```python
import pandas as pd
import numpy as np
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler


#Importing the Data from the give files
train_set = pd.read_csv("tweets-train-data.csv", sep= ",", header = None, names = ["Tweet", "Date", "Retweets", "Likes", "Location"])
test_set = pd.read_csv("tweets-test-data.csv", sep= ",", header = None, names = ["Tweet", "Date", "Retweets", "Likes", "Location"])
train_targets = pd.read_csv("tweets-train-targets.csv", sep= ",", header = None, names = ["Author"])
test_targets = pd.read_csv("tweets-test-targets.csv", sep= ",", header = None, names = ["Author"])

#Formating the data
training = pd.concat([train_set, train_targets], axis = 1)
training = training[training["Date"].astype(str).str.startswith("2016")]
training["Likes"] = training["Likes"].astype(float)
training["Date"], training["Time"] = training["Date"].str.split("T").str
training["Date"].astype("datetime64")
print(training)

#Likes information
Trump = training[training["Author"] == "DT"]
Clinton = training[training["Author"] == "HC"]
Clinton_likes = Clinton["Likes"]
Trump_likes = Trump["Likes"]
print(Trump_likes)
print(Clinton_likes)

#Ploting the retweets
Clinton_rt = Clinton["Retweets"]
Trump_rt = Trump["Retweets"]

plt.plot(Clinton_rt, "c")
plt.plot(Trump_rt, "r")
plt.xlabel("Tweet id")
plt.ylabel("Number of rts")
plt.legend(("HC", "DT"))
plt.grid(True)

plt.show()
plt.close()

#Clinton most popular tweet (rts)
print(max(Clinton_rt))
top_Clinton_rt = training.loc[training["Retweets"]==max(Clinton_rt)]
print(top_Clinton_rt)

#Trump most popular tweet (rts)
print(max(Trump_rt))
top_Trump_rt = training.loc[training["Retweets"]==max(Trump_rt)]
print(top_Trump_rt)

#Clinton most liked tweet
print(max(Clinton_likes))
top_Clinton_likes = training.loc[training["Likes"]==max(Clinton_likes)]
print(top_Clinton_likes)

#Trump most liked tweet
print(max(Trump_likes))
top_Trump_likes = training.loc[training["Likes"]==max(Trump_likes)]
print(top_Trump_likes)

#Formating the testing data
testing = pd.concat([test_set, test_targets], axis = 1)
testing = testing[testing["Date"].astype(str).str.startswith("2016")]
testing["Likes"] = testing["Likes"].astype(float)
testing["Date"], testing["Time"] = testing["Date"].str.split("T").str
```

```python
testing["Date"].astype("datetime64")
print(testing)

#Extracting the features
train_features = training["Author"]
train_rts = training["Retweets"].values.reshape(-1,1)
test_features = testing["Author"]
test_rts = testing["Retweets"].values.reshape(-1,1)


#Normalize the tweets length
a = np.mean(training["Tweet"].apply(len))
b = np.mean(testing["Tweet"].apply(len))

training["word_length"] = training["Tweet"].apply(len)-a
testing["word_length"] = testing["Tweet"].apply(len)-b

#Re-assign
Trump = training[training["Author"] == "DT"]
Clinton = training[training["Author"] == "HC"]

Trump_average_length = Trump["word_length"]
Clinton_average_length = Clinton["word_length"]

#Spaces count
training["space_count"] = training["Tweet"].str.count(" ")
training["a_count"] = training["Tweet"].str.count("a")
testing["space_count"] = testing["Tweet"].str.count(" ")
testing["a_count"] = testing["Tweet"].str.count("a")

#Taking logarithms (monotonic function)
training["log_likes"] = np.log(training["Likes"])
training["log_rts"] = np.log(training["Retweets"])
testing["log_likes"] = np.log(testing["Likes"])
testing["log_rts"] = np.log(testing["Retweets"])

#Dot count
training["dot_count"] = training["Tweet"].str.count(".")
testing["dot_count"] = testing["Tweet"].str.count(".")

#Hours
training["hour"] = training["Time"].str[0:2]
pd.to_numeric(training["hour"])
testing["hour"] = testing["Time"].str[0:2]
pd.to_numeric(testing["hour"])

#Months
training["month"] = training["Date"].str[5:7]
pd.to_numeric(training["month"])
testing["month"] = testing["Date"].str[5:7]
pd.to_numeric(testing["month"])

scale = StandardScaler()

log_training_matrix = training[["log_likes", "log_rts", "word_length", "a_count", "space_count", "dot_count", "month", "hour", "Retweets"]]
log_testing_matrix = testing[["log_likes", "log_rts", "word_length", "a_count", "space_count", "dot_count", "month", "hour", "Retweets"]]

transformed_x = scale.fit_transform(log_training_matrix, train_features)
transformed_y = scale.fit_transform(log_testing_matrix, test_features)

#Training using SVC

clf = SVC(C=10, kernel = "rbf", gamma = 0.02)
clf.fit(transformed_x, train_features)

y_predict = clf.predict(transformed_y)

from sklearn import metrics

report = metrics.classification_report(test_features, y_predict)
```

```python
metrics.accuracy_score(test_features, y_predict)

#K-fold cross Validation
try:
    from sklearn.model_selection import KFold, cross_val_score
    legacy = False
except ImportError:
    from sklearn.model_selection import KFold, cross_val_score
    legacy = True

#K=3
if legacy:
    kf = KFold(len(train_features), n_folds = 3, shuffle = True, random_state = 42)
else:
    kf = KFold(n_splits = 3, shuffle = True, random_state = 42)

gamma_values = [0.01, 0.001, 0.0001, 0.00001]
accuracy_scores = []

#K-Fold cross validation algorithm:
#-Train predictor
#-Compute score
for gamma in gamma_values:

    #Train classifier
    clf = SVC(C=10, kernel = "rbf", gamma = gamma)

    #Score
    if legacy:
        scores = cross_val_score(clf, transformed_x, train_features, cv = kf, scoring="accuracy")
    else:
        scores = cross_val_score(clf, transformed_x, train_features, cv = kf.split(transformed_x), scoring = "accuracy")

    #Compute Average score
    accuracy_score = scores.mean()
    accuracy_scores.append(accuracy_score)

#Best value of gamma
best_index = np.array(accuracy_scores).argmax()
best_gamma = gamma_values[best_index]

#Train with the selected gamma
clf = SVC(C=10, kernel = "rbf", gamma = best_gamma)
clf.fit(transformed_x, train_features)

#Evaluate on the test set
y_predict = clf.predict(transformed_y)
accuracy = metrics.accuracy_score(test_features, y_predict)

print(accuracy)

print(best_gamma)

from sklearn.model_selection import learning_curve

plt.figure(2)
plt.title("Learning curve")
plt.xlabel("Training examples")
plt.ylabel("Score")
plt.grid(True)


clf = SVC(C=10, kernel = "rbf", gamma = best_gamma)

train_sizes, train_scores, val_scores = learning_curve(clf, transformed_x, train_features, scoring = "accuracy")

train_scores_mean = np.mean(train_scores, axis = 1)
train_scores_std = np.std(train_scores, axis = 1)
val_scores_mean = np.mean(val_scores, axis = 1)
```

```
val_scores_std = np.std(val_scores, axis = 1)

#Training scores: mean
plt.plot(train_sizes, train_scores_mean, '*' ,color = "r", label = "Training Score")
#Training scores: std
plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean + train_scores_std, alpha = 0.1, color = "r")
#Validation scores: mean
plt.plot(train_sizes, val_scores_mean,'*', color = "c", label = "Cross Validation Scores")
#Validation scores: std
plt.fill_between(train_sizes, val_scores_mean - val_scores_std, val_scores_mean + val_scores_std, alpha = 0.1, color = "c")

plt.ylim(0.05, 1.3)
plt.legend()
plt.show()
plt.close()

try:
    from sklearn.model_selection import GridSearchCV
except ImportError:
    from sklearn.grid_search import GridSearchCV
possible_parameters = {
        'C' : [10,4,3,2],
        'gamma' : [0.01,0.001,0.0001,0.00001]
         }

svc = SVC(kernel = "rbf")

#The GridSearch is a classifier. Hence, we try to fit it with the training data,
#and then use it for prediction
clf = GridSearchCV(svc, possible_parameters, n_jobs = 4, cv = 3)
clf.fit(transformed_x, train_features)

y_predidct = clf.predict(transformed_y)
accuracy = metrics.accuracy_score(test_features, y_predict)

#Training phase
clf = SVC(C=10, kernel = "rbf", gamma = 0.01)
clf.fit(transformed_x, train_features)
#Prediction phase
y_predict = clf.predict(transformed_y)

report = metrics.classification_report(test_features, y_predict)
metrics.accuracy_score(test_features, y_predict)


print(report)

#Performance parameters
from sklearn.model_selection import cross_val_score

accuracy = cross_val_score(clf, transformed_x, train_features, cv = 10, scoring = "accuracy")

#Labels
labels = train_features.map(lambda x: 1 if x == "HC" else 0).values

precision = cross_val_score(clf, transformed_x, labels, cv = 10, scoring = "precision")

recall = cross_val_score(clf, transformed_x, labels, cv = 10, scoring = "recall")

f1 = cross_val_score(clf, transformed_x, labels, cv=10, scoring = "f1")

print("Avg Precision: {}".format(round(precision.mean(), 3)))
print("Avg Recall: {}".format(round(recall.mean(), 3)))
print("Avg Accuracy: {}".format(round(accuracy.mean(), 3)))
print("Avg f1: {}".format(round(f1.mean(), 3)))
```

```
#Retweets plot
plt.hist(training["Retweets"][training["Author"]=="HC"], color = "c")
plt.hist(training["Retweets"][training["Author"]=="DT"], color = "r")
plt.title("Retweets")
plt.legend()
plt.grid(True)
plt.show()
plt.close()


#Log Retweets plot
plt.hist(training["log_rts"][training["Author"]=="HC"], color = "c")
plt.hist(training["log_rts"][training["Author"]=="DT"], color = "r")
plt.title("Log Retweets")
plt.legend()
plt.grid(True)
plt.show()
plt.close()

#Likes plot
plt.hist(training["Likes"][training["Author"]=="HC"], color = "c")
plt.hist(training["Likes"][training["Author"]=="DT"], color = "r")
plt.title("Likes")
plt.legend()
plt.grid(True)
plt.show()
plt.close()

#Log likes plot
plt.hist(training["log_likes"][training["Author"]=="HC"], color = "c")
plt.hist(training["log_likes"][training["Author"]=="DT"], color = "r")
plt.title("Log Likes")
plt.legend()
plt.grid(True)
plt.show()
plt.close()

```