Project Report

- sentiment analysis with Twitter

Jiao Qu A20386614, Yuan-An Liu A20375099, Zhenyu Zhang A20287371

1. Introduction

To figure out how much people like your product is never an easy job. Social networks often become the most popular tool to analyze. The ultimate goal for this project is to analyze the mood of people who tweets about {$keyword}. At here, we will use "Chicago bear" as an example to demonstrate.

Related work:

Comparing our result to the research: Sentiment Analysis of Twitter Data [1] In this research, they use a unigram model as their baseline. Researchers report state-of-the-art performance for sentiment analysis on Twitter data using a unigram model (Go et al., 2009; Pak and Paroubek, 2010) with the tree kernel they have designed.

2. Design

Data:

All the data are gathering from twitter using Twitter API [4]. The data will contain: users, friends, and tweets. We collected the train and test data from a Stanford University research [2], [3] for testing our classifying method's accuracy.
The data is a CSV with emoji removed. Data file format has 6 fields:
0 - the polarity of the tweet (0 = negative, 4 = positive)
1 - the id of the tweet (2087)
2 - the date of the tweet (Sat May 16 23:58:44 UTC 2009)
3 - the query (lyx). If there is no query, then this value is NO_QUERY.
4 - the user that tweeted (robotickilldozr)
5 - the text of the tweet (Lyx is cool)

Appoarch:

We will collect users who tweet about "Chicago bear" and get their friends and tweets of each users by using Twitter request. So we will collect users, friends, and tweets. Dump the data into user, friend, and tweet files by using pickle. The application flow is shown in Fig. 2.1.

First, we try to collect 10 users tweeting about "Chicago bear" and get 20 friends and tweets of each users by using Twitter request [4].
So we end up collecting 10 users, 200 friends and 2000 tweets. Dump the data into users.txt, friends.txt, and tweets.txt by using pickle.

Second, we load the data (users and their friends) from previous step. Create a Networkx graph with the data. Then we cluster users into communities by using Girvan-Newman.
In order to implement the algorithm we also use betweenness_centrality [App. 3] from the Networkx library [10]. After clustering, dump the result into sum.txt by using append pickle.

Third, we load the data (users and their tweets) from the first step. Also, we load the train data which was manually pre-labeled tweets and store it by using Pandas. Then use the TfidfVectorize, SVM, and

classification_report from SKlearn [5]. Then we use TfidfVectorizer [App. 4], svm with 3 different classification methods: RDF-kernel, Linear-kernel, Poly-kernel with the degree of 3 to do the fit. After classifying, dump the result into sum.txt by using append pickle.
Last, we print the results by loading sum.txt

↑ Fig. 2.1 Application Flow

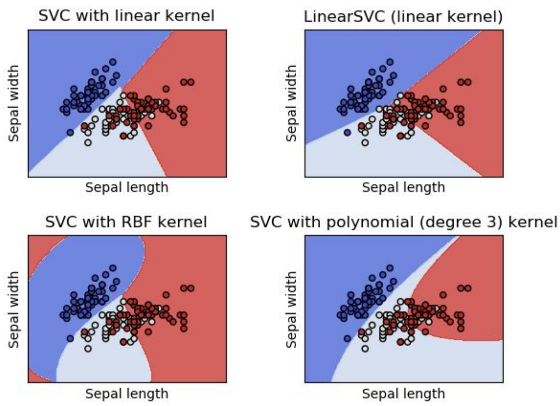
3. Usage and testing for accuracy:

Prerequisite:

- Python Version >= 3.5
- Packages:
  TwitterAPI, SciKit-Learn, Pandas, NetworkX, NumPy, SciPy

To run the code, simply run the shell script: Run.sh [App. 1]. The script will check for python version first. It will make sure the system has python version higher than 3.5 and then run all the python files.

We also have written and seperate file: “classify_testing.py [App. 5]” to test the accuracy for different classification methods with our test dataset which contains 500 tweets . Fig. 3.1 Gives an example of each method. [6]



↑ Fig. 3.1 Example of all the kernels

The source code will perform multiple classification methods with pre-labeled train and test data. Then it will print out the accuracies for each method.

