorithm selection
c = 1
for outer_train_idx, outer_valid_idx in skfold.split(X_train,y_train):
    for name, gs_est in sorted(gridcvs.items()):
        print('outer fold %d/5 | tuning %-8s' % (c, name), end='')

        # The inner loop for hyperparameter tuning
        gs_est.fit(X_train.iloc[outer_train_idx], y_train.iloc[outer_train_idx])
        y_pred = gs_est.predict(X_train.iloc[outer_valid_idx])
        acc = accuracy_score(y_true=y_train.iloc[outer_valid_idx], y_pred=y_pred)
        print(' | inner ACC %.2f%% | outer ACC %.2f%%' %
              (gs_est.best_score_ * 100, acc * 100))
        cv_scores[name].append(acc)

    c += 1
```

```
outer fold 1/5 | tuning KNN      | inner ACC 90.62% | outer ACC 91.20%
outer fold 1/5 | tuning Logistic | inner ACC 72.62% | outer ACC 69.30%
outer fold 1/5 | tuning RandomForest | inner ACC 93.77% | outer ACC 94.50%
outer fold 2/5 | tuning KNN      | inner ACC 90.53% | outer ACC 92.60%
outer fold 2/5 | tuning Logistic | inner ACC 71.75% | outer ACC 71.70%
outer fold 2/5 | tuning RandomForest | inner ACC 93.40% | outer ACC 94.70%
outer fold 3/5 | tuning KNN      | inner ACC 90.22% | outer ACC 91.10%
outer fold 3/5 | tuning Logistic | inner ACC 71.95% | outer ACC 73.20%
outer fold 3/5 | tuning RandomForest | inner ACC 93.42% | outer ACC 95.30%
outer fold 4/5 | tuning KNN      | inner ACC 90.50% | outer ACC 90.70%
outer fold 4/5 | tuning Logistic | inner ACC 72.28% | outer ACC 73.80%
outer fold 4/5 | tuning RandomForest | inner ACC 93.73% | outer ACC 92.70%
outer fold 5/5 | tuning KNN      | inner ACC 90.42% | outer ACC 90.30%
outer fold 5/5 | tuning Logistic | inner ACC 72.45% | outer ACC 71.80%
outer fold 5/5 | tuning RandomForest | inner ACC 93.35% | outer ACC 92.60%
CPU times: user 38.9 s, sys: 3.93 s, total: 42.8 s
Wall time: 29min 51s
```

In [21]:
```
# Looking at the results
for name in cv_scores:
    print('%-8s | outer CV acc. %.2f%% +\- %.3f' % (
          name, 100 * np.mean(cv_scores[name]), 100 * np.std(cv_scores[name])))
print()
for name in cv_scores:
    print('{} best parameters'.format(name), gridcvs[name].best_params_)
```

```
KNN      | outer CV acc. 91.18% +\- 0.778
RandomForest | outer CV acc. 93.96% +\- 1.102
Logistic | outer CV acc. 71.96% +\- 1.555

KNN best parameters {'classifier__n_neighbors': 20, 'classifier__weights': 'distance'}
RandomForest best parameters {'classifier__max_features': 4}
Logistic best parameters {'classifier__C': 100.0}
```

In [22]:
```
# Fitting a model to the whole training set
# using the "best" KNN algorithm
best_algo = gridcvs['KNN']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('Accuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['KNN'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
knn_accuracy.append(test_acc)
knn_accuracy_train.append(train_acc)

# using the "best" RandomForest algorithm
best_algo = gridcvs['RandomForest']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['RandomForest'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
randforest_accuracy.append(test_acc)
randforest_accuracy_train.append(train_acc)

# using the "best" Logistic algorithm
best_algo = gridcvs['Logistic']

best_algo.fit(X_train, y_train)
train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))


print('\nAccuracy %.2f%% (average over CV test folds)' %
      (100 * best_algo.best_score_))
print('Best Parameters: %s' % gridcvs['Logistic'].best_params_)
```

```
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))
logreg_accuracy.append(test_acc)
logreg_accuracy_train.append(train_acc)
```

```
Accuracy 91.80% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 20, 'classifier__weights': 'distance'}
Training Accuracy: 100.00%
Test Accuracy: 92.02%

Accuracy 94.24% (average over CV test folds)
Best Parameters: {'classifier__max_features': 4}
Training Accuracy: 100.00%
Test Accuracy: 94.66%

Accuracy 72.14% (average over CV test folds)
Best Parameters: {'classifier__C': 100.0}
Training Accuracy: 72.40%
Test Accuracy: 72.33%
```

In [23]:
```
print(knn_accuracy[2])
print(randforest_accuracy[2])
print(logreg_accuracy[2])
```

```
0.9202
0.9466
0.7233333333333334
```

In [24]:
```
#report average train accuracy per classifier (with best parameter)
print("Average KNN Test Accuracy: ", np.mean(knn_accuracy_train))
print("Average Random Forest Test Accuracy: ", np.mean(randforest_accuracy_train))
print("Average Logistic Regression Test Accuracy: ", np.mean(logreg_accuracy_train))
```

```
Average KNN Test Accuracy:  1.0
Average Random Forest Test Accuracy:  1.0
Average Logistic Regression Test Accuracy:  0.7258666666666667
```

In [25]:
```
#report average test accuracy per classifier (with best parameter)
print("Average KNN Test Accuracy: ", np.mean(knn_accuracy))
print("Average Random Forest Test Accuracy: ", np.mean(randforest_accuracy))
print("Average Logistic Regression Test Accuracy: ", np.mean(logreg_accuracy))
```

```
Average KNN Test Accuracy:  0.9268
Average Random Forest Test Accuracy:  0.9481111111111112
Average Logistic Regression Test Accuracy:  0.7242888888888889
```

T - test for LETTER dataset

In [26]:
```
from scipy import stats
import numpy as np
```

In [27]:
```
# T-test between the different algorithms

knn_forest = stats.ttest_ind(knn_accuracy, randforest_accuracy)

forest_logistic = stats.ttest_ind(randforest_accuracy, logreg_accuracy)

logistic_knn = stats.ttest_ind(logreg_accuracy, knn_accuracy)
```

In [28]:
```
# Results of the T-tests

print('KNN and RandomForest: ', knn_forest)
print('\nRandomForest and Logistic Regression: ', forest_logistic)
print('\nLogistic Regression and KNN: ', logistic_knn)
```

```
KNN and RandomForest:  Ttest_indResult(statistic=-6.018486029250711, pvalue=0.003838995866673888)

RandomForest and Logistic Regression:  Ttest_indResult(statistic=201.03832462525068, pvalue=3.672520186965212
8e-09)

Logistic Regression and KNN:  Ttest_indResult(statistic=-57.1373229787318, pvalue=5.618046883959578e-07)
```

T - test across all 9 trials

In [1]:
```
from scipy import stats
import numpy as np
```

In [2]:
```
# Lists to hold accuracies of 9 test set scoress from all datasets

knn_acc_all = [0.8237024083936094, 0.8230267864239726, 0.8185756299181305,
               0.7697964625736964, 0.7713051116990618, 0.7741297056311327,
               0.9283333333333333, 0.9318666666666666, 0.9202]
forest_acc_all = [0.8501311501470471, 0.8438120976075034, 0.8471902074556872,
                  0.8223422428699402, 0.8233127087630119, 0.8221842600501379,
                  0.9492, 0.9485333333333333, 0.9466]
logistic_acc_all = [0.8424608536682299, 0.8433351879818775, 0.846395358079644,
                    0.7504895731338931, 0.7497447969833961, 0.755926959855003,
                    0.7258666666666667, 0.7236666666666667, 0.7233333333333334]
```

In [4]:
```
# Lists to hold accuracies of 9 train set scoress from all datasets

knn_acc_all_train = [1.0, 1.0, 0.828, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
forest_acc_all_train = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
logistic_acc_all_train = [0.8528, 0.8548, 0.842, 0.7502, 0.7614, 0.7514, 0.723, 0.7306, 0.724]
```

In [5]:
```
# T-test between the different algorithms

knn_forest = stats.ttest_ind(knn_acc_all, forest_acc_all)

forest_logistic = stats.ttest_ind(forest_acc_all, logistic_acc_all)

logistic_knn = stats.ttest_ind(logistic_acc_all, knn_acc_all)
```

In [6]:
```
# Results of the T-tests

print('KNN and RandomForest: ', knn_forest)
print('\nRandomForest and Logistic Regression: ', forest_logistic)
print('\nLogistic Regression and KNN: ', logistic_knn)
```

```
KNN and RandomForest:  Ttest_indResult(statistic=-1.0874015890279447, pvalue=0.29296952937840315)

RandomForest and Logistic Regression:  Ttest_indResult(statistic=3.753740970595021, pvalue=0.0017340131831185
953)

Logistic Regression and KNN:  Ttest_indResult(statistic=-2.283867112188633, pvalue=0.036380095762982935)
```

Average test set accuracy across all 9 trials

In [7]:
```
# Computing the average test set accuracy of the algorithms over 3 trials of each of 3 datasets

print("Average test set KNN Accuracy: ", np.mean(knn_acc_all))
print("Average test set Random Forest Accuracy: ", np.mean(forest_acc_all))
print("Average test set Logistic Regression Accuracy: ", np.mean(logistic_acc_all))
```

```
Average test set KNN Accuracy:  0.8401040116266226
Average test set Random Forest Accuracy:  0.8725895555807401
Average test set Logistic Regression Accuracy:  0.7734688218187457
```

Average train set accuracy across all 9 trials

In [8]:
```
# Computing the average train set accuracy of the algorithms over 3 trials of each of 3 datasets

print("Average train set KNN Accuracy: ", np.mean(knn_acc_all_train))
print("Average train set Random Forest Accuracy: ", np.mean(forest_acc_all_train))
print("Average train set Logistic Regression Accuracy: ", np.mean(logistic_acc_all_train))
```

```
Average train set KNN Accuracy:  0.9808888888888888
Average train set Random Forest Accuracy:  1.0
Average train set Logistic Regression Accuracy:  0.776688888888889
```