Load Data

```
import pandas as pd
import numpy as np
from matplotlib import pyplot
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

```
#mount drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473189

Enter your authorization code:
··········
Mounted at /content/drive
```

```
#load cities dataset
df_cities = pd.read_csv("/content/drive/My Drive/Data Science with Python Spring2020/week6_26

####################################
# Summarize Data

# Descriptive statistics
# shape
```

```
print(df_cities.shape)
# head
print(df_cities.head(20))
# data type
print(df_cities.dtypes)
# descriptions
print(df_cities.describe())
```

```
(500, 34)
    Unnamed: 0 StateAbbr  ... TEETHLOST_CrudePrev                       Geolocation
0            1        CA  ...                 6.8  (38.67504943280, -121.147605753)
1            2        FL  ...                18.3  (27.90909077340, -82.7714203383)
2            3        CA  ...                 6.7  (37.87256787650, -122.274907975)
3            4        CA  ...                11.2  (38.29804246490, -122.301093331)
4            5        FL  ...                16.2  (26.15468783030, -80.2998411020)
5            6        FL  ...                14.1  (26.01273875340, -80.3384522664)
6            7        NJ  ...                26.1  (40.22372899810, -74.7639943311)
7            8        CO  ...                17.7  (38.27339572510, -104.612001218)
8            9        WI  ...                16.9  (42.72745994940, -87.8134530240)
9           10        WA  ...                13.6  (47.30385443250, -122.210810557)
10          11        TX  ...                10.1  (30.30686103420, -97.7554771245)
11          12        IL  ...                20.8  (42.37029555820, -87.8712595095)
12          13        MN  ...                10.3  (45.11120339160, -93.3505067942)
13          14        WA  ...                 9.7  (47.47605467520, -122.191153327)
14          15        OK  ...                17.7  (35.46756428800, -97.5137615524)
15          16        GA  ...                22.6  (33.36445615270, -82.0708396775)
16          17        MA  ...                26.0  (42.11549779990, -72.5395254143)
17          18        CA  ...                15.2  (35.35133028550, -119.029786003)
18          19        LA  ...                21.6  (30.20306799660, -93.2148796496)
```

Data Pre-processing

```
StateAbbr                  object
```

```
missing_values = df_cities.isnull().sum(axis=0)
missing_values
```

```
Unnamed: 0                0
StateAbbr                 0
PlaceName                 0
PlaceFIPS                 0
Population2010            0
ACCESS2_CrudePrev         0
ARTHRITIS_CrudePrev       0
BINGE_CrudePrev           0
```

```
#removing 47 missing values
#drop NaN rows
df_cities = df_cities.dropna()
df_cities.shape
```

```
(453, 34)
COLON_SCREEN_CrudePrev    0
```

```
#Set the random seed to “123”.
np.random.seed(123)
```

```
CSMOKING_CrudePrev        0
```

```
#Shuffle the rows in your dataset
#df_cities.apply(np.random.shuffle(df_cities.values),axis=0)
from sklearn.utils import shuffle
df_cities = shuffle(df_cities)
df_cities
```

	Unnamed: 0	StateAbbr	PlaceName	PlaceFIPS	Population2010	ACCESS2_CrudePrev	ART
52	53	IL	Rockford	1765000	152871	15.5	

Data Preparation

Recode

```
#Recode the target variable to be binary
#If greater than median, give it a “1”
#Otherwise, give it a “0”

# class distribution
# make a new target variable called 'class'
# returns true/false and multiplying by 1 gives integer
df_cities['Population2010'] = np.where(df_cities['Population2010'] > np.median(df_cities['Pop
print(df_cities.groupby('Population2010').size())
```

```
Population2010
0    227
1    226
dtype: int64
```

```
#predictor variables
df_x = df_cities.iloc[:,5:33]
print(df_x.shape)

#target variables
df_y = df_cities['Population2010']
```

```
(453, 28)
```

Data Partitioning

```
#partition data
validation_size = 0.20
seed = 123
X_train, X_validation, Y_train, Y_validation = train_test_split(df_x, df_y, test_size=validat
# let's look at the shape of these partitions
print("-----------------Shape of the Partitions------------------")
print("X_train.shape is ", X_train.shape)
print("Y_train.shape is ", Y_train.shape)
print("X_test.shape is ", X_validation.shape)
print("Y_test.shape is ", Y_validation.shape)
print("My original data was ", df_cities.shape)
```

```
print("My original data was ", df_cities.shape)
```

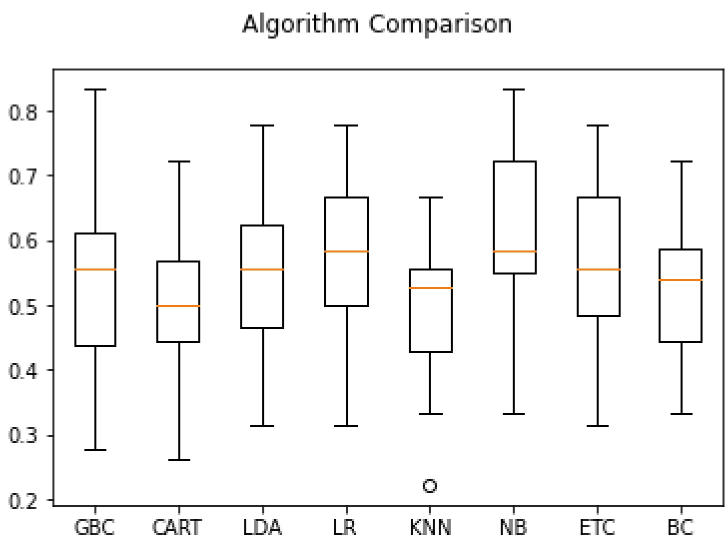
```
-----------------Shape of the Partitions------------------
X_train.shape is  (362, 28)
Y_train.shape is  (362,)
X_test.shape is  (91, 28)
Y_test.shape is  (91,)
My original data was  (453, 34)
```

Spot-checking and k-fold cross-validation

```
#SpotCheck for Classification Models

####################################################
# Spot-Check Algorithms
# Update the models
models = []
models.append(('GBC', GradientBoostingClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('LR', LogisticRegression(max_iter=1000000)))
models.append(('KNN', KNeighborsClassifier()))
models.append(('NB', GaussianNB()))
models.append(('ETC', ExtraTreesClassifier()))
models.append(('BC', BaggingClassifier()))


##################################################
# Use a 20-fold cross-validation on the data
# evaluate each model in turn
results = []   #accuracy of 10 folds
names = []

# store preds
from sklearn.model_selection import cross_val_predict
smPreds = []
for name, model in models:
  kfold = KFold(n_splits=20, random_state=seed, shuffle=True)
  # store the metrics
  cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accuracy')
  results.append(cv_results)    #stores accuracy of 10 folds for each model
  names.append(name)
  msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
  print(msg)
```

```
GBC: 0.530702 (0.137288)
CART: 0.520175 (0.110945)
```

```
#####################################
# Compare Algorithms
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
# picking top three models that worked the best-Logistic Regression, Gaussian NB, Extra Tree
```



Pipeline

```
LogisticRegression().get_params()
```

```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

```
ExtraTreesClassifier().get_params()
```

```
{'bootstrap': False,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

```
GaussianNB().get_params()
```

```
{'priors': None, 'var_smoothing': 1e-09}
```

```
#construct pipelines
#Use standard scaling and PCA as pre-processing
#LogisticRegression
pipe_lr_pca = Pipeline([('scl', StandardScaler()),
      ('pca', PCA(n_components=2)),
      ('lr', LogisticRegression(random_state=123))])

#ExtraTreesClassifier
pipe_etc_pca = Pipeline([('scl', StandardScaler()),
      ('pca', PCA(n_components=2)),
      ('etc', ExtraTreesClassifier())])

#GaussianNB
pipe_gnb_pca = Pipeline([('scl', StandardScaler()),
      ('pca', PCA(n_components=2)),
      ('gnb', GaussianNB())])


param_range = [1, 2, 3]
param_range_fl = [1.0, 0.5, 0.1]

# Set grid search params
#LogisticRegression
grid_params_lr = [{'lr__penalty': ['l1', 'l2'],
                   'lr__C': param_range_fl,
                   'lr__solver': ['liblinear']}]

#ExtraTreesClassifier
grid_params_etc = [{'etc__criterion': ['gini', 'entropy'],
                    'etc__min_impurity_decrease': param_range
```

```
                    'etc__min_impurity_decrease': param_range,
                    'etc__min_samples_leaf': param_range}]

#GaussianNB
grid_params_gnb = [{'gnb__var_smoothing': param_range}]


# Construct grid searches
jobs = -1

gs_lr = GridSearchCV(estimator=pipe_lr_pca,
      param_grid=grid_params_lr,
      scoring='accuracy',
      cv=10)

gs_etc = GridSearchCV(estimator=pipe_etc_pca,
      param_grid=grid_params_etc,
      scoring='accuracy',
      cv=10,
      n_jobs=jobs)

gs_gnb = GridSearchCV(estimator=pipe_gnb_pca,
          param_grid=grid_params_gnb,
          scoring='accuracy',
          cv=10)


# List of pipelines for ease of iteration
grids = [gs_lr, gs_etc, gs_gnb]

# Dictionary of pipelines and classifier types for ease of reference
grid_dict = {0: 'Logistic Regression', 1: 'Extra Trees Classifier' , 2: 'Gaussian Naive Bayes


# Fit the grid search objects
print('Performing model optimizations...')
best_acc = 0.0
best_clf = 0
best_gs = ''
for idx, gs in enumerate(grids):
  print('\nEstimator: %s' % grid_dict[idx])

  # Fit grid search
  gs.fit(X_train, Y_train)

  # Best params
  print('Best params: %s' % gs.best_params_)

  # Best training data accuracy
  print('Best training accuracy: %.3f' % gs.best_score_)

  # Predict on test data with best params
  y_pred = gs.predict(X_validation)
```

```
  # Test data accuracy of model with best params(compare actual and predicted)
  print('Test set accuracy score for best params: %.3f ' % accuracy_score(Y_validation, y_pre

  # Track best (highest test accuracy) model
  if accuracy_score(Y_validation, y_pred) > best_acc:
    best_acc = accuracy_score(Y_validation, y_pred)
    best_gs = gs
    best_clf = idx
print('\nClassifier with best test set accuracy: %s' % grid_dict[best_clf])
```

```
Performing model optimizations...

Estimator: Logistic Regression
Best params: {'lr__C': 0.5, 'lr__penalty': 'l2', 'lr__solver': 'liblinear'}
Best training accuracy: 0.594
Test set accuracy score for best params: 0.549

Estimator: Extra Trees Classifier
Best params: {'etc__criterion': 'gini', 'etc__min_impurity_decrease': 1, 'etc__min_sampl
Best training accuracy: 0.505
Test set accuracy score for best params: 0.473

Estimator: Gaussian Naive Bayes
Best params: {'gnb__var_smoothing': 1}
Best training accuracy: 0.550
Test set accuracy score for best params: 0.538

Classifier with best test set accuracy: Logistic Regression
```

```
#Use the best fitting model and architecture to predict the holdout data
############################################
# Make predictions on validation dataset
#For the algorithm that performs the best, run the model on all training data and predict the
model1 = LogisticRegression(C = 0.5, penalty = 'l2', solver = 'liblinear')
model1.fit(X_train, Y_train)
predictions = model1.predict(X_validation)


#Accuracy metrics
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.5714285714285714
[[25 23]
 [16 27]]
              precision    recall  f1-score   support

           0       0.61      0.52      0.56        48
           1       0.54      0.63      0.58        43

    accuracy                           0.57        91
   macro avg       0.57      0.57      0.57        91
weighted avg       0.58      0.57      0.57        91
```