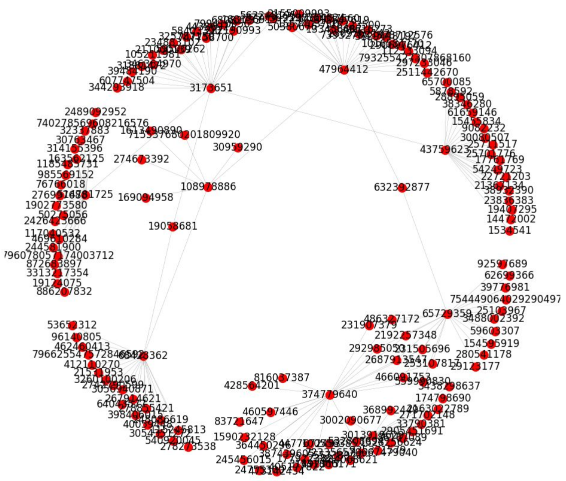
4. Result

Fig. 4.1 Shows the result of clustering users into different communities using Girvan-Newman algorithm. As the result, there are 3 communities.

```
> python cluster.py                                        [14:25:08]
graph has 162 nodes and 167 edges
[<networkx.classes.graph.Graph object at 0x1140c3d30>,
<networkx.classes.graph.Graph object at 0x1140c3d68>,
<networkx.classes.graph.Graph object at 0x114660e48>]
```

↑ Fig. 4.1 Clustering result

For the reference, we also save the network graph. In Fig 4.2, each node (red dot) represents a user. Each edge (gray line) represents following.



↑ Fig. 4.2 Users network

Fig 4.3 Shows the result comparing different classification methods.

```
> python classify_testing.py                               [14:22:26]
Results for SVC(kernel=rbf)
  'precision', 'predicted', average, warn_for)
             precision    recall  f1-score   support

          0       0.50      1.00      0.67       101
          4       0.00      0.00      0.00       100

avg / total       0.25      0.50      0.34       201

Results for SVC(kernel=linear)
Training time: 0.003126s; Prediction time: 0.001882s
             precision    recall  f1-score   support

          0       0.88      0.73      0.80       101
          4       0.77      0.90      0.83       100

avg / total       0.83      0.82      0.81       201

Results for SVC(kernel=poly)
Training time: 0.004142s; Prediction time: 0.002478s
             precision    recall  f1-score   support

          0       0.50      1.00      0.67       101
          4       0.00      0.00      0.00       100

avg / total       0.25      0.50      0.34       201
```

↑ Fig. 4.3 Classifications Report

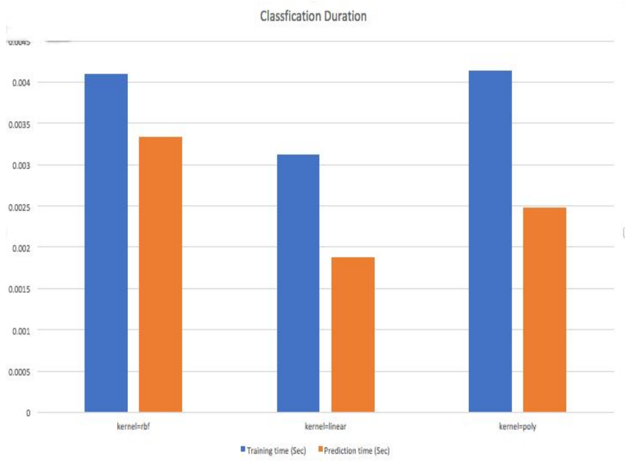
The f1-score shows the harmonic mean of precision and recall. According to, sklearn.metrics' document [8]: "The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:"

```
F1 = 2 * (precision * recall) /
(precision + recall)
```

The scores corresponding to every class will represent the accuracy of the classifier in classifying the data points in that particular class compared to all other classes. In classes, 4 is Positive, 0 means Negative.

The support is the number of samples of the true response that lie in that class.

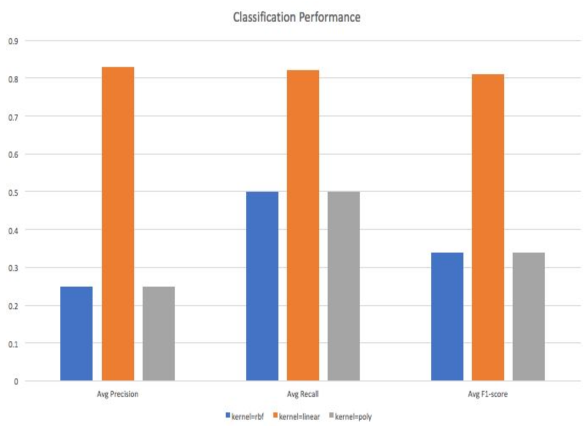
Fig 4.4 Shows the time elapsed for performing different classification methods.



↑ Fig. 4.4 Classification Duration

The graph indicates that "Linear" has is the fastest comparing to other two methods.

Fig 4.5 Shows the performance for different classification methods.



↑ Fig. 4.5 Classification Performance

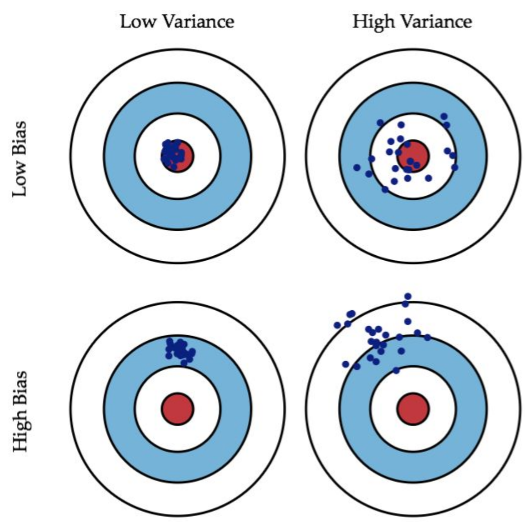
For this project, the result shows that "Linear" gives a significant performance for the classification.

5. Conclusion

At first, We were thinking about using AFINN[11] to do the Lexicon. The reason that we decided not to use it is because that those lexicon dictionary doesn't work properly with my method. Though it covers a lot of common words people use. And Instead of using R or NLTK [12], we feel like SKLearn with Python might be an easier option. In the result, sometimes, we can only get less tweets and users than what we expected. Later on, we realized that it was because that users might not have enough tweets or followers than we expected.

The sentiment analysis may give a emotional orientation. However, it can not distinguish whether the tweet is just an event information and the tweet contains a lot of slangs. Therefore, this might cause some of the tweets are misclassified.

Comparing our result to the research: Sentiment Analysis of Twitter Data [1], their result is meeting 60.50% average accuracy. And our best result is 81%. As the result, our project has a better performance (increasing 20.5% accuracy). However, Their research has larger datasets and more into details. Our project is smaller and simpler than theirs which means higher bias and lower variance, as shown in Fig. 5.1 [9]



↑ Fig. 5.1 Relationship between Bias and Variance [9]

6. Future work

The approach method for this project still has a few things which can be improved. First, we can increase the number of the datasets for a more accurate classification by paying for Twitter's developer account (more data and requests). Second, for the real-time tweets we didn't remove emojis which would affect the result. However, the pre-labeled dataset has removed all the emoji. In our opinion, it might not be the best way to do so because emoji is also a way for users to represent their emotions. We can come up with a better way to handle this issue in the future.

In the end, there are still a lot of different ways for classification that we haven't tried for this project. We may want to put those in our project in the future to compare the performances.

Reference

[1] Sentiment Analysis of Twitter Data: Apoorv Agarwal, Boyi Xie, Ilia Vovsh, Owen Rambow Rebecca Passonneau

[2] Sentiment140 - A Twitter Sentiment Analysis Tool

[3]http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip

[4]https://developer.twitter.com/en/docs

[5] scikit-learn Machine Learning in Python http://scikit-learn.org/stable/index.html

[6]http://scikit-learn.org/stable/modules/svm.html

[8] sklearn.metrics http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

[9] Substance.io.

[10] NetworkX https://networkx.github.io

[11]AFINN

[12]Natural Language Toolkit

Appendix

<u>Source Code:</u>

1. Run.sh:

```
#!/bin/bash

ver=$(python --version 2>&1 | sed 's/.*
\([0-9]\).\([0-9]\).*/\1\2/')
if [ $ver -lt 35 ]; then
        echo "Reuired Python 3.5"
        exit 1
fi

python collect.py
python cluster.py
python classify.py
```

2. Collect.py:

```
"""
collect.py
"""
import sys
import time
from TwitterAPI import TwitterAPI
import pickle

consumer_key =
'VsAo207IoIRF5AASDIlD3H7yE'
consumer_secret =
'4bNlsfogEbneVQp1TOLMk1ZGnwjbcqNe
N8apiintWoa7bYJrHA'
access_token =
'3164289948-CA0b188o68fVkbWJxXjkX1
3FnTmoKBplRf0nGZp'
access_token_secret =
'lz1ZweTDeKjZiBihfKcs2JQA3W58TNJfEl
qlA3aHYunEY'

def get_twitter():
   return TwitterAPI(consumer_key,
consumer_secret, access_token,
access_token_secret)
```

```
def robust_request(twitter, resource,
params, max_tries=5):
    """ If a Twitter request fails, sleep for 15
minutes.
    Do this at most max_tries times before
quitting.
    Args:
      twitter .... A TwitterAPI object.
      resource ... A resource string to
request; e.g., "friends/ids"
      params ..... A parameter dict for the
request, e.g., to specify
                   parameters like screen_name
or count.
      max_tries .. The maximum number of
tries to attempt.
    Returns:
      A TwitterResponse object, or None if
failed.
    """
    for i in range(max_tries):
        request = twitter.request(resource,
params)
        if request.status_code == 200:
            return request
        else:
            print('Got error %s \nsleeping for
15 minutes.' % request.text)
            sys.stderr.flush()
            time.sleep(61 * 15)


def get_users(twitter):
    return  twitter.request('users/search',
{'q':'Chicago Bears','count':10}).json()

def get_users_friend(twitter,
screen_names):
    """
        return a dict of users friends
    """
    ret_dict = {}
    for screen_name in screen_names:
        list_friends =
twitter.request('friends/ids',
{'screen_name':screen_name,
'count':20}).json()
        ret_dict[screen_name] = list_friends

    return ret_dict

def get_tweets(twitter, screen_names,
tweets_count):
    ret_dict = {}
    for screen_name in screen_names:
        list_tweets =
twitter.request('statuses/user_timeline',
{'screen_name':screen_name,
'count':tweets_count, "lang": "en"}).json()
        ret_dict[screen_name] = list_tweets

    return ret_dict

def get_num_of_friends(users,
friends_dict):
    num = 0
    for u in users:
        screen_name = u['screen_name']
        num +=
len(friends_dict[screen_name]['ids'])
    return num

def main():
    twitter = get_twitter()
    users = get_users(twitter)
    f = open('./data/users.txt','wb')
    pickle.dump(users, f)
    users_list = [ n['screen_name'] for n in
users]
    friend_dict = get_users_friend(twitter,
users_list)
    f2 = open('./data/friends.txt','wb')
    pickle.dump(friend_dict, f2)
    tweets_count = 200
    tweets = get_tweets(twitter, users_list,
tweets_count)
    f3 = open('./data/tweets.txt','wb')
    pickle.dump(tweets, f3)
    test_data = []
    for key, val in tweets.items():
        for t in val:
```

```
        test_data.append(t['text'])
    """"
    train_dict =
twitter.request('search/tweets',
{'q':'Chicago Bears','count':20, "lang":
"en"}).json()
    f4 = open('./data/train_tweets.txt','wb')
    pickle.dump(train_dict, f4)
    """
    num_friend =
get_num_of_friends(users, friend_dict)
    list_of_summarize = []

list_of_summarize.append(len(users_list)
+ num_friend)

list_of_summarize.append(len(test_data))
    f4 = open('./data/sum.txt','wb')
    pickle.dump(list_of_summarize, f4)
if __name__ == '__main__':
    main()
```

3. Cluster.py:

```
"""
cluster.py
"""
import sys
import networkx as nx
from collections import Counter
import pickle
import matplotlib.pyplot as plt # for
debugging
import math
from collections import Counter,
defaultdict, deque
import copy

def create_graph(users, friends_dict):
    """
    Args:
      users...........The list of user dicts.
      friend_counts...The Counter dict
mapping each friend to the number of
candidates that follow them.
    Returns:
      A networkx Graph
    """
    list_friend = []
    edges = []
    #list_friend = [i for i in friend_counts if
friend_counts[i]>1]
    graph = nx.Graph()

    for u in users:
        screen_name_id = u['id']
        screen_name = u['screen_name']
        graph.add_node(screen_name_id)
        #f = set(list_friend) &
set(friends_dict[screen_name])
        f =
set(friends_dict[screen_name]['ids'])
        for i in f:
            tup = (screen_name_id, i)
            edges.append(tup)

    graph.add_edges_from(edges)
    return graph

def draw_network(graph, users, filename):
    """
        for debugging the graph
    """
    label = {n:n for n in graph.nodes()}
    plt.figure(figsize=(12, 12))
    nx.draw_networkx(graph,
node_color='r', labels=label, width=.1,
node_size=100)
    plt.axis("off")
    plt.savefig(filename)
    plt.show()

def partition_girvan_newman(graph,
max_depth):
    graph_c = graph.copy()
    ret_list = []
    #ibet_dict =
approximate_betweenness(graph_c,
max_depth)
    ibet_dict =
nx.betweenness_centrality(graph)
    ib = sorted(ibet_dict.items(),
key=lambda i: i[1], reverse=True)
```

```
    components = [c for c in
nx.connected_component_subgraphs(gra
ph_c)]
    while len(components) == 1:
        graph_c.remove_edge(*ib[0][0])
        del ib[0]
        components = [c for c in
nx.connected_component_subgraphs(gra
ph_c)]
    for c in components:
        ret_list.append(c)

    return ret_list

def main():
    f = open('./data/users.txt','rb')
    f2 = open('./data/friends.txt', 'rb')
    users = pickle.load(f)
    user_list = [u['screen_name'] for u in
users]
    # Creating the graph
    friends_dict = pickle.load(f2)
    graph = create_graph(users,
friends_dict)
    print('graph has %s nodes and %s
edges' % (len(graph.nodes()),
len(graph.edges())))
    draw_network(graph, user_list,
'network.png')
    # begin clustering
    clusters =
partition_girvan_newman(graph, math.inf)
    print (clusters)
    total_nodes = 0
    for c in clusters:
        total_nodes += c.number_of_nodes()

    list_of_summarize = []
    list_of_summarize.append(len(clusters))
    list_of_summarize.append(total_nodes /
len(clusters))

    f4 = open('./data/sum.txt','ab')
    pickle.dump(list_of_summarize, f4)
if __name__ == '__main__':
    main()
```

4. Classify.py:

```
"""
classify.py
"""
import sys
import pickle
import os
import time
from sklearn.feature_extraction.text import
TfidfVectorizer
from sklearn import svm
from sklearn.metrics import
classification_report
import pandas as pd

def main():
    f = open('./data/tweets.txt', 'rb')
    tweets = pickle.load(f)
    f2 = open('./data/users.txt', 'rb')
    users = pickle.load(f2)
    user_list = sorted([u['screen_name'] for
u in users])
    test_data = []
    for u in user_list:
        for t in tweets[u]:
            test_data.append(t['text'])
    train_data_pd =
pd.read_csv('./data/trainData.csv',
encoding = "utf8")
    train_labels =
train_data_pd['polarity'].tolist()
    train_data = train_data_pd['text'].tolist()
    # Create feature vectors
    vectorizer = TfidfVectorizer(min_df=5,
                        max_df = 0.8,
                        sublinear_tf=True,
                        use_idf=True)
    train_vectors =
vectorizer.fit_transform(train_data)
    test_vectors =
vectorizer.transform(test_data)


    # Perform classification with SVM,
kernel=rbf
```

```
    classifier_rbf = svm.SVC()
    t0 = time.time()
    classifier_rbf.fit(train_vectors,
train_labels)
    t1 = time.time()
    prediction_rbf =
classifier_rbf.predict(test_vectors)
    t2 = time.time()
    time_rbf_train = t1-t0
    time_rbf_predict = t2-t1


    # Perform classification with SVM,
kernel=linear
    classifier_linear =
svm.SVC(kernel='linear')
    t0 = time.time()
    classifier_linear.fit(train_vectors,
train_labels)
    t1 = time.time()
    prediction_linear =
classifier_linear.predict(test_vectors)
    t2 = time.time()
    time_linear_train = t1-t0
    time_linear_predict = t2-t1


    # Perform classification with SVM,
kernel=poly, degree=3
    classifier_poly = svm.SVC(kernel='poly')
    t0 = time.time()
    classifier_poly.fit(train_vectors,
train_labels)
    t1 = time.time()
    prediction_poly =
classifier_poly.predict(test_vectors)
    t2 = time.time()
    time_poly_train = t1-t0
    time_poly_predict = t2-t1

    list_of_summarize = []
    pos = neg = 0

    for r in prediction_linear:
        if r == 4:
            pos += 1
        else:
            neg += 1

    list_of_summarize.append(pos)
    list_of_summarize.append(neg)


list_of_summarize.append(prediction_line
ar[0])
    list_of_summarize.append(test_data[0])

    f4 = open('./data/sum.txt','ab')
    pickle.dump(list_of_summarize, f4)
    #print(type(prediction_linear))


    print("Results for SVC(kernel=rbf)")
    print("Training time: %fs; Prediction
time: %fs" % (time_rbf_train,
time_rbf_predict))
    # print(classification_report(test_labels,
prediction_rbf))
    print(prediction_rbf)
    print("Results for SVC(kernel=linear)")
    print("Training time: %fs; Prediction
time: %fs" % (time_linear_train,
time_linear_predict))
    # print(classification_report(test_labels,
prediction_linear))
    print(prediction_linear)

    print("Results for SVC(kernel=poly)")
    print("Training time: %fs; Prediction
time: %fs" % (time_poly_train,
time_poly_predict))
    # print(classification_report(test_labels,
prediction_poly))
    print(prediction_poly)

if __name__ == '__main__':
    main()
```

5. Classify_testing.py:

```
"""
classify_testing.py
"""
```

```
import sys
import pickle
import os
import time
from sklearn.feature_extraction.text import
TfidfVectorizer
from sklearn import svm
from sklearn.metrics import
classification_report
import pandas as pd

def main():
    """
    f = open('./data/tweets.txt', 'rb')
    tweets = pickle.load(f)
    f2 = open('./data/users.txt', 'rb')
    users = pickle.load(f2)
    user_list = sorted([u['screen_name'] for
u in users])
    """
    test_data_pd =
pd.read_csv('./data/trainData.csv',
encoding = "utf8")
    test_labels =
test_data_pd['polarity'].tolist()
    test_data = test_data_pd['text'].tolist()

    train_data_pd =
pd.read_csv('./data/trainData.csv',
encoding = "utf8")
    train_labels =
train_data_pd['polarity'].tolist()
    train_data = train_data_pd['text'].tolist()
    # Create feature vectors
    vectorizer = TfidfVectorizer(min_df=5,
                                 max_df = 0.8,
                                 sublinear_tf=True,
                                 use_idf=True)
    train_vectors =
vectorizer.fit_transform(train_data)
    test_vectors =
vectorizer.transform(test_data)


    # Perform classification with SVM,
kernel=rbf
    classifier_rbf = svm.SVC()
    t0 = time.time()
    classifier_rbf.fit(train_vectors,
train_labels)
    t1 = time.time()
    prediction_rbf =
classifier_rbf.predict(test_vectors)
    t2 = time.time()
    time_rbf_train = t1-t0
    time_rbf_predict = t2-t1


    # Perform classification with SVM,
kernel=linear
    classifier_linear =
svm.SVC(kernel='linear')
    t0 = time.time()
    classifier_linear.fit(train_vectors,
train_labels)
    t1 = time.time()
    prediction_linear =
classifier_linear.predict(test_vectors)
    t2 = time.time()
    time_linear_train = t1-t0
    time_linear_predict = t2-t1


    # Perform classification with SVM,
kernel=poly, degree=3
    classifier_poly = svm.SVC(kernel='poly')
    t0 = time.time()
    classifier_poly.fit(train_vectors,
train_labels)
    t1 = time.time()
    prediction_poly =
classifier_poly.predict(test_vectors)
    t2 = time.time()
    time_poly_train = t1-t0
    time_poly_predict = t2-t1

    print("Results for SVC(kernel=rbf)")
    print("Training time: %fs; Prediction
time: %fs" % (time_rbf_train,
time_rbf_predict))
    print(classification_report(test_labels,
prediction_rbf))
```

```
    print("Results for SVC(kernel=linear)")
    print("Training time: %fs; Prediction
time: %fs" % (time_linear_train,
time_linear_predict))
    print(classification_report(test_labels,
prediction_linear))

    print("Results for SVC(kernel=poly)")
    print("Training time: %fs; Prediction
time: %fs" % (time_poly_train,
time_poly_predict))
    print(classification_report(test_labels,
prediction_poly))

if __name__ == '__main__':
    main()
```