Twitter User Analyzer and Classifier

Software Engineering Project

Our Team

ASWANTH - IIT2016105

ASWIN VB - IIT2016106

DRUVAL CR - BIM2016501

SARATH CHANDRA - IRM2016006

UTPAL AMAN - IIT2016133

INTRODUCTION

We analyze the tweets of several users in twitter based on several parameters that we are going to discuss further to predict whether a user is **Anomalous, Suspected to be Anomalous** or **Non Anomalous.** And we apply several **Classification** Algorithms based on these results.

The parameters used for user analysis

User can be :

1. Anomalous
2. Suspected
3. Non Anomalous

FIVE PARAMETERS

1. Alexa URL Ranking
2. Similarity of Tweets
3. Time difference between tweets
4. Malware content
5. Adult content

Alexa URL Ranking

- **Alexa URL Ranking** is to check how genuine a URL is.
- We are using **Alexa** Website which provides a rank for each URL submitted.
- For each user we will fetch all the URLs present in his/her tweet’s.
- We submit it to the Alexa website and by Web Scraping we obtain the rank of the URL.
- If the rank is less than 2,00,000 then the **counter** is increased.
- Using this **counter** we determine the Alexa URL Rank(which is out of **10**) of the user using the following Formula :

$$\text{Ratio} = \frac{\text{Number of URL with rank less than 2,00,000}}{\text{Total number of URLs posted}} * 10$$

The Code Implementation

```
def url_ranking(urls,counter=0,total=0):
    if(len(urls)==0):
        return [counter,total]
    for url in urls:
        try:
            r=requests.get(url)
            url=r.url
            parts=url.split("/")
            if(parts[2]=="twitter.com"):
                total=total+1
            elif(url!="https://t.co/"):
                print(url)
                if(bs4.BeautifulSoup(urlopen("http://data.alexa.com/data?
                    rank=bs4.BeautifulSoup(urlopen("http://data.alexa.com
                    print("rank : "+str(rank))
                    if(int(rank)>200000):
                        counter=counter+1
                    total=total+1
            else:
                total=total+1
        except:
            print("cannot check url")

    return [counter,total]
```

Similarity of Tweets

- In this we check full tweet and not only URL. We get all the tweets and check each tweet with its 3 previous and 3 next tweets.
- This forms the cluster of 7 tweets. In this cluster if there exist any **two tweets** that are **75% similar** then we increase the **counter**.
- Using this **counter** we calculate the Similarity Rank for the User which is calculated by the following Formula.

$$\text{Ratio} = \frac{\text{Counter}}{\text{Total number of clusters}} * 10$$

The Code Implementation

```
def rank_similarity(dataset):
    counter=0
    total=0
    for i in range(2,len(dataset)-3):
        total=total+1
        hitinCluster = 0
        for j in range(i-3,i+4):
            if(j>0 and j<len(dataset)):
                for k in range(j+1,i+4):
                    if(j!=k and k>0 and k<len(dataset)):
                        if(cosine_sim(dataset[j],dataset[k])):
                            counter = counter+1
                            hitinCluster = 1
                            break
            if(hitinCluster==1):
                break

    if(total==0):
        return 0
    print(counter," of ",total," clusters are similar")
    similarity_rank=(float(counter)/total)*10
    return similarity_rank
```

Time difference between Tweets

- In this we check full tweet and not only URL. We get all the tweets and check each tweet with its 3 previous and 3 next tweets.
- This forms the cluster of 7 tweets. In this cluster if there exist any **5 tweets** that occured within the span of **5 minutes** then we increase the **counter**.
- Using this **counter** we calculate the Time Difference Rank for the User which is calculated by the following Formula.

$$\text{Ratio} = \frac{\text{Counter}}{\text{Total number of clusters}} * 10$$

The Code Implementation

```python
def rank_time(dataset):
    cluster=0
    hits=0
    if(len(dataset) < 7):   #cannot predict
        return 0
    #7 tweets is taken as a cluster
    for i in range(0,len(dataset)-6):
        for j in range(0,3):
            if(findTimeDiff(dataset,i+j,i+j+4)):
                hits = hits+1
                break
    cluster=len(dataset)-6

    rank = (float(hits)/cluster)*10
    return rank
```

Malware Content

- We use Web Of Trust (WOT) for the checking the reputation of a URL. It tells us a whether a URL is good or bad.
- We collect all the URLs from a User’s tweets and then pass it to the WOT API
- The API sends back the result in either of the 4 categories, that is, Negative, Questionable, Neutral or Positive.
- If the API returns that a URL is Negative, Questionable or Neutral then we Increase the Counter. Using this counter we can obtain the WOT Malware rank of the user. If the User has **5% tweets** containing malicious URL then rank is **10** otherwise it is **0.**

Ratio = 10 (If 5% of tweets are malicious)
Ratio = 0 (If less than 5% of tweets are malicious)

The Code Implementation

```
def rank_wot(dataset):
    counter=0
    total=0
    for data in dataset:
        mal=False
        urls=fetch_url(data)
        total=total+1
        for url in urls:
            if mal:
                break
            try:
                r=requests.get(url)
                url=r.url
                report = wot_reports_for_domains([url], KEY)

                # print(parse_attributes_for_report(report))
                for key in report:
                    print(url," rank is ",report[key]['0'][1])
                    if(report and report[key] and report[key]['0'][1]<40):
                        counter=counter+1
                        mal=True
                        break
            except:
                print("cannot check")
    print(counter," ",len(dataset))
    WOT_RANK=(float(counter)/len(dataset))*100
    if(WOT_RANK<5):
        return 0
    return 10
```

Adult Content

- In this we fetch the URL that the user has shared in his tweet. We made the dataset of all the URLs that are possible to have adult content.
- Since user post shortened URL on twitter so we first expand the URL that we have fetched and then check each of the URL with the URL stored in the dataset.
- If the URL is found to contain adult content then the user is directly declared an anomaly. We consider a URL containing adult content to be anomalous,
- So if the URL is found to contain adult content then it is directly declared as Anomalous so ratio is either 10 or 0 as specified by following equation.

$$\text{Ratio} = 10 \text{ (If adult content is present)}$$
$$\text{Ratio} = 0 \quad \text{(If no adult content is present)}$$

The Code Implementation

```
def checkAdultContent(dataset):
    adultContentDataset=None
    try:
        adultContentDataset = pd.read_csv(dirpath+'adultcontenturl.csv')
    except:
        adultContentDataset = pd.read_csv('adultcontenturl.csv')
    adultContentDataset = adultContentDataset.iloc[0:3,0].values
    urlExpand = UrlExpand()

    if(len(dataset) == 0):
        return 0
    for data in dataset:
        urls=fetch_url(data)
        for url in urls:
            try:
                r=requests.get(url)
                url=r.url
                if(url=="https://t.co/"):
                    continue
                print("checking url : "+url)
                result = urlExpand.decodeURL(url)
                if(result in adultContentDataset):
                    #returns 10, if adult content is present
                    return 10
            except:
                print("Invalid url")


    #returns 0, if adult content isn't present
    return 0
```

FAL VALUE

(Final Anomaly Level)

A twitter user is classified into Anomalous, Non Anomalous and Intermediate using 5 parameters and each of these parameter will be given a rank:

- Time Difference (denoted by a)
- Similarity of Tweets (b)
- URL Ranking (c)
- Malware URL (d)
- Adult Content (e)

Each of these parameter will be assigned a value from 1-10 for each user and these parameters have a weight which together will decide whether a user is anomalous or not

Weights of each parameter are :

- Time Difference: 0.15
- Similarity of Tweets: 0.25
- URL Ranking: 0.30
- Malware URL: 0.30
- Adult Content: 1

Calculating FAL Value

- If the value of **e(Adult content Ranking) is 10** the **FAL Value = 10**
- Otherwise we follow calculate FAL value using the Following Equation

FAL Value = a*0.15 + b*0.25 + c*0.3 + d*0.3

ANALYZING

- **FAL value** is between **0 - 3** then user is **Non Anomalous**
- **FAL value** is between **4 - 5** then user is **Suspected**
- **FAL value** is between **6 - 10** then user is **Anomalous**

Classification

This algorithm for finding FAL value is run on a dataset containing twitter handles which will, in turn, produce a dataset of the a,b,c,d,e and FAL values upon which classification algorithms can be applied.

Classification Algorithms used are:

- K-nearest neighbors (**KNN**)
- **Naive Bayes** classifier
- Random Forest
- Support Vector Machine (**SVM**)
- Decision Tree

The result of these classification algorithms will be a confusion matrix which is often used to **describe the performance of a classification model** (or "classifier") on a set of test data for which the true values are known.

CONFUSION MATRIX :

		predicted class		
		class 1	class 2	class 3
actual class	class 1	True positives		
	class 2		True positives	
	class 3			True positives

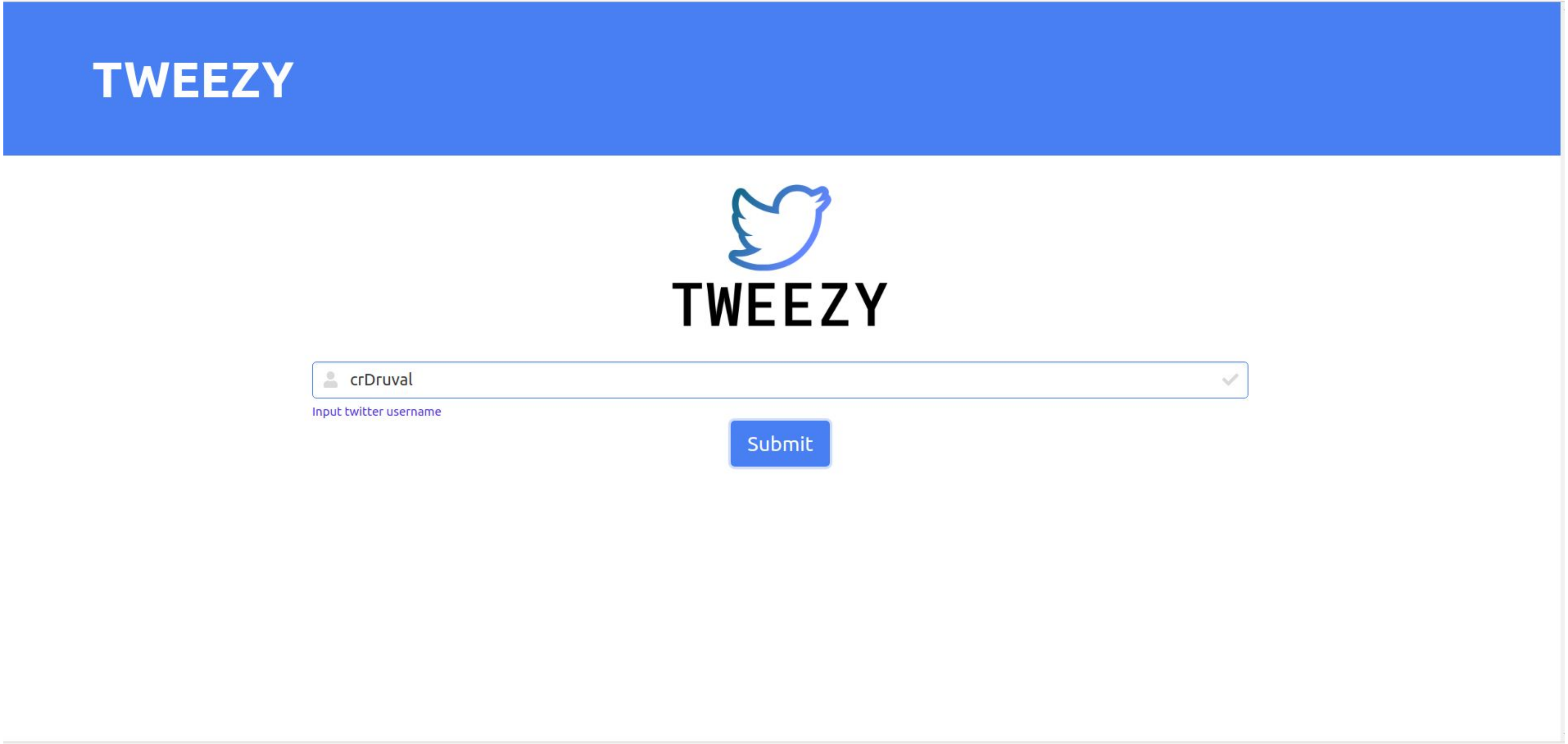
```
[[4 0 0]
 [2 0 0]
 [0 0 1]]
Accuracy : 71.42857142857143
```

What can we learn from this matrix?

- There are three possible predicted classes: "Anomalous", “Suspected” and "Non-Anomalous".
- The confusion matrix in Right image consist a total of 7 predictions (e.g., 7 twitter handles were being tested for the presence of Anomaly).
- Out of those 8 handles, the classifier predicted "Anomalous" 6 (4+2) times, and "Non-Anomalous" 1 time.
- In reality, 4 handles in the sample were Anomalous and 1 was Non-Anomalous and 2 were suspected category.

```python
def NaiveBayesClassifier(dataset):
    # Importing the dataset
    X = dataset.iloc[:, [0,1,2,3,4]].values
    y = dataset.iloc[:, 6].values

    # Splitting the dataset into the Training set and Test set
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

    # Feature Scaling
    from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)

    # Fitting classifier to the Training set
    # Create your classifier here
    classifier=GaussianNB()
    classifier.fit(X_train,y_train)

    # Predicting the Test set results
    y_pred = classifier.predict(X_test)

    # Making the Confusion Matrix
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test, y_pred)

    # print(cm)
    return cm
```

Screenshots


TWEEZY
TWEEZY
crDruval
Input twitter username
Submit

URL RANK	SIMILARITY RANK	WOT RANK		TIME RANK	ADULT CONTENT RANK	FAL VALUE
5.05	10.0	10.0		5.45	0.0	7.83

DA_Green_Agent

Anomalous

FAL VALUE = 7.83/10

URL RANK	SIMILARITY RANK	WOT RANK		TIME RANK	ADULT CONTENT RANK	FAL VALUE
0.4	0.0	10.0	**benaffleck**	0.0	0.0	3.12

Non Anomalous

FAL VALUE = 3.12/10

```
KNN Classification
==================
[[4 0 0]
 [2 0 0]
 [1 0 0]]
Accuracy : 57.14285714285714

Naive Bayes Classification
==========================
[[4 0 0]
 [2 0 0]
 [0 0 1]]
Accuracy : 71.42857142857143

Decistion Tree Classification
=============================
[[4 0 0]
 [2 0 0]
 [0 0 1]]
Accuracy : 71.42857142857143

Random Forest Classification
============================
[[4 0 0]
 [2 0 0]
 [0 0 1]]
Accuracy : 71.42857142857143

SVM Classification
==================
[[4 0 0]
 [2 0 0]
 [0 0 1]]
Accuracy : 71.42857142857143
```

References

- https://ieeexplore.ieee.org/document/8204141
- https://www.mywot.com/
- https://www.iplocation.net/alexa-traffic-rank
- https://en.wikipedia.org/wiki/Web_of_trust

THANK YOU