In [4]:
```
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
```

Dataset 1: LETTER

In [5]:
```
import pandas as pd

letter_dataset = pd.read_csv('letter-recognition.data')

letter_dataset.columns = ['Capital Letter', 'Horizontal Position of box', 'Vertical Position of box', 'Width of box', 'Height of box', 'Total # on pixels', 'Mean x of on pixels in box', 'Mean y of on pixels in box', 'Mean x variance', 'Mean y variance', 'Mean x y correlation', 'Mean of x * x * y', 'Mean of x * y * y', 'Mean edge count left to right', 'Correlation of x-edge with y', 'Mean edge count bottom to top', 'correlation of y-edge with x']

# letter_dataset.head()
# letter_mean = np.mean(letter_dataset)
# print(letter_mean)

letter_dataset
```

Out[5]:

	Capital Letter	Horizontal Position of box	Vertical Position of box	Width of box	Height of box	Total # on pixels	Mean x of on pixels in box	Mean y of on pixels in box	Mean x variance	Mean y variance
0	I	5	12	3	7	2	10	5	5	4
1	D	4	11	6	8	6	10	6	2	6
2	N	7	11	6	6	3	5	9	4	6
3	G	2	1	3	1	1	8	6	6	6
4	S	4	11	5	8	3	8	8	6	9
...	...	...	...	...	...	...	...	...	...	...
19994	D	2	2	3	3	2	7	7	7	6
19995	C	7	10	8	8	4	4	8	6	9
19996	T	6	9	6	7	5	6	11	3	7
19997	S	2	3	4	2	1	8	7	2	6
19998	A	4	9	6	6	2	9	5	3	1

19999 rows × 17 columns

In [6]:
```
# X = letter_dataset.drop(['Capital Letter'],axis=1)
# y = letter_dataset['Capital Letter']

# X = adult_dataset.drop(['Yearly Income'],axis=1)
# y = adult_dataset['Yearly Income']

letter_dict = {'A':0,'B':0,'C':0,'D':0,'E':0,'F':0,'G':0,'H':0,'I':0,'
J':0,'K':0,'L':0,'M':0,'N':1,'O':1,'P':1,'Q':1,'R':1,'S':1,'T':1,'U':1
,'V':1,'W':1,'X':1,'Y':1,'Z':1}

letter_dataset['Capital Letter'] = letter_dataset['Capital Letter'].ma
p(letter_dict)


y = letter_dataset[['Capital Letter']]
y = np.array(y)

# y = y.to_numpy()
# y = y.as_matrix(columns=y.columns[1:])


# y = y.ravel()
# letter_dataset = letter_dataset [(letter_dataset.astype(str) != ' ?'
).all(axis=1)]
# y.head()
# print(y.shape)

# np.isnan(X)
# np.isnan(y)

print(y)
```

```
[[0]
 [0]
 [1]
 ...
 [1]
 [1]
 [0]]
```

Trial 1

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

X = letter_dataset.drop(['Capital Letter'],axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y.ravel(),
                                                    train_size=0.2501,
                                                    random_state=12345
)

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: (5001, 16) and shape of target variable
y_train: (5001,)
Shape of input data X_test: (14998, 16) and shape of target variable
y_test: (14998,)
```

```
/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:12: DataConversionWarning: Data with input dtype int64
were all converted to float64 by StandardScaler.
  if sys.path[0] == '':
```

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]
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 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 89.68% | outer ACC 91.5
1%
outer fold 1/5 | tuning Logistic | inner ACC 72.50% | outer ACC 72.1
3%
outer fold 1/5 | tuning RandomForest | inner ACC 93.12% | outer ACC
93.81%
outer fold 2/5 | tuning KNN      | inner ACC 89.80% | outer ACC 89.5
1%
outer fold 2/5 | tuning Logistic | inner ACC 73.30% | outer ACC 71.3
3%
outer fold 2/5 | tuning RandomForest | inner ACC 92.85% | outer ACC
93.61%
outer fold 3/5 | tuning KNN      | inner ACC 89.68% | outer ACC 91.8
0%
outer fold 3/5 | tuning Logistic | inner ACC 72.31% | outer ACC 74.4
0%
outer fold 3/5 | tuning RandomForest | inner ACC 93.13% | outer ACC
93.50%
outer fold 4/5 | tuning KNN      | inner ACC 89.78% | outer ACC 92.0
0%
outer fold 4/5 | tuning Logistic | inner ACC 72.26% | outer ACC 73.2
0%
outer fold 4/5 | tuning RandomForest | inner ACC 92.68% | outer ACC
94.10%
outer fold 5/5 | tuning KNN      | inner ACC 89.61% | outer ACC 90.6
9%
outer fold 5/5 | tuning Logistic | inner ACC 72.99% | outer ACC 72.0
7%
outer fold 5/5 | tuning RandomForest | inner ACC 93.33% | outer ACC
93.59%
KNN      | outer CV acc. 91.10% +\- 0.912
RandomForest | outer CV acc. 93.72% +\- 0.214
Logistic | outer CV acc. 72.63% +\- 1.069

KNN best parameters {'classifier__n_neighbors': 20, 'classifier__wei
ghts': 'distance'}
RandomForest best parameters {'classifier__max_features': 12}
Logistic best parameters {'classifier__C': 100.0, 'classifier__penal
ty': 'l2'}
CPU times: user 41.7 s, sys: 2.17 s, total: 43.9 s
Wall time: 13min 7s
```

In [14]:
```
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 89.61% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 20, 'classifier__weight
s': 'distance'}
Training Accuracy: 98.14%
Test Accuracy: 91.67%
```

In [15]:
```
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 72.99% (average over CV test folds)
Best Parameters: {'classifier__C': 100.0, 'classifier__penalty': 'l2
'}
Training Accuracy: 72.89%
Test Accuracy: 72.36%
```

In [16]:
```
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 93.33% (average over CV test folds)
Best Parameters: {'classifier__max_features': 12}
Training Accuracy: 98.72%
Test Accuracy: 94.47%
```

In [ ]:

In [ ]:

In [ ]:

Trial 2

In [7]:
```
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

X = letter_dataset.drop(['Capital Letter'],axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y.ravel(),
                                                    train_size=0.2501,
                                                    random_state=5252)

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: (5001, 16) and shape of target variable
y_train: (5001,)
Shape of input data X_test: (14998, 16) and shape of target variable
y_test: (14998,)

/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:12: DataConversionWarning: Data with input dtype int64
were all converted to float64 by StandardScaler.
  if sys.path[0] == '':
```

In [8]:
```
# 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]
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 [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)

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 89.70% | outer ACC 91.6
1%
outer fold 1/5 | tuning Logistic | inner ACC 71.92% | outer ACC 73.1
3%
outer fold 1/5 | tuning RandomForest | inner ACC 93.33% | outer ACC
93.81%
outer fold 2/5 | tuning KNN      | inner ACC 89.50% | outer ACC 90.2
1%
outer fold 2/5 | tuning Logistic | inner ACC 72.08% | outer ACC 72.2
3%
outer fold 2/5 | tuning RandomForest | inner ACC 93.35% | outer ACC
93.61%
outer fold 3/5 | tuning KNN      | inner ACC 89.40% | outer ACC 91.9
0%
outer fold 3/5 | tuning Logistic | inner ACC 71.46% | outer ACC 72.9
0%
outer fold 3/5 | tuning RandomForest | inner ACC 93.30% | outer ACC
94.90%
outer fold 4/5 | tuning KNN      | inner ACC 89.30% | outer ACC 92.3
0%
outer fold 4/5 | tuning Logistic | inner ACC 71.91% | outer ACC 71.4
0%
outer fold 4/5 | tuning RandomForest | inner ACC 93.25% | outer ACC
94.10%
outer fold 5/5 | tuning KNN      | inner ACC 89.41% | outer ACC 90.0
9%
outer fold 5/5 | tuning Logistic | inner ACC 72.74% | outer ACC 70.0
7%
outer fold 5/5 | tuning RandomForest | inner ACC 93.18% | outer ACC
94.99%
KNN      | outer CV acc. 91.22% +\- 0.903
RandomForest | outer CV acc. 94.28% +\- 0.567
Logistic | outer CV acc. 71.94% +\- 1.114

KNN best parameters {'classifier__n_neighbors': 20, 'classifier__wei
ghts': 'distance'}
RandomForest best parameters {'classifier__max_features': 4}
Logistic best parameters {'classifier__C': 10.0, 'classifier__penalt
y': 'l2'}
CPU times: user 30.8 s, sys: 1.67 s, total: 32.5 s
Wall time: 7min 43s
```

In [10]:
```
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 89.41% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 20, 'classifier__weight
s': 'distance'}
Training Accuracy: 98.02%
Test Accuracy: 90.78%
```

In [11]:
```
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 72.74% (average over CV test folds)
Best Parameters: {'classifier__C': 10.0, 'classifier__penalty': 'l2'
}
Training Accuracy: 72.45%
Test Accuracy: 72.44%
```

In [12]:
```
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 93.18% (average over CV test folds)
Best Parameters: {'classifier__max_features': 4}
Training Accuracy: 99.00%
Test Accuracy: 94.22%
```

In [ ]:

In [ ]:

In [ ]:

Trial 3

In [21]:

```
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

X = letter_dataset.drop(['Capital Letter'],axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y.ravel(),
                                                    train_size=0.2501,
                                                    random_state=3769)

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_test: {}".format(X_test.shape, y_test.shape))
```

```
Shape of input data X_train: (7537, 16) and shape of target variable
y_train: (7537,)
Shape of input data X_test: (12462, 16) and shape of target variable
y_test: (12462,)
```

In [22]:
```
# 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]
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 [23]:
```
%%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 92.70% | outer ACC 92.4
4%
outer fold 1/5 | tuning Logistic | inner ACC 72.83% | outer ACC 70.0
9%
outer fold 1/5 | tuning RandomForest | inner ACC 94.71% | outer ACC
94.96%
outer fold 2/5 | tuning KNN      | inner ACC 92.72% | outer ACC 92.7
7%
outer fold 2/5 | tuning Logistic | inner ACC 72.48% | outer ACC 71.5
5%
outer fold 2/5 | tuning RandomForest | inner ACC 94.41% | outer ACC
94.83%
outer fold 3/5 | tuning KNN      | inner ACC 92.04% | outer ACC 93.6
3%
outer fold 3/5 | tuning Logistic | inner ACC 72.44% | outer ACC 72.4
6%
outer fold 3/5 | tuning RandomForest | inner ACC 94.58% | outer ACC
95.02%
outer fold 4/5 | tuning KNN      | inner ACC 92.09% | outer ACC 93.7
6%
outer fold 4/5 | tuning Logistic | inner ACC 72.12% | outer ACC 73.1
3%
outer fold 4/5 | tuning RandomForest | inner ACC 94.48% | outer ACC
94.96%
outer fold 5/5 | tuning KNN      | inner ACC 92.40% | outer ACC 93.9
6%
outer fold 5/5 | tuning Logistic | inner ACC 71.79% | outer ACC 73.6
6%
outer fold 5/5 | tuning RandomForest | inner ACC 94.58% | outer ACC
95.55%
KNN      | outer CV acc. 93.31% +\- 0.596
RandomForest | outer CV acc. 95.06% +\- 0.253
Logistic | outer CV acc. 72.18% +\- 1.257

KNN best parameters {'classifier__n_neighbors': 20, 'classifier__wei
ghts': 'distance'}
RandomForest best parameters {'classifier__max_features': 6}
Logistic best parameters {'classifier__C': 10.0, 'classifier__penalt
y': 'l1'}
CPU times: user 47.4 s, sys: 2.58 s, total: 50 s
Wall time: 17min 13s
```

In [24]:
```
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 92.40% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 20, 'classifier__weight
s': 'distance'}
Training Accuracy: 98.79%
Test Accuracy: 93.95%
```

In [25]:
```
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 71.79% (average over CV test folds)
Best Parameters: {'classifier__C': 10.0, 'classifier__penalty': 'l1'
}
Training Accuracy: 72.34%
Test Accuracy: 72.87%
```

In [26]:
```
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 94.58% (average over CV test folds)
Best Parameters: {'classifier__max_features': 6}
Training Accuracy: 99.11%
Test Accuracy: 96.02%
```

Trial 4 (Extra Credit)

In [31]:
```
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

X = letter_dataset.drop(['Capital Letter'],axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y.ravel(),
                                                    train_size=0.2501,
                                                    random_state=8773)

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: (5001, 16) and shape of target variable
y_train: (5001,)
Shape of input data X_test: (14998, 16) and shape of target variable
y_test: (14998,)
```

In [32]:
```
# 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]
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 [33]:
```
%%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 91.03% | outer ACC 90.3
1%
outer fold 1/5 | tuning Logistic | inner ACC 73.35% | outer ACC 72.5
3%
outer fold 1/5 | tuning RandomForest | inner ACC 93.33% | outer ACC
93.71%
outer fold 2/5 | tuning KNN      | inner ACC 89.45% | outer ACC 91.7
0%
outer fold 2/5 | tuning Logistic | inner ACC 72.61% | outer ACC 72.6
0%
outer fold 2/5 | tuning RandomForest | inner ACC 92.85% | outer ACC
95.10%
outer fold 3/5 | tuning KNN      | inner ACC 90.98% | outer ACC 91.1
0%
outer fold 3/5 | tuning Logistic | inner ACC 72.83% | outer ACC 73.6
0%
outer fold 3/5 | tuning RandomForest | inner ACC 92.93% | outer ACC
94.00%
outer fold 4/5 | tuning KNN      | inner ACC 90.15% | outer ACC 91.9
0%
outer fold 4/5 | tuning Logistic | inner ACC 72.63% | outer ACC 70.9
0%
outer fold 4/5 | tuning RandomForest | inner ACC 93.15% | outer ACC
94.50%
outer fold 5/5 | tuning KNN      | inner ACC 90.40% | outer ACC 91.8
0%
outer fold 5/5 | tuning Logistic | inner ACC 71.63% | outer ACC 74.5
0%
outer fold 5/5 | tuning RandomForest | inner ACC 93.35% | outer ACC
94.10%
KNN      | outer CV acc. 91.36% +\- 0.595
RandomForest | outer CV acc. 94.28% +\- 0.482
Logistic | outer CV acc. 72.83% +\- 1.204

KNN best parameters {'classifier__n_neighbors': 20, 'classifier__wei
ghts': 'distance'}
RandomForest best parameters {'classifier__max_features': 6}
Logistic best parameters {'classifier__C': 10.0, 'classifier__penalt
y': 'l1'}
CPU times: user 30.2 s, sys: 1.55 s, total: 31.8 s
Wall time: 8min 13s
```

In [34]:
```
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 90.40% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 20, 'classifier__weight
s': 'distance'}
Training Accuracy: 98.36%
Test Accuracy: 91.65%
```

In [35]:
```
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 71.63% (average over CV test folds)
Best Parameters: {'classifier__C': 10.0, 'classifier__penalty': 'l1'
}
Training Accuracy: 72.87%
Test Accuracy: 72.37%
```

In [36]:
```
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 93.35% (average over CV test folds)
Best Parameters: {'classifier__max_features': 6}
Training Accuracy: 98.82%
Test Accuracy: 94.39%
```

In [ ]:

In [ ]:
```
## here we are fitting the model with optimal parameters
clf_knn = KNeighborsClassifier(weights=optimal weight(distance or unif
orm?, n_neighbors=optimal number of neighbor here)
#fit the training data here
clf_knn.fit(X_train,y_train)
# get the accuracy of training data on classifier here
knn_accuracy = cross_val_score(clf_knn, X_train,y_train)
# print out training accuracy
print('KNN Train Accuracy',np.mean(knn_accuracy))
# see the testing accuracy here
print('KNN Test Accuracy',clf_knn.score(X_test, y_test))
# store the scores in dictionaries if you want
train_dict['KNN Train Accuracy'] = np.mean(knn_accuracy)
test_dict['KNN Test Accuracy'] = clf_knn.score(X_test, y_test)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: