```
import pandas as pd
     import numpy as np
     from sklearn.svm import SVC
     import matplotlib.pyplot as plt
     from sklearn.preprocessing import StandardScaler
 6
7
     #Importing the Data from the give files
     train set = pd.read csv("tweets-train-data.csv", sep= ",", header = None, names =
9
     ["Tweet", "Date", "Retweets", "Likes", "Location"])
     test set = pd.read csv("tweets-test-data.csv", sep= ",", header = None, names =
10
     ["Tweet", "Date", "Retweets", "Likes", "Location"])
     train targets = pd.read csv("tweets-train-targets.csv", sep= ",", header = None,
11
     names = ["Author"])
     test targets = pd.read csv("tweets-test-targets.csv", sep= ",", header = None, names
12
     = ["Author"])
1.3
14
     #Formating the data
15
     training = pd.concat([train_set, train_targets], axis = 1)
16
     training = training[training["Date"].astype(str).str.startswith("2016")]
17
     training["Likes"] = training["Likes"].astype(float)
18
     training["Date"], training["Time"] = training["Date"].str.split("T").str
19
    training["Date"].astype("datetime64")
20
    print(training)
21
22
     #Likes information
23
    Trump = training[training["Author"] == "DT"]
     Clinton = training[training["Author"] == "HC"]
24
25
     Clinton_likes = Clinton["Likes"]
26
     Trump likes = Trump["Likes"]
27
    print(Trump likes)
28
    print(Clinton likes)
29
30
     #Ploting the retweets
31
     Clinton rt = Clinton["Retweets"]
     Trump r\bar{t} = Trump["Retweets"]
32
33
34
    plt.plot(Clinton_rt, "c")
35
    plt.plot(Trump_rt, "r")
36
    plt.xlabel("Tweet id")
37
    plt.ylabel("Number of rts")
    plt.legend(("HC", "DT"))
38
39
    plt.grid(True)
40
41
    plt.show()
42
    plt.close()
43
44
     #Clinton most popular tweet (rts)
45
    print(max(Clinton rt))
46
    top Clinton rt = training.loc[training["Retweets"]==max(Clinton rt)]
47
    print(top Clinton rt)
48
49
     #Trump most popular tweet (rts)
50
    print(max(Trump rt))
51
     top Trump rt = training.loc[training["Retweets"] == max(Trump rt)]
52
    print(top Trump rt)
53
54
     #Clinton most liked tweet
55
     print(max(Clinton likes))
    top Clinton likes = training.loc[training["Likes"] == max(Clinton likes)]
56
57
    print(top Clinton likes)
58
59
    #Trump most liked tweet
60
    print(max(Trump likes))
61
    top Trump likes = training.loc[training["Likes"] == max(Trump likes)]
62
    print(top_Trump_likes)
63
     #Formating the testing data
64
     testing = pd.concat([test_set, test_targets], axis = 1)
65
     testing = testing[testing["Date"].astype(str).str.startswith("2016")]
66
67
     testing["Likes"] = testing["Likes"].astype(float)
     testing["Date"], testing["Time"] = testing["Date"].str.split("T").str
```

```
69
      testing["Date"].astype("datetime64")
 70
      print(testing)
 71
 72
      #Extracting the features
 73
      train features = training["Author"]
 74
      train rts = training["Retweets"].values.reshape(-1,1)
 75
      test features = testing["Author"]
      test rts = testing["Retweets"].values.reshape(-1,1)
 76
 77
 78
 79
      #Normalize the tweets length
      a = np.mean(training["Tweet"].apply(len))
 80
      b = np.mean(testing["Tweet"].apply(len))
 81
 82
      training["word length"] = training["Tweet"].apply(len)-a
 83
      testing["word length"] = testing["Tweet"].apply(len)-b
 84
 8.5
 86
      #Re-assign
 87
      Trump = training[training["Author"] == "DT"]
 88
      Clinton = training[training["Author"] == "HC"]
 89
      Trump_average_length = Trump["word length"]
 90
 91
      Clinton_average_length = Clinton["word_length"]
 92
 93
      #Spaces count
 94
     training["space count"] = training["Tweet"].str.count(" ")
 95
      training["a count"] = training["Tweet"].str.count("a")
 96
      testing["space count"] = testing["Tweet"].str.count(" ")
 97
      testing["a count"] = testing["Tweet"].str.count("a")
 98
 99
      #Taking logarithms (monotonic function)
      training["log likes"] = np.log(training["Likes"])
100
      training["log rts"] = np.log(training["Retweets"])
101
      testing["log likes"] = np.log(testing["Likes"])
102
      testing["log rts"] = np.log(testing["Retweets"])
103
104
105
      #Dot count
106
      training["dot count"] = training["Tweet"].str.count(".")
107
      testing["dot count"] = testing["Tweet"].str.count(".")
108
109
      #Hours
      training["hour"] = training["Time"].str[0:2]
110
111
      pd.to numeric(training["hour"])
112
      testing["hour"] = testing["Time"].str[0:2]
113
      pd.to numeric(testing["hour"])
114
115
      #Months
116
      training["month"] = training["Date"].str[5:7]
117
      pd.to numeric(training["month"])
      testing["month"] = testing["Date"].str[5:7]
118
119
      pd.to numeric(testing["month"])
120
121
      scale = StandardScaler()
122
      log_training_matrix = training[["log_likes", "log_rts", "word length", "a count",
123
      "space_count", "dot_count", "month", "hour", "Retweets"]]
log_testing_matrix = testing[["log_likes", "log_rts", "word_length", "a_count",
124
      "space count", "dot count", "month", "hour", "Retweets"]]
125
126
      transformed x = \text{scale.fit transform(log training matrix, train features)}
127
      transformed y = scale.fit transform(log testing matrix, test features)
128
129
      #Training using SVC
130
131
      clf = SVC(C=10, kernel = "rbf", gamma = 0.02)
132
      clf.fit(transformed x, train features)
133
134
      y predict = clf.predict(transformed y)
135
136
      from sklearn import metrics
137
      report = metrics.classification report(test_features, y_predict)
138
```

```
139
      metrics.accuracy score(test features, y predict)
140
141
      #K-fold cross Validation
142
      try:
143
          from sklearn.model selection import KFold, cross val score
144
          legacy = False
145
      except ImportError:
146
          from sklearn.model selection import KFold, cross val score
147
          legacy = True
148
149
150
      if legacy:
          kf = KFold(len(train_features), n_folds = 3, shuffle = True, random state = 42)
151
152
      else:
153
          kf = KFold(n splits = 3, shuffle = True, random state = 42)
154
155
      gamma values = [0.01, 0.001, 0.0001, 0.00001]
156
      accuracy_scores = []
157
158
      #K-Fold cross validation algorithm:
159
     #-Train predictor
160
      #-Compute score
161
      for gamma in gamma_values:
162
163
          #Train classifier
164
          clf = SVC(C=10, kernel = "rbf", gamma = gamma)
165
166
          #Score
167
          if legacy:
168
              scores = cross val score(clf, transformed x, train features, cv = kf,
              scoring="accuracy")
169
          else:
170
              scores = cross val score(clf, transformed x, train features, cv =
              kf.split(transformed x), scoring = "accuracy")
171
172
          #Compute Average score
173
          accuracy score = scores.mean()
174
          accuracy scores.append(accuracy score)
175
176
      #Best value of gamma
177
      best index = np.array(accuracy scores).argmax()
178
      best gamma = gamma values[best index]
179
180
      #Train with the selected gamma
      clf = SVC(C=10, kernel = "rbf", gamma = best gamma)
181
182
      clf.fit(transformed x, train features)
183
184
      #Evaluate on the test set
185
      y predict = clf.predict(transformed y)
186
      accuracy = metrics.accuracy score(test features, y predict)
187
188
      print(accuracy)
189
190
      print(best gamma)
191
192
      from sklearn.model selection import learning curve
193
194
     plt.figure(2)
195
     plt.title("Learning curve")
196
     plt.xlabel("Training examples")
197
     plt.ylabel("Score")
198
      plt.grid(True)
199
200
      clf = SVC(C=10, kernel = "rbf", gamma = best gamma)
201
202
203
      train_sizes, train_scores, val_scores = learning_curve(clf, transformed_x,
      train features, scoring = "accuracy")
204
      train scores_mean = np.mean(train_scores, axis = 1)
205
      train scores_std = np.std(train_scores, axis = 1)
206
      val scores_mean = np.mean(val_scores, axis = 1)
207
```

```
208
      val scores std = np.std(val scores, axis = 1)
209
210
      #Training scores: mean
211
      plt.plot(train sizes, train scores mean, '*', color = "r", label = "Training Score")
212
      #Training scores: std
213
      plt.fill between (train sizes, train scores mean - train scores std,
      train scores mean + train scores std, alpha = 0.1, color = "r")
214
      #Validation scores: mean
      plt.plot(train_sizes, val_scores_mean,'*', color = "c", label = "Cross Validation
215
      Scores")
216
      #Validation scores: std
217
      plt.fill between(train sizes, val scores mean - val scores std, val scores mean +
      val scores std, alpha = 0.1, color = "c")
218
219
     plt.ylim(0.05, 1.3)
220
     plt.legend()
221
     plt.show()
222
     plt.close()
223
224
      try:
225
          from sklearn.model_selection import GridSearchCV
226
      except ImportError:
227
          from sklearn.grid_search import GridSearchCV
228
      possible_parameters = {
229
              C' : [10,4,3,2],
230
              'gamma': [0.01,0.001,0.0001,0.00001]
231
232
      svc = SVC(kernel = "rbf")
233
234
235
      #The GridSearch is a classifier. Hence, we try to fit it with the training data,
236
      #and then use it for prediction
237
      clf = GridSearchCV(svc, possible parameters, n jobs = 4, cv = 3)
238
      clf.fit(transformed x, train features)
239
240
      y predidct = clf.predict(transformed y)
      accuracy = metrics.accuracy_score(test features, y predict)
241
242
243
      #Training phase
244
      clf = SVC(C=10, kernel = "rbf", gamma = 0.01)
245
      clf.fit(transformed x, train features)
246
      #Prediction phase
247
      y predict = clf.predict(transformed y)
248
249
      report = metrics.classification report(test features, y predict)
250
      metrics.accuracy score(test features, y predict)
251
252
253
      print(report)
254
255
      #Performance parameters
256
      from sklearn.model selection import cross val score
257
258
      accuracy = cross val score(clf, transformed x, train features, cv = 10, scoring =
      "accuracy")
259
260
261
      labels = train features.map(lambda x: 1 if x == "HC" else 0).values
262
263
      precision = cross val score(clf, transformed x, labels, cv = 10, scoring =
      "precision")
264
265
      recall = cross val score(clf, transformed x, labels, cv = 10, scoring = "recall")
266
267
      f1 = cross val score(clf, transformed x, labels, cv=10, scoring = "f1")
268
269
      print("Avg Precision: {}".format(round(precision.mean(), 3)))
      print("Avg Recall: {}".format(round(recall.mean(), 3)))
270
      print("Avg Accuracy: {}".format(round(accuracy.mean(), 3)))
271
272
      print("Avg f1: {}".format(round(f1.mean(), 3)))
273
274
```

```
275
      #Retweets plot
     plt.hist(training["Retweets"][training["Author"]=="HC"], color = "c")
276
277
     plt.hist(training["Retweets"][training["Author"]=="DT"], color = "r")
278
     plt.title("Retweets")
279
     plt.legend()
280
    plt.grid(True)
281
     plt.show()
282
     plt.close()
283
284
285
      #Log Retweets plot
286
    plt.hist(training["log rts"][training["Author"]=="HC"], color = "c")
287
     plt.hist(training["log rts"][training["Author"]=="DT"], color = "r")
288
    plt.title("Log Retweets")
289
    plt.legend()
290
     plt.grid(True)
291
     plt.show()
292
     plt.close()
293
294
     #Likes plot
    plt.hist(training["Likes"][training["Author"]=="HC"], color = "c")
295
296
     plt.hist(training["Likes"][training["Author"]=="DT"], color = "r")
297
     plt.title("Likes")
298
     plt.legend()
299
     plt.grid(True)
300
     plt.show()
301
     plt.close()
302
303
     #Log likes plot
304
    plt.hist(training["log likes"][training["Author"]=="HC"], color = "c")
     plt.hist(training["log likes"][training["Author"]=="DT"], color = "r")
305
306
     plt.title("Log Likes")
307
     plt.legend()
308
     plt.grid(True)
309
     plt.show()
310
     plt.close()
311
```