In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random

from sklearn.pipeline import make_pipeline

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import KFold

from sklearn.metrics import mean_squared_error

from sklearn.model_selection import cross_val_score

from sklearn.metrics import make_scorer

from sklearn.model_selection import validation_curve

from sklearn.metrics import r2_score

from sklearn.model_selection import learning_curve

from sklearn.svm import SVC

from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import StratifiedKFold

from sklearn.pipeline import Pipeline

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear_model import LogisticRegression

from sklearn import datasets

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.preprocessing import OneHotEncoder
```

Cov_Type

In [2]:
```
import pandas as pd

cov_type_dataset = pd.read_csv('covtype.data.gz')

cov_type_dataset.columns = ['Elevation', 'Aspect', 'Slope', 'Horizontal distance to hydrology features', 'Vertical distance to hydrology features', 'Horizontal distance to roadways', 'Hillshade 9am solstice', 'Hillshade noon solstice', 'Hillshade 3pm solstice', 'Horizontal distance to firepoints ', 'Wilderness area 1', 'Wilderness area 2', 'Wilderness area 3', "Wilderness area 4", 'Soil Type 1', 'Soil Type 2', 'Soil Type 3', 'Soil Type 4', 'Soil Type 5', 'Soil Type 6', 'Soil Type 7', 'Soil Type 8', 'Soil Type 9', 'Soil Type 10', 'Soil Type 11', 'Soil Type 12', 'Soil Type 13', 'Soil Type 14', 'Soil Type 15', 'Soil Type 16', 'Soil Type 17', 'Soil Type 18', 'Soil Type 19', 'Soil Type 20', 'Soil Type 21', 'Soil Type 22', 'Soil Type 23', 'Soil Type 24', 'Soil Type 25', 'Soil Type 26', 'Soil Type 27', 'Soil Type 28', 'Soil Type 29', 'Soil Type 30', 'Soil Type 31', 'Soil Type 32', 'Soil Type 33', 'Soil Type 34', 'Soil Type 35', 'Soil Type 36', 'Soil Type 37', 'Soil Type 38', 'Soil Type 39', 'Soil Type 40', 'Cover_Type']

# cov_type_dataset.head()
# cov_type_mean = np.mean(cov_type_dataset)
# print(cov_type_mean)

cov_type_dataset
```

Out[2]:

	Elevation	Aspect	Slope	Horizontal distance to hydrology features	Vertical distance to hydrology features	Horizontal distance to roadways	Hillshade 9am solstice	Hillshade noon solstice	Hillsl sol
0	2590	56	2	212	-6	390	220	235	
1	2804	139	9	268	65	3180	234	238	
2	2785	155	18	242	118	3090	238	238	
3	2595	45	2	153	-1	391	220	234	
4	2579	132	6	300	-15	67	230	237	
...	...	...	...	...	...	...	...	...	
581006	2396	153	20	85	17	108	240	237	
581007	2391	152	19	67	12	95	240	237	
581008	2386	159	17	60	7	90	236	241	
581009	2384	170	15	60	5	90	230	245	
581010	2383	165	13	60	4	67	231	244	

581011 rows × 55 columns

In [3]:

```
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
```

Trial 1

In [4]:
```
from sklearn import preprocessing

cov_type_dataset = cov_type_dataset[(cov_type_dataset.astype("int64")
!= 3).all(axis=1)]
cov_type_dataset = cov_type_dataset[(cov_type_dataset.astype("int64")
!= 4).all(axis=1)]
cov_type_dataset = cov_type_dataset[(cov_type_dataset.astype("int64")
!= 5).all(axis=1)]
cov_type_dataset = cov_type_dataset[(cov_type_dataset.astype("int64")
!= 6).all(axis=1)]
cov_type_dataset = cov_type_dataset[(cov_type_dataset.astype("int64")
!= 7).all(axis=1)]
cov_type_dataset = cov_type_dataset.drop([1])
cov_type_dict = {1: 0, 2: 1}

cov_type_dataset['Cover_Type'] = cov_type_dataset['Cover_Type'].map(co
v_type_dict)

X = cov_type_dataset.drop(['Cover_Type'],axis=1)
y = cov_type_dataset[['Cover_Type']].to_numpy()


print(y)
print(cov_type_dataset.shape)
```

```
[[1]
 [1]
 [1]
 ...
 [1]
 [1]
 [1]]
(368427, 55)
```

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y.ravel(),
                                                     train_size=0.01357
3,
                                                     random_state=12345
,
                                                     stratify=y)

standardscale = StandardScaler()
X_train = standardscale.fit_transform(X_train)
X_test = standardscale.transform(X_test)

print("Shape of input data X_train: {} and shape of target variable y_
train: {}".format(X_train.shape, y_train.shape))
print("Shape of input data X_test: {} and shape of target variable y_t
est: {}".format(X_test.shape, y_test.shape))
```

```
/Users/adriannahohil/anaconda3/lib/python3.7/site-packages/sklearn/m
odel_selection/_split.py:2179: FutureWarning: From version 0.21, tes
t_size will always complement train_size unless both are specified.
  FutureWarning)
/Users/adriannahohil/anaconda3/lib/python3.7/site-packages/sklearn/p
reprocessing/data.py:625: DataConversionWarning: Data with input dty
pe int64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/Users/adriannahohil/anaconda3/lib/python3.7/site-packages/sklearn/b
ase.py:462: DataConversionWarning: Data with input dtype int64 were
all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
/Users/adriannahohil/anaconda3/lib/python3.7/site-packages/ipykernel
_launcher.py:8: DataConversionWarning: Data with input dtype int64 w
ere all converted to float64 by StandardScaler.


Shape of input data X_train: (5000, 54) and shape of target variable
y_train: (5000,)
Shape of input data X_test: (363427, 54) and shape of target variabl
e y_test: (363427,)
```

In [6]:
```
# Initializing Classifiers

clf1 = KNeighborsClassifier()
clf2 = RandomForestClassifier(n_estimators = 1024)
clf3 = LogisticRegression()

# Building the pipelines

pipe1 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf1)])
pipe2 = Pipeline([('std', StandardScaler()),
                 ('classifier', clf2)])
pipe3 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf3)])

# Declaring some parameter values

C_list = np.power(10., np.arange(-8, 4)) #For Logistic Regression
F_list = [1, 2, 4, 6, 8, 12, 16, 20]
K_list = [n*20 for n in range(1,26)] #Every 20 neighbors up to 500
penalty_list = ['l1','l2']
weight_list = ['uniform','distance']

# 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,
                'classifier__penalty': ['l1','l2']}]

# 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=6,
                       cv=5, # 5-fold inner
                       verbose=0,
                       return_train_score=True)
    gridcvs[name] = gcv
```

In [7]:
```
%%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)

import warnings
# there are a lot of convergence warnings for some params, however be
careful with this!!
# sometimes you need to see those wanrings, and now we've screwed tha
tup for the whole notebook from here on!!
warnings.filterwarnings('ignore')

# 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[outer_train_idx], y_train[outer_train_idx])
        y_pred = gs_est.predict(X_train[outer_valid_idx])
        acc = accuracy_score(y_true=y_train[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

# Looking at the results

for name in cv_scores:
    print('%-8s | outer CV acc. %.2f%% +\- %.3f' % (name, 100 * np.mea
n(cv_scores[name]), 100 * np.std(cv_scores[name])))
print()
for name in cv_scores:
    print('{} best parameters'.format(name), gridcvs[name].best_params
_)
```

```
outer fold 1/5 | tuning KNN      | inner ACC 79.57% | outer ACC 76.9
0%
outer fold 1/5 | tuning Logistic | inner ACC 78.80% | outer ACC 77.0
0%
outer fold 1/5 | tuning RandomForest | inner ACC 82.90% | outer ACC
81.50%
outer fold 2/5 | tuning KNN      | inner ACC 79.00% | outer ACC 79.9
0%
outer fold 2/5 | tuning Logistic | inner ACC 78.83% | outer ACC 75.9
0%
outer fold 2/5 | tuning RandomForest | inner ACC 82.15% | outer ACC
83.60%
outer fold 3/5 | tuning KNN      | inner ACC 78.92% | outer ACC 80.6
0%
outer fold 3/5 | tuning Logistic | inner ACC 77.48% | outer ACC 80.4
0%
outer fold 3/5 | tuning RandomForest | inner ACC 82.00% | outer ACC
84.00%
outer fold 4/5 | tuning KNN      | inner ACC 78.12% | outer ACC 81.2
0%
outer fold 4/5 | tuning Logistic | inner ACC 77.00% | outer ACC 81.4
0%
outer fold 4/5 | tuning RandomForest | inner ACC 81.90% | outer ACC
85.40%
outer fold 5/5 | tuning KNN      | inner ACC 79.60% | outer ACC 79.1
0%
outer fold 5/5 | tuning Logistic | inner ACC 78.77% | outer ACC 75.6
0%
outer fold 5/5 | tuning RandomForest | inner ACC 82.50% | outer ACC
79.90%
KNN      | outer CV acc. 79.54% +\- 1.495
RandomForest | outer CV acc. 82.88% +\- 1.945
Logistic | outer CV acc. 78.06% +\- 2.386

KNN best parameters {'classifier__n_neighbors': 20, 'classifier__wei
ghts': 'distance'}
RandomForest best parameters {'classifier__max_features': 4}
Logistic best parameters {'classifier__C': 0.01, 'classifier__penalt
y': 'l2'}
CPU times: user 42.2 s, sys: 1.87 s, total: 44.1 s
Wall time: 13min 26s
```

In [8]:
```
t1_KNN = gridcvs['KNN']

train_results = {}
test_results = {}

train_acc = accuracy_score(y_true=y_train, y_pred=t1_KNN.predict(X_tra
in))
test_acc = accuracy_score(y_true=y_test, y_pred=t1_KNN.predict(X_test)
)
# print out results
print('Accuracy %.2f%% (average over CV test folds)' % (100 * t1_KNN.b
est_score_))
print('Best Parameters: %s' % gridcvs['KNN'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))

train_results['KNN Train Score'] = train_acc
test_results['KNN Test Score'] = test_acc
```

```
Accuracy 79.60% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 20, 'classifier__weight
s': 'distance'}
Training Accuracy: 95.82%
Test Accuracy: 79.14%
```

In [9]:
```
t1_log_reg = gridcvs['Logistic']

train_acc = accuracy_score(y_true=y_train, y_pred=t1_log_reg.predict(X
_train))
test_acc = accuracy_score(y_true=y_test, y_pred=t1_log_reg.predict(X_t
est))
# print out results
print('Accuracy %.2f%% (average over CV test folds)' % (100 * t1_log_r
eg.best_score_))
print('Best Parameters: %s' % gridcvs['Logistic'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))

train_results['Logistic'] = train_acc
test_results['Logistic'] = test_acc
```

```
Accuracy 78.77% (average over CV test folds)
Best Parameters: {'classifier__C': 0.01, 'classifier__penalty': 'l2'
}
Training Accuracy: 78.28%
Test Accuracy: 78.11%
```

In [10]:
```
t1_rand_for = gridcvs['RandomForest']

train_acc = accuracy_score(y_true=y_train, y_pred=t1_rand_for.predict(
X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=t1_rand_for.predict(X_
test))
# print out results
print('Accuracy %.2f%% (average over CV test folds)' % (100 * t1_rand_
for.best_score_))
print('Best Parameters: %s' % gridcvs['RandomForest'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))

train_results['RandomForest'] = train_acc
test_results['RandomForest'] = test_acc
```

```
Accuracy 82.50% (average over CV test folds)
Best Parameters: {'classifier__max_features': 4}
Training Accuracy: 95.98%
Test Accuracy: 83.08%
```

In [ ]:

In [ ]:

In [ ]:

Trial 2

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y.ravel(),
                                                     train_size=0.01357
3,
                                                     random_state=3221,
                                                     stratify=y)

standardscale = StandardScaler()
X_train = standardscale.fit_transform(X_train)
X_test = standardscale.transform(X_test)

print("Shape of input data X_train: {} and shape of target variable y_
train: {}".format(X_train.shape, y_train.shape))
print("Shape of input data X_test: {} and shape of target variable y_t
est: {}".format(X_test.shape, y_test.shape))
```

```
Shape of input data X_train: (5000, 54) and shape of target variable
y_train: (5000,)
Shape of input data X_test: (363427, 54) and shape of target variabl
e y_test: (363427,)
```

In [12]:
```
# Initializing Classifiers

clf1 = KNeighborsClassifier()
clf2 = RandomForestClassifier(n_estimators = 1024)
clf3 = LogisticRegression()

# Building the pipelines

pipe1 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf1)])
pipe2 = Pipeline([('std', StandardScaler()),
                 ('classifier', clf2)])
pipe3 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf3)])

# Declaring some parameter values

C_list = np.power(10., np.arange(-8, 4)) #For Logistic Regression
F_list = [1, 2, 4, 6, 8, 12, 16, 20]
K_list = [n*20 for n in range(1,26)] #Every 20 neighbors up to 500
penalty_list = ['l1','l2']
weight_list = ['uniform','distance']

# 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,
                'classifier__penalty': ['l1','l2']}]

# 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=6,
                       cv=5, # 5-fold inner
                       verbose=0,
                       return_train_score=True)
    gridcvs[name] = gcv
```

In [13]:
```
%%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)

import warnings
# there are a lot of convergence warnings for some params, however be careful with this!!
# sometimes you need to see those wanrings, and now we've screwed that up for the whole notebook from here on!!
warnings.filterwarnings('ignore')

# 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[outer_train_idx], y_train[outer_train_idx])
        y_pred = gs_est.predict(X_train[outer_valid_idx])
        acc = accuracy_score(y_true=y_train[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

# 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_)
```

```
outer fold 1/5 | tuning KNN      | inner ACC 79.38% | outer ACC 78.6
0%
outer fold 1/5 | tuning Logistic | inner ACC 78.85% | outer ACC 78.9
0%
outer fold 1/5 | tuning RandomForest | inner ACC 83.35% | outer ACC
82.30%
outer fold 2/5 | tuning KNN      | inner ACC 79.25% | outer ACC 81.6
0%
outer fold 2/5 | tuning Logistic | inner ACC 79.15% | outer ACC 78.0
0%
outer fold 2/5 | tuning RandomForest | inner ACC 82.78% | outer ACC
83.00%
outer fold 3/5 | tuning KNN      | inner ACC 78.80% | outer ACC 80.6
0%
outer fold 3/5 | tuning Logistic | inner ACC 79.17% | outer ACC 79.1
0%
outer fold 3/5 | tuning RandomForest | inner ACC 83.60% | outer ACC
83.30%
outer fold 4/5 | tuning KNN      | inner ACC 78.25% | outer ACC 81.7
0%
outer fold 4/5 | tuning Logistic | inner ACC 78.38% | outer ACC 79.8
0%
outer fold 4/5 | tuning RandomForest | inner ACC 83.15% | outer ACC
84.20%
outer fold 5/5 | tuning KNN      | inner ACC 79.42% | outer ACC 77.7
0%
outer fold 5/5 | tuning Logistic | inner ACC 79.00% | outer ACC 77.9
0%
outer fold 5/5 | tuning RandomForest | inner ACC 83.33% | outer ACC
83.50%
KNN      | outer CV acc. 80.04% +\- 1.616
RandomForest | outer CV acc. 83.26% +\- 0.622
Logistic | outer CV acc. 78.74% +\- 0.712

KNN best parameters {'classifier__n_neighbors': 20, 'classifier__wei
ghts': 'distance'}
RandomForest best parameters {'classifier__max_features': 2}
Logistic best parameters {'classifier__C': 0.1, 'classifier__penalty
': 'l1'}
CPU times: user 51.2 s, sys: 1.92 s, total: 53.2 s
Wall time: 13min 54s
```

In [14]:
```
t2_KNN = gridcvs['KNN']

train_results = {}
test_results = {}

train_acc = accuracy_score(y_true=y_train, y_pred=t2_KNN.predict(X_tra
in))
test_acc = accuracy_score(y_true=y_test, y_pred=t2_KNN.predict(X_test)
)
# print out results
print('Accuracy %.2f%% (average over CV test folds)' % (100 * t2_KNN.b
est_score_))
print('Best Parameters: %s' % gridcvs['KNN'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))

train_results['KNN Train Score'] = train_acc
test_results['KNN Test Score'] = test_acc
```

```
Accuracy 79.42% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 20, 'classifier__weight
s': 'distance'}
Training Accuracy: 95.54%
Test Accuracy: 79.26%
```

In [15]:
```
t2_log_reg = gridcvs['Logistic']

train_acc = accuracy_score(y_true=y_train, y_pred=t2_log_reg.predict(X
_train))
test_acc = accuracy_score(y_true=y_test, y_pred=t2_log_reg.predict(X_t
est))
# print out results
print('Accuracy %.2f%% (average over CV test folds)' % (100 * t2_log_r
eg.best_score_))
print('Best Parameters: %s' % gridcvs['Logistic'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))

train_results['Logistic'] = train_acc
test_results['Logistic'] = test_acc
```

```
Accuracy 79.00% (average over CV test folds)
Best Parameters: {'classifier__C': 0.1, 'classifier__penalty': 'l1'}
Training Accuracy: 79.00%
Test Accuracy: 78.31%
```

In [16]:
```
t2_rand_for = gridcvs['RandomForest']

train_acc = accuracy_score(y_true=y_train, y_pred=t2_rand_for.predict(
X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=t2_rand_for.predict(X_
test))
# print out results
print('Accuracy %.2f%% (average over CV test folds)' % (100 * t2_rand_
for.best_score_))
print('Best Parameters: %s' % gridcvs['RandomForest'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))

train_results['RandomForest'] = train_acc
test_results['RandomForest'] = test_acc
```

```
Accuracy 83.33% (average over CV test folds)
Best Parameters: {'classifier__max_features': 2}
Training Accuracy: 96.70%
Test Accuracy: 83.07%
```

In [ ]:

In [ ]:

In [ ]:

Trial 3

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y.ravel(),
                                                    train_size=0.01357
3,
                                                    random_state=2551,
                                                    stratify=y)

standardscale = StandardScaler()
X_train = standardscale.fit_transform(X_train)
X_test = standardscale.transform(X_test)

print("Shape of input data X_train: {} and shape of target variable y_
train: {}".format(X_train.shape, y_train.shape))
print("Shape of input data X_test: {} and shape of target variable y_t
est: {}".format(X_test.shape, y_test.shape))
```

```
Shape of input data X_train: (5000, 54) and shape of target variable
y_train: (5000,)
Shape of input data X_test: (363427, 54) and shape of target variabl
e y_test: (363427,)
```

In [18]:
```
# Initializing Classifiers

clf1 = KNeighborsClassifier()
clf2 = RandomForestClassifier(n_estimators = 1024)
clf3 = LogisticRegression()

# Building the pipelines

pipe1 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf1)])
pipe2 = Pipeline([('std', StandardScaler()),
                 ('classifier', clf2)])
pipe3 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf3)])

# Declaring some parameter values

C_list = np.power(10., np.arange(-8, 4)) #For Logistic Regression
F_list = [1, 2, 4, 6, 8, 12, 16, 20]
K_list = [n*20 for n in range(1,26)] #Every 20 neighbors up to 500
penalty_list = ['l1','l2']
weight_list = ['uniform','distance']

# 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,
                'classifier__penalty': ['l1','l2']}]

# 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=6,
                       cv=5, # 5-fold inner
                       verbose=0,
                       return_train_score=True)
    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)

import warnings
# there are a lot of convergence warnings for some params, however be
careful with this!!
# sometimes you need to see those wanrings, and now we've screwed tha
tup for the whole notebook from here on!!
warnings.filterwarnings('ignore')

# 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[outer_train_idx], y_train[outer_train_idx])
        y_pred = gs_est.predict(X_train[outer_valid_idx])
        acc = accuracy_score(y_true=y_train[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

# Looking at the results

for name in cv_scores:
    print('%-8s | outer CV acc. %.2f%% +\- %.3f' % (name, 100 * np.mea
n(cv_scores[name]), 100 * np.std(cv_scores[name])))
print()
for name in cv_scores:
    print('{} best parameters'.format(name), gridcvs[name].best_params
_)
```

```
outer fold 1/5 | tuning KNN      | inner ACC 78.25% | outer ACC 78.0
0%
outer fold 1/5 | tuning Logistic | inner ACC 77.60% | outer ACC 77.6
0%
outer fold 1/5 | tuning RandomForest | inner ACC 82.40% | outer ACC
83.10%
outer fold 2/5 | tuning KNN      | inner ACC 78.08% | outer ACC 81.0
0%
outer fold 2/5 | tuning Logistic | inner ACC 77.12% | outer ACC 78.7
0%
outer fold 2/5 | tuning RandomForest | inner ACC 82.42% | outer ACC
83.70%
outer fold 3/5 | tuning KNN      | inner ACC 78.65% | outer ACC 79.0
0%
outer fold 3/5 | tuning Logistic | inner ACC 77.58% | outer ACC 76.1
0%
outer fold 3/5 | tuning RandomForest | inner ACC 83.43% | outer ACC
82.40%
outer fold 4/5 | tuning KNN      | inner ACC 78.27% | outer ACC 76.5
0%
outer fold 4/5 | tuning Logistic | inner ACC 77.50% | outer ACC 77.8
0%
outer fold 4/5 | tuning RandomForest | inner ACC 82.73% | outer ACC
83.50%
outer fold 5/5 | tuning KNN      | inner ACC 78.40% | outer ACC 78.7
0%
outer fold 5/5 | tuning Logistic | inner ACC 77.53% | outer ACC 78.2
0%
outer fold 5/5 | tuning RandomForest | inner ACC 83.47% | outer ACC
81.90%
KNN      | outer CV acc. 78.64% +\- 1.462
RandomForest | outer CV acc. 82.92% +\- 0.676
Logistic | outer CV acc. 77.68% +\- 0.875

KNN best parameters {'classifier__n_neighbors': 20, 'classifier__wei
ghts': 'distance'}
RandomForest best parameters {'classifier__max_features': 4}
Logistic best parameters {'classifier__C': 0.1, 'classifier__penalty
': 'l1'}
CPU times: user 38.6 s, sys: 1.75 s, total: 40.3 s
Wall time: 13min 7s
```

In [20]:
```
t3_KNN = gridcvs['KNN']

train_results = {}
test_results = {}

train_acc = accuracy_score(y_true=y_train, y_pred=t3_KNN.predict(X_tra
in))
test_acc = accuracy_score(y_true=y_test, y_pred=t3_KNN.predict(X_test)
)
# print out results
print('Accuracy %.2f%% (average over CV test folds)' % (100 * t3_KNN.b
est_score_))
print('Best Parameters: %s' % gridcvs['KNN'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))

train_results['KNN Train Score'] = train_acc
test_results['KNN Test Score'] = test_acc
```

```
Accuracy 78.40% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 20, 'classifier__weight
s': 'distance'}
Training Accuracy: 95.74%
Test Accuracy: 78.73%
```

In [21]:
```
t3_log_reg = gridcvs['Logistic']

train_acc = accuracy_score(y_true=y_train, y_pred=t3_log_reg.predict(X
_train))
test_acc = accuracy_score(y_true=y_test, y_pred=t3_log_reg.predict(X_t
est))
# print out results
print('Accuracy %.2f%% (average over CV test folds)' % (100 * t3_log_r
eg.best_score_))
print('Best Parameters: %s' % gridcvs['Logistic'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))

train_results['Logistic'] = train_acc
test_results['Logistic'] = test_acc
```

```
Accuracy 77.53% (average over CV test folds)
Best Parameters: {'classifier__C': 0.1, 'classifier__penalty': 'l1'}
Training Accuracy: 78.20%
Test Accuracy: 78.37%
```

In [22]:
```
t3_rand_for = gridcvs['RandomForest']

train_acc = accuracy_score(y_true=y_train, y_pred=t3_rand_for.predict(
X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=t3_rand_for.predict(X_
test))
# print out results
print('Accuracy %.2f%% (average over CV test folds)' % (100 * t3_rand_
for.best_score_))
print('Best Parameters: %s' % gridcvs['RandomForest'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))

train_results['RandomForest'] = train_acc
test_results['RandomForest'] = test_acc
```

```
Accuracy 83.47% (average over CV test folds)
Best Parameters: {'classifier__max_features': 4}
Training Accuracy: 96.38%
Test Accuracy: 83.42%
```

Trial 4 (Extra Credit)

In [23]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y.ravel(),
                                                    train_size=0.01357
3,
                                                    random_state=8773,
                                                    stratify=y)


standardscale = StandardScaler()
X_train = standardscale.fit_transform(X_train)
X_test = standardscale.transform(X_test)

print("Shape of input data X_train: {} and shape of target variable y_
train: {}".format(X_train.shape, y_train.shape))
print("Shape of input data X_test: {} and shape of target variable y_t
est: {}".format(X_test.shape, y_test.shape))
```

```
Shape of input data X_train: (5000, 54) and shape of target variable
y_train: (5000,)
Shape of input data X_test: (363427, 54) and shape of target variabl
e y_test: (363427,)
```

In [24]:
```
# Initializing Classifiers

clf1 = KNeighborsClassifier()
clf2 = RandomForestClassifier(n_estimators = 1024)
clf3 = LogisticRegression()

# Building the pipelines

pipe1 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf1)])
pipe2 = Pipeline([('std', StandardScaler()),
                 ('classifier', clf2)])
pipe3 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf3)])

# Declaring some parameter values

C_list = np.power(10., np.arange(-8, 4)) #For Logistic Regression
F_list = [1, 2, 4, 6, 8, 12, 16, 20]
K_list = [n*20 for n in range(1,26)] #Every 20 neighbors up to 500
penalty_list = ['l1','l2']
weight_list = ['uniform','distance']

# 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,
                'classifier__penalty': ['l1','l2']}]

# 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=6,
                       cv=5, # 5-fold inner
                       verbose=0,
                       return_train_score=True)
    gridcvs[name] = gcv
```

In [25]:
```
%%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)

import warnings
# there are a lot of convergence warnings for some params, however be careful with this!!
# sometimes you need to see those wanrings, and now we've screwed that up for the whole notebook from here on!!
warnings.filterwarnings('ignore')

# 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[outer_train_idx], y_train[outer_train_idx])
        y_pred = gs_est.predict(X_train[outer_valid_idx])
        acc = accuracy_score(y_true=y_train[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

# 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_)
```

```
outer fold 1/5 | tuning KNN      | inner ACC 77.85% | outer ACC 79.5
0%
outer fold 1/5 | tuning Logistic | inner ACC 78.22% | outer ACC 76.9
0%
outer fold 1/5 | tuning RandomForest | inner ACC 82.95% | outer ACC
82.10%
outer fold 2/5 | tuning KNN      | inner ACC 78.00% | outer ACC 78.3
0%
outer fold 2/5 | tuning Logistic | inner ACC 78.08% | outer ACC 77.4
0%
outer fold 2/5 | tuning RandomForest | inner ACC 83.08% | outer ACC
82.90%
outer fold 3/5 | tuning KNN      | inner ACC 77.90% | outer ACC 81.0
0%
outer fold 3/5 | tuning Logistic | inner ACC 78.08% | outer ACC 77.8
0%
outer fold 3/5 | tuning RandomForest | inner ACC 83.62% | outer ACC
83.70%
outer fold 4/5 | tuning KNN      | inner ACC 77.88% | outer ACC 79.8
0%
outer fold 4/5 | tuning Logistic | inner ACC 77.20% | outer ACC 80.4
0%
outer fold 4/5 | tuning RandomForest | inner ACC 82.50% | outer ACC
85.00%
outer fold 5/5 | tuning KNN      | inner ACC 78.30% | outer ACC 77.5
0%
outer fold 5/5 | tuning Logistic | inner ACC 78.20% | outer ACC 77.1
0%
outer fold 5/5 | tuning RandomForest | inner ACC 83.85% | outer ACC
82.00%
KNN      | outer CV acc. 79.22% +\- 1.216
RandomForest | outer CV acc. 83.14% +\- 1.115
Logistic | outer CV acc. 77.92% +\- 1.277

KNN best parameters {'classifier__n_neighbors': 20, 'classifier__wei
ghts': 'distance'}
RandomForest best parameters {'classifier__max_features': 16}
Logistic best parameters {'classifier__C': 0.1, 'classifier__penalty
': 'l1'}
CPU times: user 49.3 s, sys: 1.56 s, total: 50.9 s
Wall time: 12min 22s
```

In [26]:
```
t4_KNN = gridcvs['KNN']

train_results = {}
test_results = {}

train_acc = accuracy_score(y_true=y_train, y_pred=t4_KNN.predict(X_tra
in))
test_acc = accuracy_score(y_true=y_test, y_pred=t4_KNN.predict(X_test)
)
# print out results
print('Accuracy %.2f%% (average over CV test folds)' % (100 * t4_KNN.b
est_score_))
print('Best Parameters: %s' % gridcvs['KNN'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))

train_results['KNN Train Score'] = train_acc
test_results['KNN Test Score'] = test_acc
```

```
Accuracy 78.30% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 20, 'classifier__weight
s': 'distance'}
Training Accuracy: 95.50%
Test Accuracy: 79.67%
```

In [27]:
```
t4_log_reg = gridcvs['Logistic']

train_acc = accuracy_score(y_true=y_train, y_pred=t4_log_reg.predict(X
_train))
test_acc = accuracy_score(y_true=y_test, y_pred=t4_log_reg.predict(X_t
est))
# print out results
print('Accuracy %.2f%% (average over CV test folds)' % (100 * t4_log_r
eg.best_score_))
print('Best Parameters: %s' % gridcvs['Logistic'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))

train_results['Logistic'] = train_acc
test_results['Logistic'] = test_acc
```

```
Accuracy 78.20% (average over CV test folds)
Best Parameters: {'classifier__C': 0.1, 'classifier__penalty': 'l1'}
Training Accuracy: 78.08%
Test Accuracy: 77.95%
```

In [28]:
```
t4_rand_for = gridcvs['RandomForest']

train_acc = accuracy_score(y_true=y_train, y_pred=t4_rand_for.predict(
X_train))
test_acc = accuracy_score(y_true=y_test, y_pred=t4_rand_for.predict(X_
test))
# print out results
print('Accuracy %.2f%% (average over CV test folds)' % (100 * t4_rand_
for.best_score_))
print('Best Parameters: %s' % gridcvs['RandomForest'].best_params_)
print('Training Accuracy: %.2f%%' % (100 * train_acc))
print('Test Accuracy: %.2f%%' % (100 * test_acc))

train_results['RandomForest'] = train_acc
test_results['RandomForest'] = test_acc
```

```
Accuracy 83.85% (average over CV test folds)
Best Parameters: {'classifier__max_features': 16}
Training Accuracy: 96.40%
Test Accuracy: 83.22%
```

In [ ]: