

Twitter United Airlines Sentiment Analysis Based on Support Vector Regression and Word to Vector

Yuanhui Yang and Cao Xue
{YuanhuiYang2015 and CaoXue2016}@u.northwestern.edu

Introduction

Nowadays social networking sites are at the boom, so large amount of data is generated. Millions of people are sharing their views daily on micro blogging sites, since it contains short and simple expressions. Twitter is one of the famous social networking services where user can read and post messages which are 148 characters in length. Twitter messages are also called as Tweets. We will use these tweets as raw data. We will use a method that automatically extracts tweets into positive, negative or neutral sentiments. By using the sentiment analysis the customer can know the feedback about the product or services before making a purchase. The company can use sentiment analysis to know the opinion of customers about their products, so that they can analyze customer satisfaction and according to that they can improve their product.

Every major American airline maintains a customer service twitter feed. It seems like a thankless job, consisting largely of replying to endless grievances about weather-related cancellations and spotty in-flight wifi with “We’re sorry you’re upset. Please send us your reservation number.” Then you get yelled at. That said, it’s a tremendous source of sentiment data.

In this project, we will use machine learning algorithms for classifying the sentiment of Twitter messages about Airline company. Thus the airline company would get a clear insight of their service and make a quick response to those automatically chosen Tweets with a negative sentiments. The best perspective is that machine would read individual tweets (and conversations with customer service agents) and determine whether they were positive, negative, or neutral.

Data selection

Based on the dataset from Crowdfunder's Data for Everyone library, the Twitter data was scraped from February of 2015 and contributors were asked to first classify positive, negative, and neutral tweets, followed by categorizing negative reasons (such as "late flight" or "rude service"). Note that Tweets database table includes the following labels:

- tweet_id
- airline_sentiment
- airline_sentiment_confidence
- negativereason
- negativereason_confidence
- airline

- airline_sentiment_gold
- name
- negativereason_gold
- retweet_count
- text
- tweet_coord
- tweet_created
- tweet_location
- user_timezone

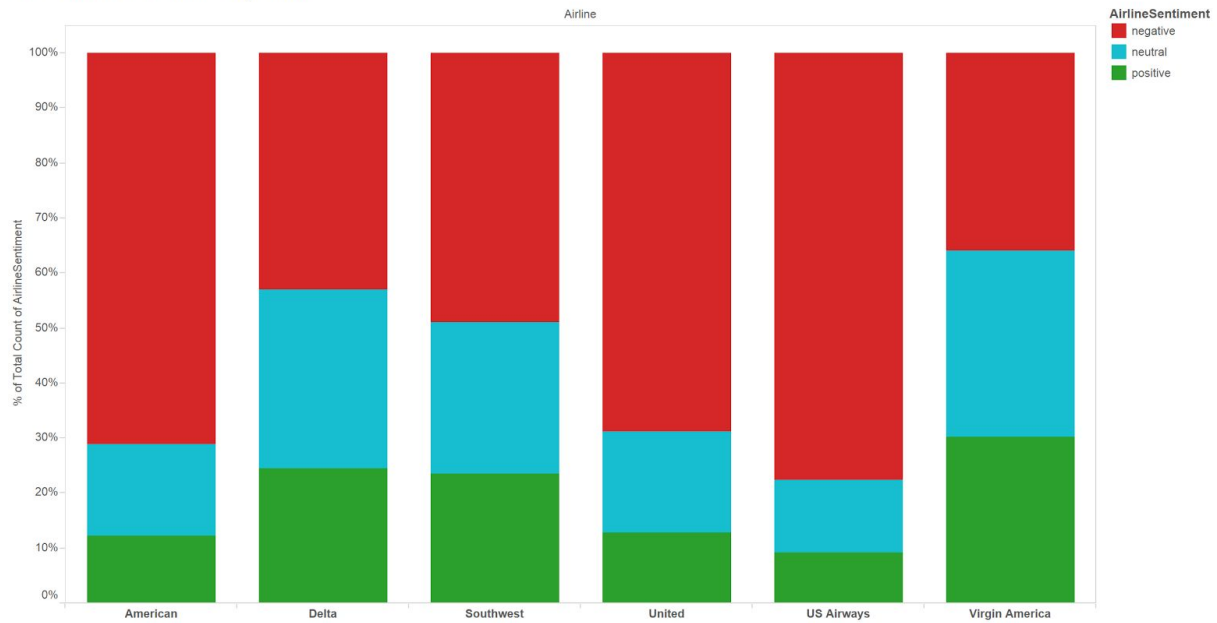
tweet_id	airline_sentiment	airline_sentiment_gold	negativereason	negativereason_gold	airline	airline_sentiment	name	negativereason	retweet_count	text	tweet_coord	tweet_created	tweet_location	user_timezone
5.7031E+17	neutral	1			Virgin America	cairdin			0	@VirginAmerica What @#####				Eastern Time (US & Canada)
5.703E+17	positive	0.3486		0	Virgin America	jnardino			0	@VirginAmerica plus you#####				Pacific Time (US & Canada)
5.703E+17	neutral	0.6837			Virgin America	yvonallynn			0	@VirginAmerica I didn't t##### Lets Play				Central Time (US & Canada)
5.703E+17	negative	1	Bad Flight	0.7033	Virgin America	jnardino			0	@VirginAmerica it's really#####				Pacific Time (US & Canada)
5.703E+17	negative	1	Can't Tell	1	Virgin America	jnardino			0	@VirginAmerica and it's a#####				Pacific Time (US & Canada)
										@VirginAmerica seriously would pay \$30 a flight for seats that didn't have this playing. it's really the only bad thing about				
5.703E+17	negative	1	Can't Tell	0.6842	Virgin America	jnardino			0	flying VA #####				Pacific Time (US & Canada)
5.703E+17	positive	0.6745		0	Virgin America	cjmcginnis			0	@VirginAmerica yes, near##### San Francisco				Pacific Time (US & Canada)
5.703E+17	neutral	0.634			Virgin America	pilot			0	@VirginAmerica Really m##### Los Angeles				Pacific Time (US & Canada)
5.703E+17	positive	0.6559			Virgin America	dhepburn			0	@virginamerica Well, I did##### San Diego				Pacific Time (US & Canada)
5.703E+17	positive	1			Virgin America	YupitsTate			0	@VirginAmerica it was an##### Los Angeles				Eastern Time (US & Canada)
5.7029E+17	neutral	0.6769		0	Virgin America	idk_but_youtube			0	@VirginAmerica did you w##### 1/1 loner sq				Eastern Time (US & Canada)
5.7029E+17	positive	1			Virgin America	HyperCamiLax			0	@VirginAmerica I <3 pr##### NYC				America/New_York
5.7029E+17	positive	1			Virgin America	HyperCamiLax			0	@VirginAmerica This is su##### NYC				America/New_York
5.7029E+17	positive	0.6451			Virgin America	mollanderson			0	@VirginAmerica @virgin#####				Eastern Time (US & Canada)
5.7029E+17	positive	1			Virgin America	sjspers			0	@VirginAmerica Thanks! ##### San Francisco				Pacific Time (US & Canada)
5.7028E+17	negative	0.6842	Late Flight	0.3684	Virgin America	smartwatermelon			0	@VirginAmerica SFO-PDX##### palo alto, ca				Pacific Time (US & Canada)
5.7028E+17	positive	1			Virgin America	ltzBrianHunty			0	@VirginAmerica So excite##### west covina				Pacific Time (US & Canada)
5.7028E+17	negative	1	Bad Flight	1	Virgin America	heatherovieda			0	@VirginAmerica I flew frd##### this place cal				Eastern Time (US & Canada)
5.7027E+17	positive	1			Virgin America	thebrandiray			0	I ÷ flying @VirginAm##### Somewhere				Atlantic Time (Canada)
5.7027E+17	positive	1			Virgin America	JNLpierce			0	@VirginAmerica you know##### Boston Wa Quito				

This is a screenshot for our dataset. We can find that the dataset is so messy.

Data Visualization(using Tableau)

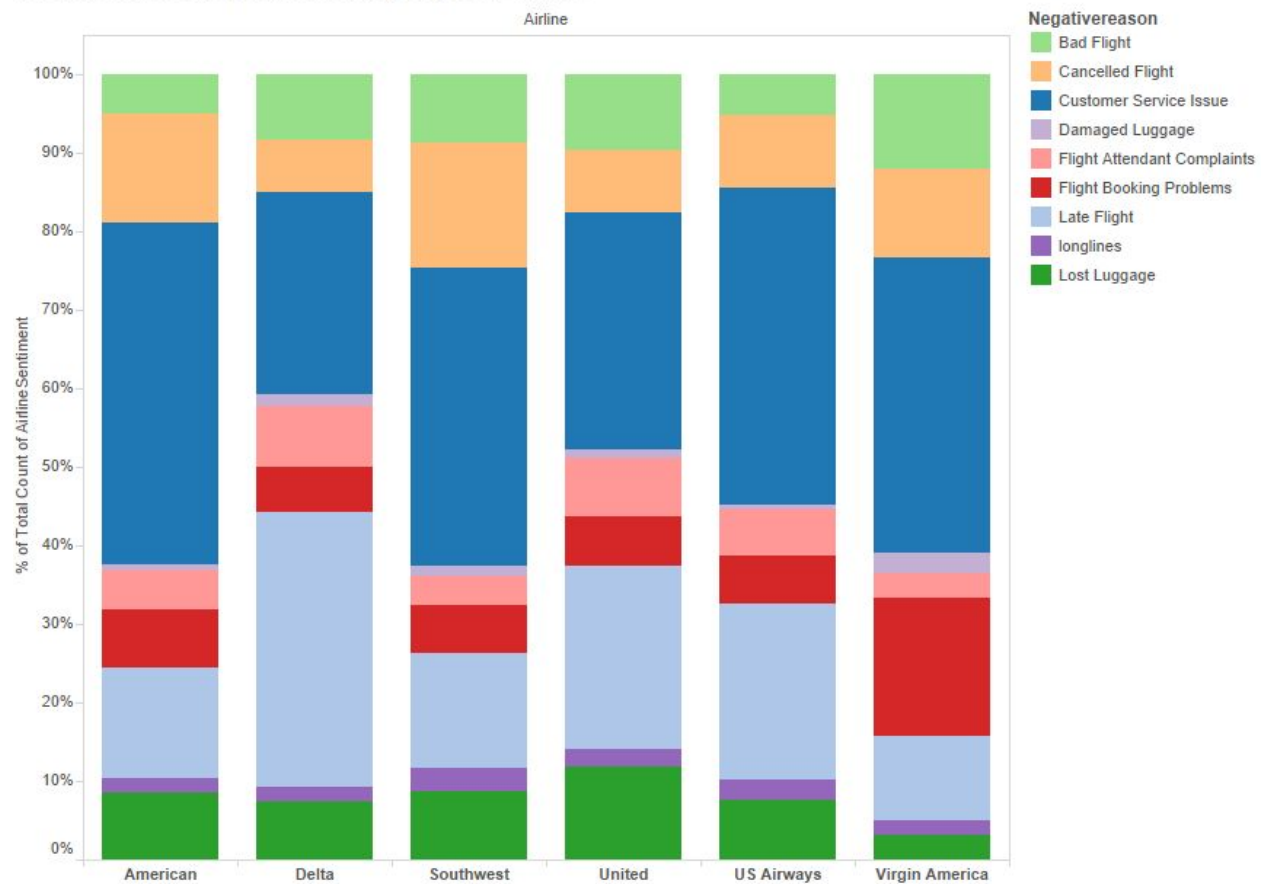
We could see the Total Distros of Sentiment by Airline

% Total Distros of Sentiment by Airline



The Total Distro of Negative Reason grouped by Airline

% Total Distro of Negative Reason grouped by Airline



Through these graphs, we could have a higher view of the construction of the reasons of negation review.

Feature Selection

As you could see in the table, some labels owns too many NULL or NaN elements (over 50%), hence we will drop these labels. In addition, generally, 'tweet_id' is a random key, which is meaningful in terms of data mining, therefore, we drop it. Finally, we choose the following labels as Input Set or Features:

['airline_sentiment', 'airline_sentiment_confidence', 'airline', 'text', 'tweet_created', 'tweet_location', 'user_timezone']

Python Code:

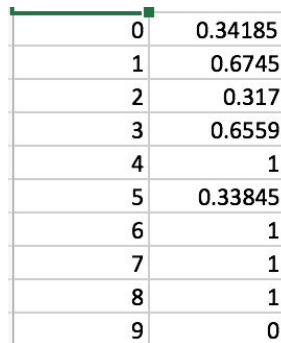
```
def feature(self):
    self.feature = ['airline_sentiment', 'airline_sentiment_confidence', 'airline', 'text',
'tweet_created', 'tweet_location', 'user_timezone']
    return self.feature
```

As for Output Set or Targets, there exists 2 meaningful labels, ['airline_sentiment', 'airline_sentiment_confidence']. Then, we maps 'airline_sentiment' as {'neutral': 0.5, 'negative':0.0, 'positive':1.0}. Finally, we merge the 2 labels with multiplication operator. Note that, we transform this classification problem into regression problem. This operation will bring more accurate solutions and answers.

Python Code:

```
def setOutput(self):
    airline_sentiment = np.array(self.raw_table['airline_sentiment'])
    airline_sentiment_confidence =
np.array(self.raw_table['airline_sentiment_confidence'])
    self.length_of_sequence = len(airline_sentiment)
    self.output = np.zeros(self.length_of_sequence)
    for idx in range(self.length_of_sequence):
        if airline_sentiment[idx] == 'neutral':
            self.output[idx] = np.float64(0.5 *
airline_sentiment_confidence[idx])
        elif airline_sentiment[idx] == 'positive':
            self.output[idx] = np.float64(1.0 *
airline_sentiment_confidence[idx])
        else:
            self.output[idx] = np.float64(0.0)
    pd.DataFrame(self.output).to_csv('./data/Output.csv', sep=',')
    return self.output
```

Finally, there still exists some invalid elements like NaN or NULL in some rows, we choose to drop the entire rows.



0	0.34185
1	0.6745
2	0.317
3	0.6559
4	1
5	0.33845
6	1
7	1
8	1
9	0

This is the screenshot of output after preprocessing.

Feature Extraction

Feature Extraction is the one of two key parts, and the other one is Support Vector Regression. It is very different from Feature Selection work. Feature Extraction aims to transforming arbitrary data, such as text or images, into numerical features usable for machine learning. In this dataset, the top most important labels are in string datatype. Therefore, text feature extraction is key parts. Specifically, we will try several word2vec technologies to find the best one to generate the feature vectors. We try common vectorizer and Tf-idf term weighting vectorizer as well as hashing trick vectorizer methods. However, the results provided from common vectorizer and Tf-idf term weighting vectorizer are too sparse (over 80% elements are 0). Finally, we choose the hashing trick vectorizer method.

- 'text' Label Vector Presentation

As for the length of 'text' label presentation vector, we try 1000, 300, 50, 30, 10, 5 and so on. Finally, we choose 30 as the length of 'text' label presentation vector.

Python Code:

```
def text2Vector(self):
    self.text2Vector = self.word2Vector('text', 30)
    pd.DataFrame(self.text2Vector).to_csv('../data/Text2Vector.csv', sep=',')
    return self.text2Vector
```

```
def word2Vector(self, item, num_feature):
    corpus = np.array(self.raw_table[item])
    hashingVectorizer = HashingVectorizer(decode_error='ignore',
    n_features=num_feature, non_negative=False)
    word2Vector = hashingVectorizer.fit_transform(corpus).toarray()
    return word2Vector
```

```

        # pd.DataFrame(word2Vector).to_csv('../data/result.csv', sep=',',
encoding='utf-8',)
        # print self.vector.ravel()
        # print type(corpus[0])
        # print corpus
        # self.vectorizer = TfidfVectorizer(min_df=1, max_df=1.0,
stop_words='english', max_features=10, norm='l2', sublinear_tf=True)
        # print self.vectorizer
        # vec = self.vectorizer.fit_transform(corpus).toarray()
        # print vec.toarray()

```

- 'airline' Label Vector Presentation

Actually, there are 6 different airplane companies, which can be calculated by set() function. Then we design a unit vector to present it. This vector only owns 1 non-zero element. This operation will differ them very much. For example, [1, 0, 0, 0, 0, 0] is vertical to [0, 1, 0, 0, 0, 0], so this difference is relatively large, and they cannot present each other absolutely.

Python Code:

```

def airline2Vector(self):
    corpus = np.array(self.raw_table['airline'])
    set_corpus = set(corpus)
    list_corpus = list(set_corpus)
    # print array_corpus
    length_row = len(corpus)
    length_column = len(set_corpus)
    self.airline2Vector = np.zeros((length_row, length_column))
    # print corpus
    for i in range(length_row):
        for j in range(length_column):
            if corpus[i] == list_corpus[j]:
                self.airline2Vector[i][j] = 1.0
    pd.DataFrame(self.airline2Vector).to_csv('../data/Airline2Vector.csv', sep=',')
    return self.airline2Vector

```

- - 'tweet_created', 'tweet_location' and 'user_timezone' Label Vector Presentation

For the remaining label, they are all string data type. In fact, we can still present them as vector in the similar word2vec technology. In this research, we use 10-size vector to present them.

Python Code

```
def tweetCreated2Vector(self):
    self.tweetCreated2Vector = self.word2Vector('tweet_created', 10)

pd.DataFrame(self.tweetCreated2Vector).to_csv('../data/TweetCreated2Vector.csv',
sep=',')

return self.tweetCreated2Vector
```

Python Code

```
def tweetlocation2Vector(self):
    self.tweetlocation2Vector = self.word2Vector('tweet_location', 10)

pd.DataFrame(self.tweetlocation2Vector).to_csv('../data/TweetLocation2Vector.csv',
sep=',')

return self.tweetlocation2Vector
```

Python Code

```
def userTimezone2Vector(self):
    self.userTimezone2Vector = self.word2Vector('user_timezone', 10)

pd.DataFrame(self.userTimezone2Vector).to_csv('../data/UserTimezone2Vector.csv',
sep=',')

return self.userTimezone2Vector
```

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF		
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30		
0	0	0	0	0	0	0	0	0.3162278	0	0	0	0.6309573	0	-0.3162278	0	0	0	-0.3333333	0	0	0.3333333	0	0	0	-0.3162278	-0.3162278	0	0	0	0	0		
1	0	0	0	0	0	0	0	-0.3333333	0	0.3333333	0	0	0	-0.2672611	0	0	0	-0.3333333	0	0	0.3333333	0	0	0	-0.3333333	-0.3333333	0	0	-0.3333333	0	0		
2	0	0	0	0	0	0	0	0	0.2672611	0	0	0	0	0	0	0	0	0	0	0	0	0.2672611	0.5345222	-0.2672611	0	-0.2672611	0.2672611	0.3758162	0.2672611	0	0		
3	0	0	0	0	0	0	0	0.4082483	0.4082483	-0.4082483	0	0.4082483	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
4	0	0	0	0	0	0	0	0	0	0	-0.3015111	0	0	0	-0.3015111	0	0.3015111	0	0.3015111	0	0	0	0	0	0	0.3015111	0.3015111	0	0.3015111	0	0		
5	0	0	0	0	0	0	0	0	0	0	0.2773501	0.2773501	-0.2773501	0	0	-0.2773501	0	-0.2773501	0	-0.2773501	0	0	0	0	0	0	0	0	0	0	0		
6	0	0	0	0	0	0	0	0	0	0.3333334	0.3333334	0	0	0.3333334	0	0	0	0	0.3333334	0	0	0	0	0	0	0.3333334	-0.3333333	0	0	0	0		
7	0	0	0	0	0	0	0	0	0.3849002	0	0.1924501	0	0	0	0.3849002	-0.3849002	0	0.1924501	-0.1924501	0	0.1924501	-0.1924501	0	0	0.3849002	-0.1924501	0.3849002	0	0	0	0		
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
10	0	0	0	0	0	0	0	0	0.2132007	0	0.2132007	0	-0.2132007	0	0	0.2132007	0	0.2132007	0	0.2132007	0	0	0	0	0.2132007	-0.2132007	-0.2132007	-0.2132007	0	0	0.2132007	0	
11	0	0	0	0	0	0	0	0	0.3651484	0.1825742	0	0	-0.1825742	0.1825742	0	-0.1825742	0	-0.5477351	0	0	0	0	0	0	0.1825742	0.1825742	0.3651484	0	-0.1825742	0	0		
12	0	0	0	0	0	0	0	0	0.7071068	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
13	0	0	0	0	0	0	0	0	0.4545454	0	0.2181179	-0.2181179	0	0	0.2181179	-0.2181179	0	-0.2181179	0	0.2181179	-0.2181179	0	0	0	0	-0.2181179	-0.2181179	0	0	0	0		
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-0.3535533	-0.3535533	0	0	-0.3535533	0.3535534	0	-0.3535533	0	-0.3535533	0	0	0		
15	0	0	0	0	0	0	0	0	0.5015111	0	0.3015111	0	0	0.3015111	0	0	0	0	-0.3015111	0	0	0	0	0	0	0.3015111	0.3015111	0	0.3015111	0	0	0.3015111	
16	0	0	0	0	0	0	0	0	0.1825742	0	0	0.1825742	0	0	0.3651484	0	0.1825742	0	0	0	0	0	-0.1825742	-0.1825742	0.1825742	0.5477351	0	0	0.1825742	0	0		
17	0	0	0	0	0	0	0	0	0.2773501	0	0	0	-0.2773501	0	0	0	0.2773501	0	0	0	0	0	-0.2773501	-0.2773501	0	0	0	0	0	0	0		
18	0	0	0	0	0	0	0	0	0.8017837	0	0	0	-0.2672611	0	0	0	0.2672611	-0.2672611	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-0.5345222	-0.2672611	0	-0.5345222	0	0	0	0	0	0	-0.2672611	0	0	0	0	-0.2672611	0	
20	0	0	0	0	0	0	0	0	0.1924501	-0.3849002	-0.1924501	0	0	0	0.1924501	-0.3849002	0	0.1924501	-0.1924501	0	0.1924501	-0.1924501	0	0	0	-0.3849002	-0.3849002	-0.1924501	0	0	0	0	
21	0	0	0	0	0	0	0	0	0.2672611	0.2672611	0	0	0.2672611	0.2672611	0	0	0.2672611	0	0	0	0	0.2672611	0	0	0	0.2672611	0	0.2672611	0.2672611	0	0	0	
22	0	0	0	0	0	0	0	0	0.25	0.25	-0.25	0	-0.25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
23	0	0	0	0	0	0	0	0	0.3535534	0	-0.3535533	0	0	0	0	0	0.3535534	0	0.3535534	0	-0.3535533	0	0	0.3535534	-0.3535533	0	0.3535534	0	0	0	0	0	
24	0	0	0	0	0	0	0	0	-0.2425356	0	-0.4850731	0	0	0.2425356	0	0.2425356	0	-0.2425356	0	0.2425356	0	0	0	0	0.4850731	-0.2425356	0	0	0.2425356	0	0		
25	0	0	0	0	0	0	0	0	0.3392106	0.1796053	0	0	0	-0.1796053	0	0	0	-0.3392106	0	0.1796053	0.3392106	-0.1796053	0	0	-0.3392106	0.1796053	0.3392106	0	0	0	0	0	
26	0	0	0	0	0	0	0	0	0.2294157	0.2294157	0	0	0	0	0	0	0	0	-0.4590731	0	0	0	0	0.2294157	-0.2294157	0	0.2294157	0	0	0	0.2294157	0	
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
32	0	0	0	0	0	0	0	0	0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989	-0.2581989		
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-0.2425356	0	0	0	0	0	0.2425356	0.2425356	0.2425356	0	0	0.2425356	0	0	
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
35	0	0	0	0	0	0	0	0	0.3015111	-0.3015111	0.3015111	-0.3015111	0	0	0	0	0	0	0	0	0	0	0	0	0.3015111	-0.3015111	0	0	0	0.3015111	0	0	
36	0	0	0	0	0	0	0	0	0.5547002	0.2773501	0	0	0	0	0	0	0.2773501	0.2773501	0	0	0	0	0	0	0	0.2773501	0	0	0.5547002	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0.2260604	0.2260604	0	0.2260604	0	0.2260604	0	0	0	0	0	0	0	0	0.2260604	-0.2260604	0	0	0.2260604	0	0	0	
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
41	0	0	0	0	0	0	0	0	0.2085144	0	-0.2085144	-0.2085144	-0.2085144	0	0	0	0	-0.2085144	0	0.2085144	0	0	0	0	-0.2085144	0.2085144	0	0	0.2085144	0	0	0.2085144	
42	0	0	0	0	0	0	0	0	-0.2773501	-0.2773501	0	0	0	0	0	0	-0.2773501	-0.2773501	0	0	0	0	0	0	0	-0.2773501	-0.2773501	0	0	0	0	0	0
43	0	0	0	0	0	0	0	0	0.3779645	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
45	0	0	0	0	0	0	0	0	0.1424871	0	0	0	0.2487142	0.1424871	0	0	-0.2487142	0	-0.2487142	0	0	0	0	0	0	0.1424871	0	0.1424871	-0.1424871	0	0	0	
46	0	0	0	0	0	0	0	0	0.5345222	0.2672611	-0.2672611	0	0	0	0	0.5345222	0.2672611	-0.2672611	0	0	0	0	0	0	0	0.2672611</							

This is the screenshot of our input to SVR after preprocessing.

Support Vector Regression

Overview

Support Vector Machines are theoretically well motivated algorithms and has been developed from statistical learning theory since the 60s. The class of algorithms called SVMs which are used for pattern recognition. They are effective and famous classification learning tool. The SVM algorithm is based on the statistical learning theory and the Vapnik Chervonenkis (VC) dimension introduced by Vladimir Vapnik and Alexey Chervonenkis. Support vector machines (SVM) are a group of supervised learning methods that can be applied to classification or regression. The principal of the SVR toolkit we used is as follows:

Given training vectors $x_i \in \mathbb{R}^p$, $i=1, \dots, n$, and a vector $y \in \mathbb{R}^n$ ε -SVR solves the following primal problem:

$$\begin{aligned} \min_{w, b, \zeta, \zeta^*} & \frac{1}{2} w^T w + C \sum_{i=1}^n (\zeta_i + \zeta_i^*) \\ \text{subject to} & y_i - w^T \phi(x_i) - b \leq \varepsilon + \zeta_i, \\ & w^T \phi(x_i) + b - y_i \leq \varepsilon + \zeta_i^*, \\ & \zeta_i, \zeta_i^* \geq 0, i = 1, \dots, n \end{aligned}$$

Its dual is

$$\begin{aligned} \min_{\alpha, \alpha^*} & \frac{1}{2} (\alpha - \alpha^*)^T Q (\alpha - \alpha^*) + \varepsilon e^T (\alpha + \alpha^*) - y^T (\alpha - \alpha^*) \\ \text{subject to} & e^T (\alpha - \alpha^*) = 0 \\ & 0 \leq \alpha_i, \alpha_i^* \leq C, i = 1, \dots, n \end{aligned}$$

$$\sum_{i=1}^n (\alpha_i - \alpha_i^*) K(x_i, x) + \rho$$

These parameters can be accessed through the members `dual_coef_` which holds the difference $\alpha_i - \alpha_i^*$, `support_vectors_` which holds the support vectors, and `intercept_` which holds the independent term ρ

We have tried three kernels:

- linear: $\langle x, x' \rangle$.
- rbf: $\exp(-\gamma |x - x'|^2)$. γ is specified by keyword gamma, must be greater than 0.
- sigmoid $\tanh(\gamma \langle x, x' \rangle + r)$, where r is specified by coef0.

When training an SVM with the Radial Basis Function (RBF) kernel, two parameters must be considered: C and gamma. The parameter C, common to all SVM kernels, trades off misclassification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly. gamma defines how much influence a single training example has.

Implementation

1. Standardization

Standardization of a dataset is a common requirement for many machine learning estimators. Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using the transform method.

Standardization
<pre> self.svr_sigmoid.fit(self.input_transform_train, self.output_transform_train) self.output_transform_predict_sigmoid = self.svr_sigmoid.predict(self.input_transform_test) self.output_predict_sigmoid = self.output_scaler.inverse_transform(self.output_transform_predict_sigmoid.reshape((self.length_of_prediction_sequence, 1))) </pre>

2. Cross-validation

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called overfitting. To avoid it, cross-validation is common practice. We use 10-fold cross-validation, in the following code, that is, 'cv=10'.

Cross-validation

```
self.svr_sigmoid = GridSearchCV(SVR(kernel='sigmoid', gamma=0.1), cv=10,
param_grid={'C': np.logspace(-10.0, 10.0, num=5, base=2.0), 'gamma':
np.logspace(-10.0, 10.0, num=5, base=2.0)})
```

3.Support Vector Regression

We use 3 popular kernels SVR methods to predict. 80% of dataset is training set and the other 20% dataset is test set.

Python Code

```
def svr_linear(self):
    self.svr_linear = GridSearchCV(SVR(kernel='linear', gamma=0.1), cv=10,
param_grid={'C': np.logspace(-10.0, 10.0, num=5, base=2.0), 'gamma':
np.logspace(-10.0, 10.0, num=5, base=2.0)})
    # self.svr_linear = SVR(kernel='linear', gamma=0.1)
    self.svr_linear.fit(self.input_transform_train, self.output_transform_train)
    self.output_transform_predict_linear =
self.svr_linear.predict(self.input_transform_test)
    self.output_predict_linear =
self.output_scaler.inverse_transform(self.output_transform_predict_linear.reshape((se
lf.length_of_prediction_sequence, 1)))
    self.square_root_of_mean_squared_error_linear =
sqrt(mean_squared_error(self.output_test, self.output_predict_linear))
    plt.figure()
    x = np.arange(0, self.length_of_prediction_sequence)
    plt.plot(x, self.output_test, 'ro-', label='Actual')
    plt.plot(x, self.output_predict_linear, 'bo-', label='Predicted')
    plt.title('linear-SVR: RMSE = %.3f'
%self.square_root_of_mean_squared_error_linear)
    plt.legend()
    plt.grid(True)
    plt.savefig('../figure/linear-SVR.eps', format='eps', dpi=1000)
    plt.savefig('../figure/linear-SVR.png', format='png', dpi=1000)
    # plt.show()
```

Python Code

```
def svr_rbf(self):
    self.svr_rbf = GridSearchCV(SVR(kernel='rbf', gamma=0.1), cv=10,
param_grid={'C': np.logspace(-10.0, 10.0, num=5, base=2.0), 'gamma':
np.logspace(-10.0, 10.0, num=5, base=2.0)})
```

```

        # self.svr_rbf = SVR(kernel='rbf', gamma=0.1)
        self.svr_rbf.fit(self.input_transform_train, self.output_transform_train)
        self.output_transform_predict_rbf =
self.svr_rbf.predict(self.input_transform_test)
        self.output_predict_rbf =
self.output_scaler.inverse_transform(self.output_transform_predict_rbf.reshape((self.l
length_of_prediction_sequence, 1)))
        self.square_root_of_mean_squared_error_rbf =
sqrt(mean_squared_error(self.output_test, self.output_predict_rbf))
        plt.figure()
        x = np.arange(0, self.length_of_prediction_sequence)
        plt.plot(x, self.output_test, 'ro-', label='Actual')
        plt.plot(x, self.output_predict_rbf, 'bo-', label='Predicted')
        plt.title('rbf-SVR: RMSE = %.3f'
%self.square_root_of_mean_squared_error_rbf)
        plt.legend()
        plt.grid(True)
        plt.savefig('../figure/rbf-SVR.eps', format='eps', dpi=1000)
        plt.savefig('../figure/rbf-SVR.png', format='png', dpi=1000)
        # plt.show()

```

Python Code

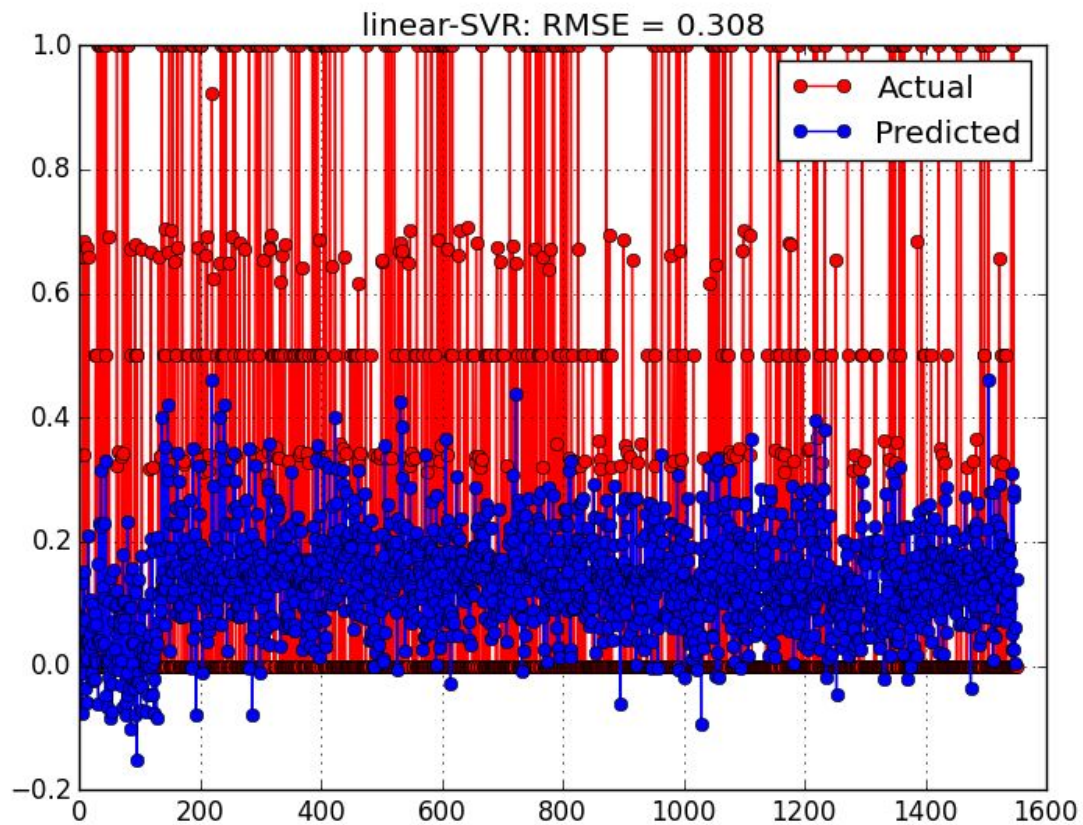
```

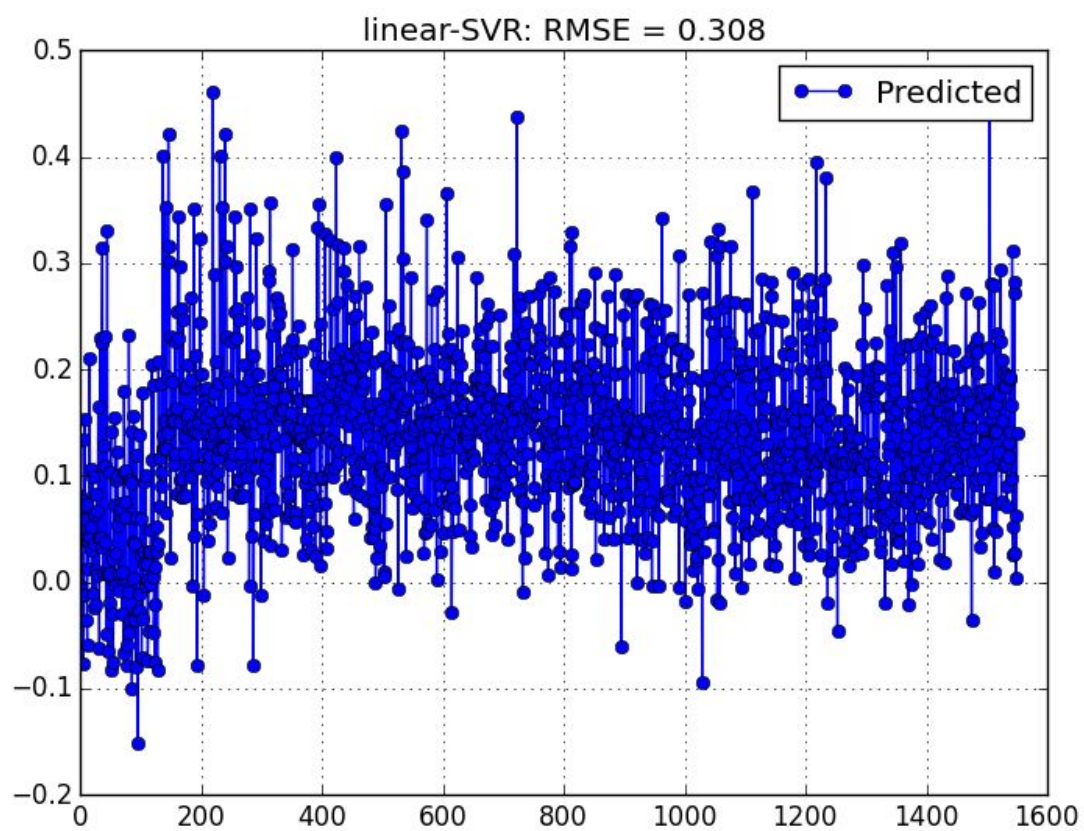
def svr_sigmoid(self):
    self.svr_sigmoid = GridSearchCV(SVR(kernel='sigmoid', gamma=0.1), cv=10,
param_grid={'C': np.logspace(-10.0, 10.0, num=5, base=2.0), 'gamma':
np.logspace(-10.0, 10.0, num=5, base=2.0)})
    # self.svr_sigmoid = SVR(kernel='sigmoid', gamma=0.1)
    self.svr_sigmoid.fit(self.input_transform_train, self.output_transform_train)
    self.output_transform_predict_sigmoid =
self.svr_sigmoid.predict(self.input_transform_test)
    self.output_predict_sigmoid =
self.output_scaler.inverse_transform(self.output_transform_predict_sigmoid.reshape((
self.length_of_prediction_sequence, 1)))
    self.square_root_of_mean_squared_error_sigmoid =
sqrt(mean_squared_error(self.output_test, self.output_predict_sigmoid))
    plt.figure()
    x = np.arange(0, self.length_of_prediction_sequence)
    plt.plot(x, self.output_test, 'ro-', label='Actual')
    plt.plot(x, self.output_predict_sigmoid, 'bo-', label='Predicted')
    plt.title('sigmoid-SVR: RMSE = %.3f'
%self.square_root_of_mean_squared_error_sigmoid)
    plt.legend()

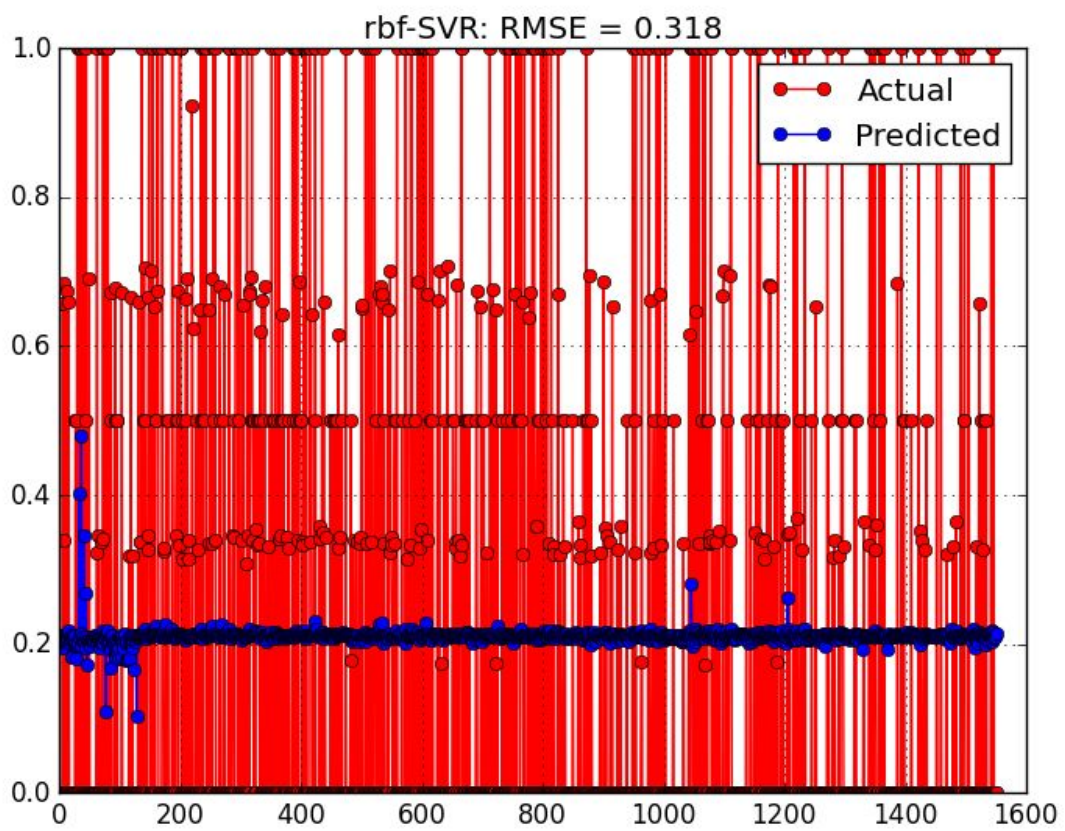
```

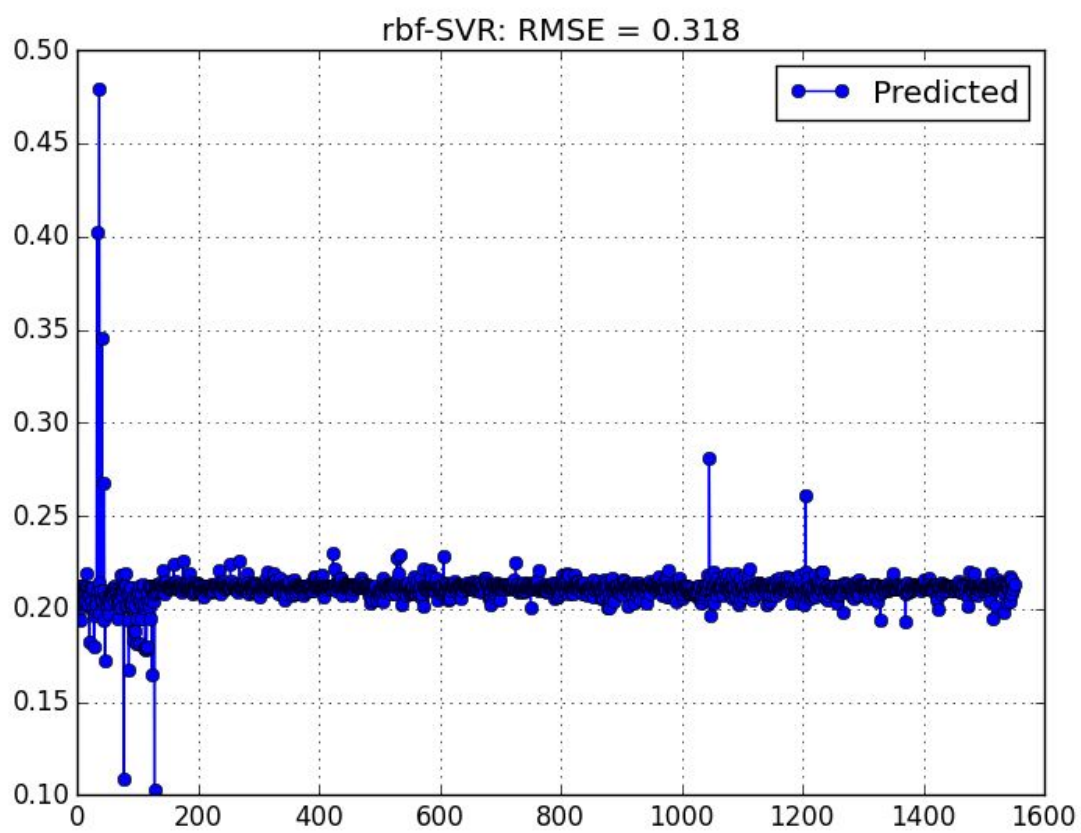
```
plt.grid(True)
plt.savefig('../figure/sigmoid-SVR.eps', format='eps', dpi=1000)
plt.savefig('../figure/sigmoid-SVR.png', format='png', dpi=1000)
# plt.show()
```

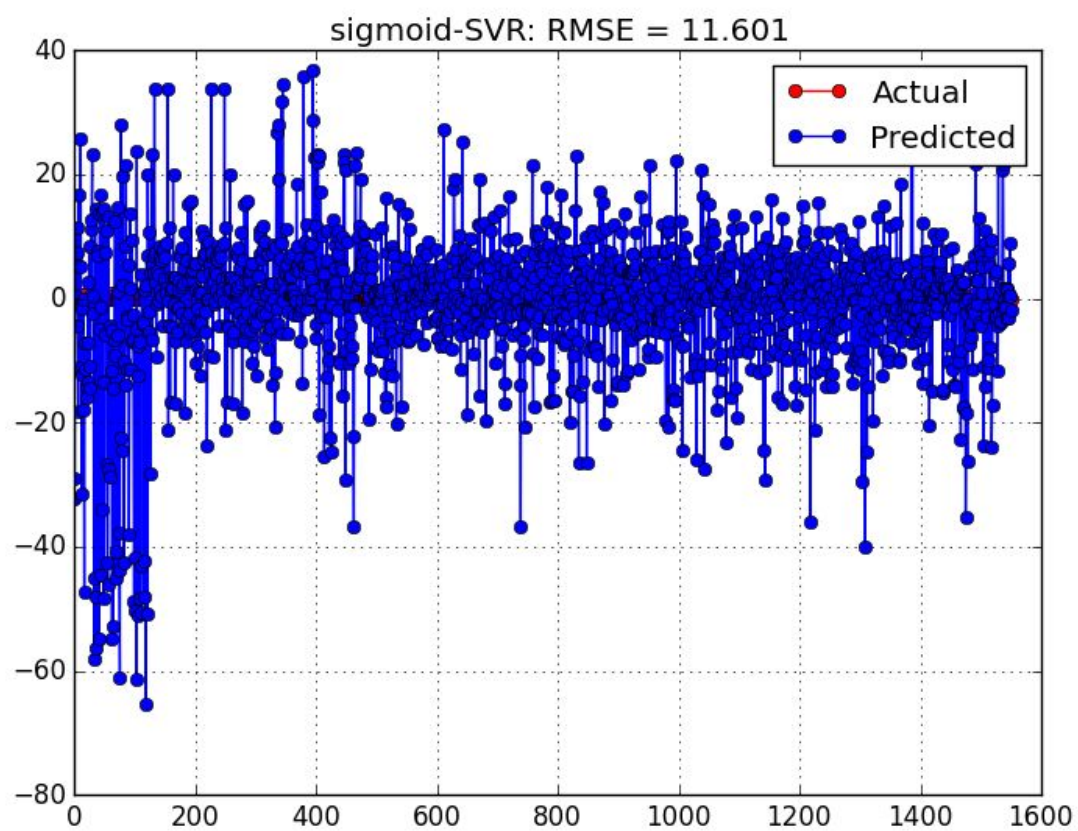
4.Experimental Results

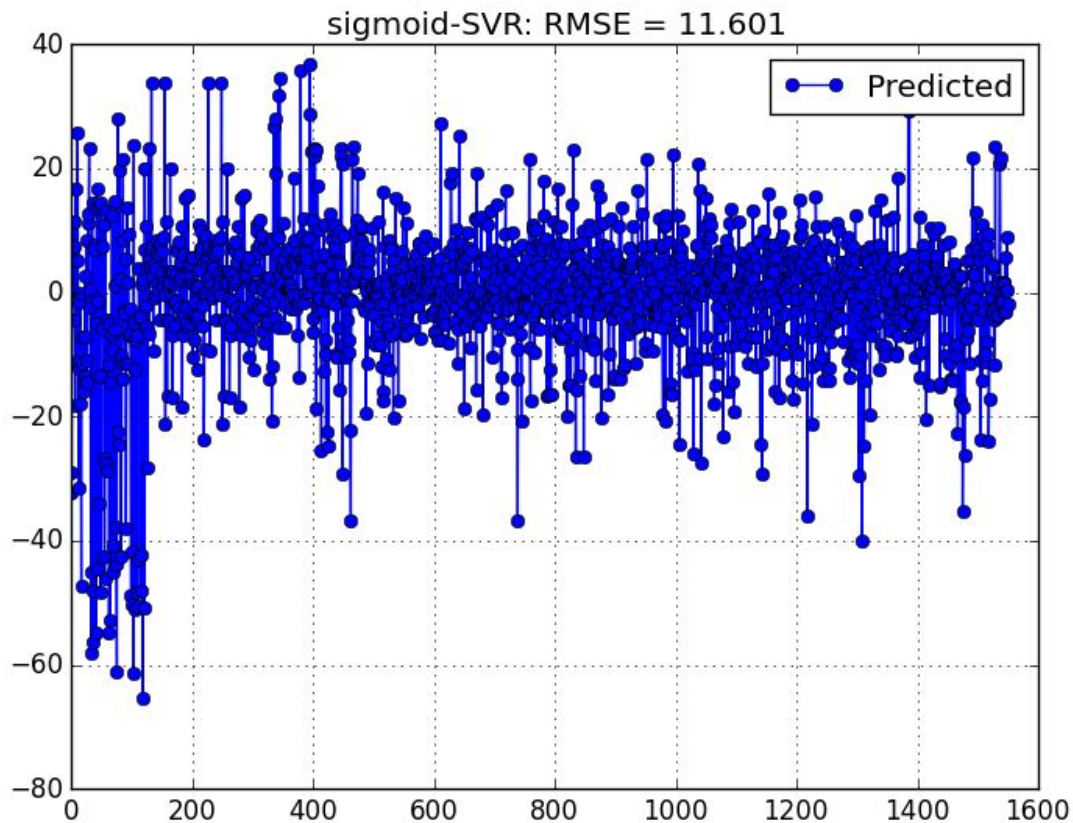












Conclusions

This research develops 3 kinds of Support Vector Regression (SVR) Application Program Interface (API), Linear-SVR and RBF-SVR as well as Sigmoid-SVR. Of these SVRs, Sigmoid-SVR and RBF-SVR are popular, and this research compares the two with Linear-SVR in terms of robustness and accuracy since Linear-SVR owns the smallest Root Mean Square Error (RMSE) value, however, Sigmoid-SVR and RBF-SVR perform overfitting.

Forthcoming Research

We tentatively conclude that sentiment analysis for Twitter data is not that different from sentiment analysis for other genres. In future work, we will explore even richer linguistic analysis, for example, parsing, semantic analysis and topic modeling. estimate a tweet's sentiment by counting the number of occurrences of "positive" and "negative" words.

Appendix

```
#!/usr/bin/python
```

```

# -*- coding: utf-8 -*-

"""
=====
Twitter United Airlines Multivariate Sentiment Analysis Based on Support Vector Machine
=====
@author: Yuanhui Yang
@email: yuanhui.yang@u.northwestern.edu
=====
"""
print(__doc__)


from types import *
import numpy as np
from sklearn.svm import SVR
from sklearn.grid_search import GridSearchCV
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.feature_extraction import DictVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn.grid_search import GridSearchCV
from sklearn.pipeline import Pipeline
from math import sqrt
import matplotlib.pyplot as plt
# import gensim, logging
import pandas as pd


class Data:
    def __init__(self, csvfilepath):
        self.feature()
        self.dataType()
        self.raw_table = pd.read_csv(csvfilepath, usecols=self.feature,
skip_blank_lines=True, dtype=self.data_type, keep_default_na=True)
        self.dropNaN()

```

```

        self.setOutput()
        self.setInput()
        self.preprocess()
        self.svr_linear()
        self.svr_rbf()
        self.svr_sigmoid()

    def printRawTable(self):
        print self.raw_table
        return self.raw_table

    def feature(self):
        self.feature = ['airline_sentiment', 'airline_sentiment_confidence',
            'airline', 'text', 'tweet_created', 'tweet_location', 'user_timezone']
        return self.feature

    def dataType(self):
        self.data_type = {'airline_sentiment': str, 'airline_sentiment_confidence':
np.float64, 'airline': str, 'text': str, 'tweet_created': str, 'tweet_location': str,
'user_timezone': str}

    def dropNaN(self):
        self.raw_table = self.raw_table.dropna(axis=0, how='any', thresh=None,
subset=None, inplace=False)
        self.raw_table = self.raw_table.dropna(axis=1, how='any', thresh=None,
subset=None, inplace=False)
        return self.raw_table

    def setOutput(self):
        airline_sentiment = np.array(self.raw_table['airline_sentiment'])
        airline_sentiment_confidence =
np.array(self.raw_table['airline_sentiment_confidence'])
        self.length_of_sequence = len(airline_sentiment)
        self.output = np.zeros(self.length_of_sequence)
        for idx in range(self.length_of_sequence):
            if airline_sentiment[idx] == 'neutral':
                self.output[idx] = np.float64(0.5 *
airline_sentiment_confidence[idx])
            elif airline_sentiment[idx] == 'positive':
                self.output[idx] = np.float64(1.0 *
airline_sentiment_confidence[idx])
            else:
                self.output[idx] = np.float64(0.0)

```

```

pd.DataFrame(self.output).to_csv('../data/Output.csv', sep=',')
return self.output

def setInput(self):
    self.text2Vector()
    self.airline2Vector()
    self.tweetCreated2Vector()
    self.tweetlocation2Vector()
    self.userTimezone2Vector()
    self.input = np.column_stack((self.airline2Vector, self.text2Vector,
self.tweetCreated2Vector, self.tweetlocation2Vector, self.userTimezone2Vector))
    pd.DataFrame(self.input).to_csv('../data/Input.csv', sep=',')
    return self.input

def preprocess(self):
    self.length_of_prediction_sequence = int(0.2 * len(self.input))
    self.input_scaler = preprocessing.StandardScaler()
    self.input_transform = self.input_scaler.fit_transform(self.input)
    self.output_scaler = preprocessing.StandardScaler()
    self.output_transform = self.output_scaler.fit_transform(self.output)
    self.input_transform_train = self.input_transform[: (self.length_of_sequence
- self.length_of_prediction_sequence)]
    self.output_transform_train =
self.output_transform[: (self.length_of_sequence - self.length_of_prediction_sequence)]
    self.input_transform_test = self.input_transform[(self.length_of_sequence -
self.length_of_prediction_sequence):]
    self.output_test = self.output[(self.length_of_sequence -
self.length_of_prediction_sequence):]
    # print len(self.input_transform_test)
    # print len(self.input_transform_train)

def svr_linear(self):
    self.svr_linear = GridSearchCV(SVR(kernel='linear', gamma=0.1), cv=10,
param_grid={'C': np.logspace(-10.0, 10.0, num=5, base=2.0), 'gamma': np.logspace(-10.0,
10.0, num=5, base=2.0)})
    # self.svr_linear = SVR(kernel='linear', gamma=0.1)
    self.svr_linear.fit(self.input_transform_train, self.output_transform_train)
    self.output_transform_predict_linear =
self.svr_linear.predict(self.input_transform_test)

```

```

        self.output_predict_linear =
self.output_scaler.inverse_transform(self.output_transform_predict_linear.reshape((self.length_of_prediction_sequence, 1)))

        self.square_root_of_mean_squared_error_linear =
sqrt(mean_squared_error(self.output_test, self.output_predict_linear))

        plt.figure()
        x = np.arange(0, self.length_of_prediction_sequence)
        plt.plot(x, self.output_test, 'ro-', label='Actual')
        plt.plot(x, self.output_predict_linear, 'bo-', label='Predicted')
        plt.title('linear-SVR: RMSE = %.3f'
%self.square_root_of_mean_squared_error_linear)

        plt.legend()
        plt.grid(True)
        plt.savefig('../figure/linear-SVR.eps', format='eps', dpi=1000)
        plt.savefig('../figure/linear-SVR.png', format='png', dpi=1000)
        # plt.show()

def svr_rbf(self):
    self.svr_rbf = GridSearchCV(SVR(kernel='rbf', gamma=0.1), cv=10,
param_grid={'C': np.logspace(-10.0, 10.0, num=5, base=2.0), 'gamma': np.logspace(-10.0,
10.0, num=5, base=2.0)})
    # self.svr_rbf = SVR(kernel='rbf', gamma=0.1)
    self.svr_rbf.fit(self.input_transform_train, self.output_transform_train)
    self.output_transform_predict_rbf =
self.svr_rbf.predict(self.input_transform_test)
    self.output_predict_rbf =
self.output_scaler.inverse_transform(self.output_transform_predict_rbf.reshape((self.length_of_prediction_sequence, 1)))

    self.square_root_of_mean_squared_error_rbf =
sqrt(mean_squared_error(self.output_test, self.output_predict_rbf))

    plt.figure()
    x = np.arange(0, self.length_of_prediction_sequence)
    plt.plot(x, self.output_test, 'ro-', label='Actual')
    plt.plot(x, self.output_predict_rbf, 'bo-', label='Predicted')
    plt.title('rbf-SVR: RMSE = %.3f'
%self.square_root_of_mean_squared_error_rbf)

    plt.legend()
    plt.grid(True)
    plt.savefig('../figure/rbf-SVR.eps', format='eps', dpi=1000)
    plt.savefig('../figure/rbf-SVR.png', format='png', dpi=1000)
    # plt.show()

```

```

def svr_sigmoid(self):
    self.svr_sigmoid = GridSearchCV(SVR(kernel='sigmoid', gamma=0.1), cv=10,
    param_grid={'C': np.logspace(-10.0, 10.0, num=5, base=2.0), 'gamma': np.logspace(-10.0,
    10.0, num=5, base=2.0)})
    # self.svr_sigmoid = SVR(kernel='sigmoid', gamma=0.1)
    self.svr_sigmoid.fit(self.input_transform_train,
self.output_transform_train)
    self.output_transform_predict_sigmoid =
self.svr_sigmoid.predict(self.input_transform_test)
    self.output_predict_sigmoid =
self.output_scaler.inverse_transform(self.output_transform_predict_sigmoid.reshape((self.le
ngth_of_prediction_sequence, 1)))
    self.square_root_of_mean_squared_error_sigmoid =
sqrt(mean_squared_error(self.output_test, self.output_predict_sigmoid))
    plt.figure()
    x = np.arange(0, self.length_of_prediction_sequence)
    plt.plot(x, self.output_test, 'ro-', label='Actual')
    plt.plot(x, self.output_predict_sigmoid, 'bo-', label='Predicted')
    plt.title('sigmoid-SVR: RMSE = %.3f'
%self.square_root_of_mean_squared_error_sigmoid)
    plt.legend()
    plt.grid(True)
    plt.savefig('../figure/sigmoid-SVR.eps', format='eps', dpi=1000)
    plt.savefig('../figure/sigmoid-SVR.png', format='png', dpi=1000)
    # plt.show()

def word2Vector(self, item, num_feature):
    corpus = np.array(self.raw_table[item])
    hashingVectorizer = HashingVectorizer(decode_error='ignore',
n_features=num_feature, non_negative=False)
    word2Vector = hashingVectorizer.fit_transform(corpus).toarray()
    return word2Vector
    # pd.DataFrame(word2Vector).to_csv('../data/result.csv', sep=',',
encoding='utf-8',)
    # print self.vector.ravel()
    # print type(corpus[0])
    # print corpus
    # self.vectorizer = TfidfVectorizer(min_df=1, max_df=1.0,
stop_words='english', max_features=10, norm='l2', sublinear_tf=True)

```

```

        # print self.vectorizer
        # vec = self.vectorizer.fit_transform(corpus).toarray()
        # print vec.toarray()
# def setInput(self):
# def airline2Vector(self):
#     corpus = np.array(self.raw_table['airline'])
#     self.dictVectorizer = DictVectorizer()
#     self.airline2Vector = self.dictVectorizer.fit_transform(corpus).toarray()
#     print self.airline2Vector
def airline2Vector(self):
    corpus = np.array(self.raw_table['airline'])
    set_corpus = set(corpus)
    list_corpus = list(set_corpus)
    # print array_corpus
    length_row = len(corpus)
    length_column = len(set_corpus)
    self.airline2Vector = np.zeros((length_row, length_column))
    # print corpus
    for i in range(length_row):
        for j in range(length_column):
            if corpus[i] == list_corpus[j]:
                self.airline2Vector[i][j] = 1.0
pd.DataFrame(self.airline2Vector).to_csv('../data/Airline2Vector.csv',
sep=',')

    return self.airline2Vector

def text2Vector(self):
    self.text2Vector = self.word2Vector('text', 30)
    pd.DataFrame(self.text2Vector).to_csv('../data/Text2Vector.csv', sep=',')
    return self.text2Vector

def tweetCreated2Vector(self):
    self.tweetCreated2Vector = self.word2Vector('tweet_created', 10)
pd.DataFrame(self.tweetCreated2Vector).to_csv('../data/TweetCreated2Vector.csv', sep=',')
    return self.tweetCreated2Vector

def tweetlocation2Vector(self):
    self.tweetlocation2Vector = self.word2Vector('tweet_location', 10)

```

```
pd.DataFrame(self.tweetlocation2Vector).to_csv('../data/TweetLocation2Vector.csv', sep=',')
    return self.tweetlocation2Vector

    def userTimezone2Vector(self):
        self.userTimezone2Vector = self.word2Vector('user_timezone', 10)

pd.DataFrame(self.userTimezone2Vector).to_csv('../data/UserTimezone2Vector.csv', sep=',')
    return self.userTimezone2Vector

def main():
    csvfilepath = '../data/Tweets.csv'
    data = Data(csvfilepath)

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