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
```

In [2]:
```
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
```

Adult Data

In [3]:
```
import pandas as pd

adult_dataset = pd.read_csv('adult.data')

adult_dataset.columns = ['Age', 'Work Class', 'fnlwgt', 'Education level', 'Education_num', 'Marital status', 'Occupation', 'Relationship', 'Race', 'Sex', 'Capital Gain', 'Capital Loss', 'Hours per week', 'Native Country', 'Yearly Income']
# adult_dataset = pd.get_dummies(adult_dataset)

adult_dataset.columns = ['Age', 'Work Class', 'fnlwgt', 'Education level', 'Education_num', 'Marital status', 'Occupation', 'Relationship', 'Race', 'Sex', 'Capital Gain', 'Capital Loss', 'Hours per week', 'Native Country', 'Yearly Income']

adult_dataset = adult_dataset[(adult_dataset.astype(str) != ' ?').all(axis=1)]
# a = '?'
# adult_dataset[:0]
# print(adult_dataset.str.find(a))

adult_dataset
```

Out[3]:

	Age	Work Class	fnlwgt	Education level	Education_num	Marital status	Occupation	Relationship
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife
...	...	...	...	...	...	...	...	...
32555	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife
32556	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband
32557	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried
32558	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child
32559	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife

30161 rows × 15 columns

In [4]:

```
from sklearn import preprocessing
lb = preprocessing.LabelBinarizer()

ohe = OneHotEncoder(handle_unknown='ignore')
X = adult_dataset.drop(['Yearly Income'],axis=1)
X = ohe.fit_transform(adult_dataset[['Work Class', 'Education level',
'Marital status', 'Occupation', 'Relationship', 'Race', 'Sex', 'Native
Country']]).toarray()
y = lb.fit_transform(adult_dataset[['Yearly Income']])
y = np.reshape(y, (30161, ))

print(X.shape)
print(y.shape)
```

```
(30161, 98)
(30161,)
```

In [5]:

```
from sklearn import preprocessing

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=0.16578
,
                                                    random_state=12345
,
                                                    stratify=y)

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


print(X.shape)
print(y.shape)
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))
```

```
(30161, 98)
(30161,)
Shape of input data X_train: (5000, 98) and shape of target variable
y_train: (5000,)
Shape of input data X_test: (25161, 98) and shape of target variable
y_test: (25161,)

/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)
```

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

# scoring_metrics ='accuracy'
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, # 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

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')

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[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 81.80% | outer ACC 81.2
0%
outer fold 1/5 | tuning Logistic | inner ACC 82.30% | outer ACC 81.3
0%
outer fold 1/5 | tuning RandomForest | inner ACC 81.73% | outer ACC
80.60%
outer fold 2/5 | tuning KNN      | inner ACC 81.65% | outer ACC 80.3
0%
outer fold 2/5 | tuning Logistic | inner ACC 82.62% | outer ACC 80.4
0%
outer fold 2/5 | tuning RandomForest | inner ACC 81.10% | outer ACC
80.70%
outer fold 3/5 | tuning KNN      | inner ACC 81.88% | outer ACC 80.5
0%
outer fold 3/5 | tuning Logistic | inner ACC 82.23% | outer ACC 81.0
0%
outer fold 3/5 | tuning RandomForest | inner ACC 81.47% | outer ACC
81.40%
outer fold 4/5 | tuning KNN      | inner ACC 81.40% | outer ACC 81.8
0%
outer fold 4/5 | tuning Logistic | inner ACC 81.65% | outer ACC 84.2
0%
outer fold 4/5 | tuning RandomForest | inner ACC 81.12% | outer ACC
82.30%
outer fold 5/5 | tuning KNN      | inner ACC 81.50% | outer ACC 81.1
0%
outer fold 5/5 | tuning Logistic | inner ACC 81.90% | outer ACC 82.4
0%
outer fold 5/5 | tuning RandomForest | inner ACC 81.30% | outer ACC
80.20%
KNN      | outer CV acc. 80.98% +\- 0.534
RandomForest | outer CV acc. 81.04% +\- 0.739
Logistic | outer CV acc. 81.86% +\- 1.338

KNN best parameters {'classifier__n_neighbors': 180, 'classifier__we
ights': 'uniform'}
RandomForest best parameters {'classifier__max_features': 16}
Logistic best parameters {'classifier__C': 10.0, 'classifier__penalt
y': 'l2'}
CPU times: user 46.4 s, sys: 2.07 s, total: 48.4 s
Wall time: 27min 59s
```

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 81.50% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 180, 'classifier__weigh
ts': 'uniform'}
Training Accuracy: 81.40%
Test Accuracy: 81.96%
```

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 81.90% (average over CV test folds)
Best Parameters: {'classifier__C': 10.0, 'classifier__penalty': 'l2'
}
Training Accuracy: 82.74%
Test Accuracy: 82.20%
```

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 81.30% (average over CV test folds)
Best Parameters: {'classifier__max_features': 16}
Training Accuracy: 86.94%
Test Accuracy: 80.86%
```

In [ ]:

In [ ]:

In [ ]:

Trial 2

In [11]:

```
# Trial 2 Adult Data

from sklearn import preprocessing

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                     train_size=0.16578
,
                                                     random_state=1369,
                                                     stratify=y)


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

print(X.shape)
print(y.shape)
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))
```

```
(30161, 98)
(30161,)
Shape of input data X_train: (5000, 98) and shape of target variable
y_train: (5000,)
Shape of input data X_test: (25161, 98) and shape of target variable
y_test: (25161,)
```

In [12]:

```
#cross validation is a random subset of training data

#nested cross validation

#Take the true validation performance as the average of the # of K fol
ds

#For Adult Dataset

# 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

# scoring_metrics ='accuracy'
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, # 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

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')

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[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 83.10% | outer ACC 81.3
0%
outer fold 1/5 | tuning Logistic | inner ACC 83.35% | outer ACC 80.6
0%
outer fold 1/5 | tuning RandomForest | inner ACC 82.38% | outer ACC
79.60%
outer fold 2/5 | tuning KNN      | inner ACC 82.08% | outer ACC 82.3
0%
outer fold 2/5 | tuning Logistic | inner ACC 82.67% | outer ACC 84.1
0%
outer fold 2/5 | tuning RandomForest | inner ACC 81.33% | outer ACC
80.80%
outer fold 3/5 | tuning KNN      | inner ACC 82.73% | outer ACC 82.1
0%
outer fold 3/5 | tuning Logistic | inner ACC 83.15% | outer ACC 82.7
0%
outer fold 3/5 | tuning RandomForest | inner ACC 81.73% | outer ACC
80.50%
outer fold 4/5 | tuning KNN      | inner ACC 82.35% | outer ACC 82.7
0%
outer fold 4/5 | tuning Logistic | inner ACC 83.15% | outer ACC 83.1
0%
outer fold 4/5 | tuning RandomForest | inner ACC 81.35% | outer ACC
82.60%
outer fold 5/5 | tuning KNN      | inner ACC 82.42% | outer ACC 82.6
0%
outer fold 5/5 | tuning Logistic | inner ACC 82.62% | outer ACC 83.3
0%
outer fold 5/5 | tuning RandomForest | inner ACC 81.42% | outer ACC
81.80%
KNN      | outer CV acc. 82.20% +\- 0.498
RandomForest | outer CV acc. 81.06% +\- 1.042
Logistic | outer CV acc. 82.76% +\- 1.172

KNN best parameters {'classifier__n_neighbors': 200, 'classifier__we
ights': 'uniform'}
RandomForest best parameters {'classifier__max_features': 1}
Logistic best parameters {'classifier__C': 0.1, 'classifier__penalty
': 'l1'}
CPU times: user 38.4 s, sys: 1.29 s, total: 39.7 s
Wall time: 19min 32s
```

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 82.42% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 200, 'classifier__weigh
ts': 'uniform'}
Training Accuracy: 82.48%
Test Accuracy: 81.87%
```

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 82.62% (average over CV test folds)
Best Parameters: {'classifier__C': 0.1, 'classifier__penalty': 'l1'}
Training Accuracy: 83.46%
Test Accuracy: 82.19%
```

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 81.42% (average over CV test folds)
Best Parameters: {'classifier__max_features': 1}
Training Accuracy: 87.60%
Test Accuracy: 81.19%
```

In [ ]:

In [ ]:

In [ ]:

Trial 3

In [17]:

```
# Trial 3 Adult Data

from sklearn import preprocessing

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=0.16578
,
                                                    random_state=5151,
                                                    stratify=y)

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

print(X.shape)
print(y.shape)
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))
```

```
(30161, 98)
(30161,)
Shape of input data X_train: (5000, 98) and shape of target variable
y_train: (5000,)
Shape of input data X_test: (25161, 98) and shape of target variable
y_test: (25161,)
```

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

# scoring_metrics ='accuracy'
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, # 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

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')

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[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 81.67% | outer ACC 82.3
0%
outer fold 1/5 | tuning Logistic | inner ACC 82.25% | outer ACC 82.4
0%
outer fold 1/5 | tuning RandomForest | inner ACC 81.12% | outer ACC
82.30%
outer fold 2/5 | tuning KNN      | inner ACC 82.67% | outer ACC 80.9
0%
outer fold 2/5 | tuning Logistic | inner ACC 83.53% | outer ACC 80.0
0%
outer fold 2/5 | tuning RandomForest | inner ACC 82.35% | outer ACC
79.00%
outer fold 3/5 | tuning KNN      | inner ACC 82.08% | outer ACC 81.2
0%
outer fold 3/5 | tuning Logistic | inner ACC 83.08% | outer ACC 81.8
0%
outer fold 3/5 | tuning RandomForest | inner ACC 81.58% | outer ACC
82.60%
outer fold 4/5 | tuning KNN      | inner ACC 81.85% | outer ACC 83.6
0%
outer fold 4/5 | tuning Logistic | inner ACC 82.08% | outer ACC 83.1
0%
outer fold 4/5 | tuning RandomForest | inner ACC 81.08% | outer ACC
82.30%
outer fold 5/5 | tuning KNN      | inner ACC 82.12% | outer ACC 81.6
0%
outer fold 5/5 | tuning Logistic | inner ACC 82.40% | outer ACC 84.1
0%
outer fold 5/5 | tuning RandomForest | inner ACC 80.83% | outer ACC
81.90%
KNN      | outer CV acc. 81.92% +\- 0.962
RandomForest | outer CV acc. 81.62% +\- 1.329
Logistic | outer CV acc. 82.28% +\- 1.373

KNN best parameters {'classifier__n_neighbors': 140, 'classifier__we
ights': 'uniform'}
RandomForest best parameters {'classifier__max_features': 6}
Logistic best parameters {'classifier__C': 0.001, 'classifier__penal
ty': 'l2'}
CPU times: user 35.5 s, sys: 1.3 s, total: 36.9 s
Wall time: 19min 36s
```

In [20]:

```python
t3_KNN = gridcvs['KNN']

train_results = {}
test_results = {}

train_acc = accuracy_score(y_true=y_train, y_pred=t3_KNN.predict(X_train))
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.best_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 82.12% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 140, 'classifier__weights': 'uniform'}
Training Accuracy: 82.04%
Test Accuracy: 81.46%
```

In [21]:

```python
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_test))
# print out results
print('Accuracy %.2f%% (average over CV test folds)' % (100 * t3_log_reg.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 82.40% (average over CV test folds)
Best Parameters: {'classifier__C': 0.001, 'classifier__penalty': 'l2'}
Training Accuracy: 83.10%
Test Accuracy: 81.96%
```

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 80.83% (average over CV test folds)
Best Parameters: {'classifier__max_features': 6}
Training Accuracy: 87.70%
Test Accuracy: 81.25%
```

Trial 4 (Extra Credit)

In [23]:

```
from sklearn import preprocessing

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=0.16578
,
                                                    random_state=8773,
                                                    stratify=y)

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

print(X.shape)
print(y.shape)
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))
```

```
(30161, 98)
(30161,)
Shape of input data X_train: (5000, 98) and shape of target variable
y_train: (5000,)
Shape of input data X_test: (25161, 98) and shape of target variable
y_test: (25161,)
```

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

# scoring_metrics ='accuracy'
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, # 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

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')

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[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 80.62% | outer ACC 82.1
0%
outer fold 1/5 | tuning Logistic | inner ACC 81.75% | outer ACC 82.8
0%
outer fold 1/5 | tuning RandomForest | inner ACC 80.62% | outer ACC
80.40%
outer fold 2/5 | tuning KNN      | inner ACC 81.33% | outer ACC 80.8
0%
outer fold 2/5 | tuning Logistic | inner ACC 81.85% | outer ACC 82.2
0%
outer fold 2/5 | tuning RandomForest | inner ACC 80.20% | outer ACC
81.00%
outer fold 3/5 | tuning KNN      | inner ACC 81.70% | outer ACC 79.6
0%
outer fold 3/5 | tuning Logistic | inner ACC 81.55% | outer ACC 82.2
0%
outer fold 3/5 | tuning RandomForest | inner ACC 80.12% | outer ACC
79.40%
outer fold 4/5 | tuning KNN      | inner ACC 81.25% | outer ACC 80.7
0%
outer fold 4/5 | tuning Logistic | inner ACC 82.03% | outer ACC 80.7
0%
outer fold 4/5 | tuning RandomForest | inner ACC 80.05% | outer ACC
80.00%
outer fold 5/5 | tuning KNN      | inner ACC 81.23% | outer ACC 80.8
0%
outer fold 5/5 | tuning Logistic | inner ACC 82.20% | outer ACC 83.0
0%
outer fold 5/5 | tuning RandomForest | inner ACC 80.50% | outer ACC
80.70%
KNN      | outer CV acc. 80.80% +\- 0.792
RandomForest | outer CV acc. 80.30% +\- 0.559
Logistic | outer CV acc. 82.18% +\- 0.806

KNN best parameters {'classifier__n_neighbors': 80, 'classifier__wei
ghts': 'uniform'}
RandomForest best parameters {'classifier__max_features': 8}
Logistic best parameters {'classifier__C': 0.01, 'classifier__penalt
y': 'l2'}
CPU times: user 36 s, sys: 1.19 s, total: 37.2 s
Wall time: 19min 26s
```

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 81.23% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 80, 'classifier__weight
s': 'uniform'}
Training Accuracy: 81.42%
Test Accuracy: 81.72%
```

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 82.20% (average over CV test folds)
Best Parameters: {'classifier__C': 0.01, 'classifier__penalty': 'l2'
}
Training Accuracy: 82.78%
Test Accuracy: 82.61%
```

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 80.50% (average over CV test folds)
Best Parameters: {'classifier__max_features': 8}
Training Accuracy: 86.66%
Test Accuracy: 80.94%
```

In [ ]: