



Hype zum Release von „World of Warcraft – Legion“ am 30.08.2016

In welchen Ländern wird das Release von „World of Warcraft – Legion“ am meisten über Twitter gehyped und welche Sprachen werden am häufigsten verwendet?

**Dokumentation zum Projekt des Seminars
I4 Angewandte Informationswissenschaft**

von

**Thorsten Brückner
Matrikelnummer: 2060488**

1. Einleitung

„World of Warcraft“ ist im Bereich „MMORPG“ mit Abstand das erfolgreichste Spiel aller Zeiten und wohl auch das bekannteste der legendären „Warcraft“ Geschichte. Zu Beginn glänzte „World of Warcraft“ mit über 12 Millionen Abonnenten. Der Publisher Blizzard Entertainment hat allerdings in der letzten Zeit schwer zu knabbern, denn das letzte Addon „Warlords of Draenor“, schrieb keine guten Zahlen und erntete Spott und Hohn der Community. Jetzt versucht man das „sinkende Schiff“ mit Hilfe der Kollegen von der Diablo-Reihe aus dem Wasser zu ziehen. Vorarbeitet leistete der gerade veröffentlichte Kinofilm „Warcraft“, der vor allem in Deutschland und Asien extrem gut gestartet ist. Diesen Anlass nutzt Blizzard Entertainment um am 30.08.2016 das 6. Addon der Spielereihe zu hypen und World of Warcraft zu neuem Glanz zu verhelfen. Bereits vor dem Release von „Legion“ war der Hype durch die Rückkehr von Illidan Stormrage Asien sehr groß. Schlagzeilen machten Künstler in Taiwan welche Illidan auf einer Hauswand aufmalten.



Als Student für Informationswissenschaften, sowie langjähriger „Warcraft“ Fan und „World of Warcraft“ Spieler, erfasste mich dieser „Hype-Train“ natürlich auch und es bot sich daher als Thema für das Projektseminar I4 Angewandte Informationswissenschaft an. Mir kam die Idee, den Hype zum Release von „Legion“ via Twitter zu verfolgen und zu analysieren. Die Fragestellung lautet:

„Hype zum Release von „World of Warcraft – Legion“ am 30.08.2016 In welchen Ländern wird das Release von „World of Warcraft – Legion“ am meisten über Twitter gehyped und welche Sprachen werden am häufigsten verwendet?“

Zur Analyse bieten sich nun Twitter Crawler, sowie die Python-Bibliotheken zur grafischen Darstellung an. Mit Hilfe eines Twitter-Streams, welcher sämtliche Tweets zu den Hashtags #Legion, #WorldofWarcraft und #Warcraft abfängt, soll eine Woche lang alles weltweit live getwitterte gesammelt und später analysiert werden. Um die gesammelten Daten zu analysieren bedarf es weiterer Hilfsmittel. Die Idee war, sämtliche Tweets zu segmentieren und in drei Bereiche aufzuteilen. Für mein Thema war es mir wichtig drei Aspekte hervorzuheben. „In welchen Sprachen wird am meisten getwittert?“, „Aus welchen Ländern kommen die meisten Tweets?“ und „Gibt es genug Geo-Daten der Tweets um diese grafisch darzustellen?“. Wie lässt sich das nun umsetzen? Für die Darstellung der Diagramme mit denen ich die Häufigkeit der Länder und Sprachen repräsentieren, habe ich mich für Python Bokeh entschlossen. Die Geo-Tags werden mit Hilfe von Matplotlib und Basemap auf einer Weltkarte eingezeichnet.

2. Methoden

Zuerst brauchte ich einen Zugang zu der Twitter API. Das erreichte ich indem ich mit einem Twitter-Account die Seite <https://apps.twitter.com/> besuche und eine neue „APP“ erstellte.

Create an application

Application Details

Name *

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.



Developer Agreement

☐ Yes, I have read and agree to the [Twitter Developer Agreement](#).

Habe ich die nötigen Angaben gemacht, komme ich zum nächsten Schritt. Den Oauth settings.

OAuth settings

Your application's OAuth settings. Keep the "Consumer secret" a secret. This key should never be human-readable in your application.

Access level	Read-only About the application permission model
Consumer key	
Consumer secret	
Request token URL	https://api.twitter.com/oauth/request_token
Authorize URL	https://api.twitter.com/oauth/authorize
Access token URL	https://api.twitter.com/oauth/access_token
Callback URL	None
Sign in with Twitter	No

Your access token

It looks like you haven't authorized this application for your own Twitter account yet. For your convenience, we give you the opportunity to create your OAuth access token here, so you can start signing your requests right away. The access token generated will reflect your application's current permission level.


[Create my access token](#)

Hier erstelle ich die benötigten access tokens welche ich dann beim nächsten Schritt erhalte um meinen Crawler mit der Twitter-API zu verbinden.


[Details](#) [Settings](#) [OAuth tool](#) [@Anywhere domains](#) [Reset keys](#) [Delete](#)

OAuth Settings

Consumer key: *




Consumer secret: *

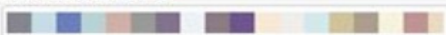


Remember this should not be shared.

Access token: *



Access token secret: *



Remember this should not be shared.

Request Settings

Request type: *

☒ GET

☐ POST

Diese Werte werden in der Datei twitter_keys.py eingetragen.

```
1 # Consumer keys and access tokens, used for OAuth
2 # https://apps.twitter.com
3 # Erstelle Dir eine App und verwende die Keys hier
4 consumer_key = ''
5 consumer_secret = ''
6 access_token = ''
7 access_token_secret = ''
8
```

Der Crawler

Die Datei twitter_keys.py ist ausschließlich für die OAuth settings zuständig und stellt die Verbindung zwischen den zwei Crawlern und der Twitter API her.

Für mein Vorhaben, habe ich mich für einen Stream-Crawler entschieden, da ich gerne die kompletten Tweets zu den hashtags #warcraft, #legion und #worldofwarcraft eine Woche vor dem, sowie am 30.08.2016 haben wollte. Der Stream musste für diesen Zweck 7 Tage dauerhaft laufen.

In der save_tweets_stream.py (dem Crawler) wird festgelegt, welche hashtags ich haben möchte.

#warcraft, #legion und #worldofwarcraft waren in diesem Fall meine gewünschten hashtags.

```
10
11 def start_stream():
12     api = tweepy.streaming.Stream(auth, TwitterStreamListener(), timeout=60, compression=True, wait_on_rate_limit=True)
13     api.filter(follow = None, track = ['#Warcraft', '#Legion', '#WorldOfWarcraft'])
14     return
```

Außerdem wähle ich auch die benötigten „fields“ aus, welche in meiner Ergebnistabelle gespeichert werden sollen. Gleichzeitig werden in diesem Codeabschnitt auch die Tweetergebnisse in die Datei „result.csv“ geschrieben.

```
28 with open('result.csv', 'ab') as resultFile:
29     writer = csv.writer(resultFile,
30                         delimiter=";",
31                         lineterminator="\r\n",
32                         encoding='utf-8')
33
34     if tweet.place and tweet.user.location:
35         writer.writerow([tweet.id_str, tweet.created_at, tweet.text, tweet.lang, tweet.place.bounding_box.coordinates, tweet.user.location])
36     elif ((not tweet.place) and (tweet.user.location)):
37         writer.writerow([tweet.id_str, tweet.created_at, tweet.text, tweet.lang, 'NaN', tweet.user.location])
38     elif ((tweet.place) and (not tweet.user.location)):
39         writer.writerow([tweet.id_str, tweet.created_at, tweet.text, tweet.lang, tweet.place.bounding_box.coordinates, 'NaN'])
40     else:
41         writer.writerow([tweet.id_str, tweet.created_at, tweet.text, tweet.lang, 'NaN', 'NaN'])
42
43     print (TwitterStreamListener.tweet_count)
44     return
```

Da ich diesen Crawler in Zusammenarbeit mit Raphael Katschke geschrieben hatte und wir beide die Ergebnisse vom Stream benötigten, konnten wir hier unsere gewünschten „fields“ zusammen wählen. Wir entschlossen uns für folgende „fields“

- tweet_id (die ID des tweets)
- tweet_created_at (Erstelldatum des tweets)
- tweet.text (Inhalt des tweets)
- tweet.lang (Sprache des tweets, die von Twitter selbst erkannt wird)
- tweet_place sowie tweet.user.location (Ortsangaben im Profil des Tweeterstellers)
- tweet.place.bounding_box.coordinates (Mitgelieferte GEO-Tags falls vorhanden)

Die Main.py

Die erfolgreich gefetchten tweets sollen nun mit Hilfe von verschiedenen Python Bibliotheken analysiert und dargestellt werden. Gestartet wird über die Main.py, welche folgenden Code dafür beinhaltet:

```

9  if __name__ == "__main__":
10     #creating instances of imported classes
11     read_data = CSVDataReader()
12     langdist = langDist()
13     landdist = landDist()
14     ldd = langDetectDist()
15
16
17     data = read_data.FileReader('result.csv') #reading results
18     #running the methods of imported classes
19     mapper = geomapping()
20     mapper.mapping(data[1], data[4])
21
22     langdist.dist(data[1])
23     ldd.dist(data[0])
24     usa = landdist.reader('states.csv')
25     germany = landdist.reader('germany.csv')
26     uk = landdist.reader('uk.csv')
27     france = landdist.reader('france.csv')
28     spain = landdist.reader('spain.csv')
29     canada = landdist.reader('canada.csv')
30     russia = landdist.reader('russia.csv')
31     landdist.dist(data[5], usa, germany, uk, france, spain, canada, russia)
32

```

Diese Datei kreiert Instanzen der Importierten Klassen und ruft nach und nach die Methoden auf und liefert deren Ergebnisse.

Die Langdist.py

Zuerst wollte ich mich um die Sprache bemühen. Twitter liefert zum einen die erkannte Sprache mit. Dies geschieht mit dem bereits erwähnten Feld „tweet.lang“. Die Spracherkennung von Twitter gilt als sehr schnell, dafür nicht unbedingt zu 100% genau. Die Spracherkennung von Twitter wird mit Hilfe dieses Codes der langdist.py durchgeführt.

```

18  for value in data.values():
19      self.langs.append(value)
20
21  self.freq = FreqDist(self.langs).most_common()
22  for elem in self.freq:
23      (self.key, self.val) = elem
24      if self.val < 500: #write to rest
25          self.dumpV.append(self.val)
26      else:
27          self.names.append(self.key+' '+str(self.val))
28          self.values.append(self.val)
29
30  for dump in self.dumpV:
31      self.v += dump
32

```


Hierbei wird verglichen, ob der Tweet der gerade überprüft wird, mit einer bekannten Sprache gematched werden kann. Ist das der Fall, wird er markiert, gezählt und der entsprechenden Sprache für die Grafik hinzugefügt. Danach wird mit Bokeh das Diagramm gezeichnet.

```
33     self.names.append('rest: '+str(self.v)) #dictionaries
34     self.values.append(self.v)
35
36     self.dicts['labels'] = self.names
37     self.dicts['vals'] = self.values
38
39     self.graph = Donut(self.dicts, values='vals', label='labels', title='Language Frequency Distribution') #draw donut with bokeh
40     output_file("language_distribution.html")
41     show(self.graph)
42
43     return
44
```

Langdetectdist.py

Da ich allerdings „langdetect“ im Projektplan vorgesehen hatte, lasse ich trotz der Überprüfung von Twitter selbst, noch einmal alle Tweets durch die langdetectdist.py nachprüfen. Dieser Vorgang ähnelt dem der langdist.py, beinhaltet aber Unterschiede.

```
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
for value in data.values(): #parse tweets and try to detect the language via langdetect and name the language
    try:
        self.langs.append(detect(value))
    except:
        self.langs.append('undetected') #if detection fails name it undetected
self.freq = FreqDist(self.langs).most_common() #frequency of languages and sort it
for elem in self.freq: #parse entries and write language and frequency value
    (self.key, self.val) = elem
    if self.val < 500: #write to rest
        self.dumpV.append(self.val)
    else:
        self.names.append(self.key+' '+str(self.val)) #write to dict for graph
        self.values.append(self.val)
```

Und zwar wird hier jeder gefetchte Tweet nachträglich überprüft. Das heißt, ich benutze langdetect.py um bei jedem Tweet separat zu checken, ob die Sprache bekannt ist oder nicht. Ist das der Fall, wird wie bei Twitters Methode on the Fly, die Sprache markiert und eingetragen. Ist dies nicht der Fall, wird die Sprache als „undetected“ markiert und zum Rest hinzugefügt. Diese Methode ist zuverlässiger, kostet aber sehr viel Zeit. Dieses Ergebnis wird ebenfalls als Kuchendiagramm über Bokeh ausgegeben um den Vergleich zwischen beiden Methoden deutlicher zu präsentieren.

Die Landdist.py

Des Weiteren möchte ich natürlich gerne noch erfahren, aus welchen Ländern denn die ganzen tweets getwittert werden. Dazu bedarf es allerdings eine Menge mehr. Dazu benötige ich hier die Ortsangaben des Tweeterstellers. Das Problem hierbei besteht darin, dass wir viele Fantasienamen, sowie mal Städte und Länder einzeln oder auch in Kombination haben. Die für mich plausibelste Lösung hierfür besteht darin, Listen mit sämtlichen größeren Städten der verschiedenen Länder zu erstellen und diese mit allen Tweets abzugleichen und zu kategorisieren.

Diese Listen sehen so aus (Listenbeispiel USA):

	A	B	C	D	E
1	Alabama	Ala.	AL	Montgomery	
2	Alaska	Alaska	AK	Juneau	
3	Arizona	Ariz.	AZ	Phoenix	
4	Arkansas	Ark.	AR	Little Rock	
5	California	Calif.	CA	Sacramento	
6	Colorado	Colo.	CO	Denver	
7	Connecticut	Conn.	CT	Hartford	
8	Delaware	Del.	DE	Dover	
9	Florida	Fla.	FL	Tallahassee	
10	Georgia	Ga.	GA	Atlanta	
11	Hawaii	Hawaii	HI	Honolulu	
12	Idaho	Idaho	ID	Boise	
13	Illinois	Ill.	IL	Springfield	
14	Indiana	Ind.	IN	Indianapolis	
15	Iowa	Iowa	IA	Des Moines	
16	Kansas	Kans.	KS	Topeka	
17	Kentucky	Ky.	KY	Frankfort	
18	Louisiana	La.	LA	Baton Rouge	
19	Maine	Maine	ME	Augusta	
20	Maryland	Md.	MD	Annapolis	
21	Massachusetts	Mass.	MA	Boston	
22	Michigan	Mich.	MI	Lansing	
23	Minnesota	Minn.	MN	St. Paul	
24	Mississippi	Miss.	MS	Jackson	
25	Missouri	Mo.	MO	Jefferson City	

Ein konkretes Beispiel wäre:

User 1 twittert mit den Ortsangaben Dover, Delaware.

User 2 twittert mit den Ortsangaben USA.

User 3 twittert mit den Ortsangaben Topeka.

Bei allen drei Ergebnissen brauch ich das Land USA, demnach gleiche ich die Listen mit folgendem Code ab und ändere die Ergebnisse auf das passende Land. Dieser Abschnitt ist nicht nur für die USA, sondern für alle relevanten Länder. Nach und nach werden alle Länder mit Hilfe der for Schleife abgearbeitet.

```
37 self.values = []
38 self.label = []
39 self.dicts = {}
40
41 self.index = 0
42 for country in self.countries: #passes through state lists and change entries to proper country
43     for state in country:
44         for elem in state:
45             self.keys = []
46             self.tmp = elem.rstrip()
47             for entry in data:
48                 if data[entry].find(self.tmp) != -1:
49                     self.country_list.append(self.names[self.index])
50                     self.keys.append(entry)
51
52             for key in self.keys: #if entry was found its deleted to not check it again for next iteration
53                 if key in data:
54                     data.pop(key, None)
```


Für Fantasienamen aus WoW wird zuerst eine Liste separat erstellt und dieser Code verwendet:

```
57
58     for wow in self.azeroth:      #passes through list and change to world of warcraft
59         self.keys = []
60         for entry in data:
61             if data[entry].find(wow) != -1:
62                 self.country_list.append('World of Warcraft')
63                 self.keys.append(entry)
64
65         for key in self.keys:
66             if key in data:
67                 data.pop(key, None)
```

Für alle anderen Länder die zusätzlich eine höhere frequency aufweisen benutze ich zusätzlich diesen Abschnitt. Hier werden diese zutreffenden Länder extra überprüft.

```
68
69     for left in self.leftover:     #passes through list and change to proper countries
70         self.keys = []
71         for entry in data:
72             if data[entry].find(left) != -1:
73                 self.country_list.append(left)
74                 self.keys.append(entry)
75             elif data[entry] == 'Россия':
76                 self.country_list.append('Russia')
77                 self.keys.append(entry)
78             elif data[entry] == 'UK':
79                 self.country_list.append('UK')
80                 self.keys.append(entry)
81             elif data[entry] == 'United Kingdom':
82                 self.country_list.append('UK')
83                 self.keys.append(entry)
84             elif data[entry].find('Wales') != -1:
85                 self.country_list.append('UK')
86                 self.keys.append(entry)
87
```

Ein wichtiger Bestandteil in dieser Datei ist diese Funktion:

```
51
52
53     for key in self.keys:          #if entry was found its deleted to not check it again for next iteration
54         if key in data:
55             data.pop(key, None)
```

Sobald ein Eintrag erkannt und geschrieben wurde, wird dieser temporär entfernt, damit er in keiner Iteration erneut überprüft und gewählt wird.

Die geo.py und die DataReader.py

Um die Analyse nun noch zu konkretisieren, möchte ich nun nicht nur die Ergebnisse grafisch in einer Abbildung haben, die mir die Häufigkeit aller Länder anzeigt, sondern ich hätte gerne die Geo-Tags separat auf der Weltkarte. Ich möchte gerne wissen, von wo und in welcher Sprache getwittert wurde. Dafür bietet sich Matplotlib zusammen mit Basemap am besten an. Hierbei ist es möglich die tweets nach Geo-Tags zu durchsuchen und anhand ihrer Koordinaten als Markierung auf der Basemap hinzuzufügen. Da Twitter allerdings nur mit bounding_box_coordinates arbeitet (diese Beschreiben ein Feld auf der Weltkarte mit 4 Koordinaten), muss ich hierfür den Schnitt der 4 Koordinaten benutzen und komme dadurch genau in das Zentrum des Geo-Tags. Dies erledigt die DataReader.py wie folgt:

```

49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72

if ('NaN' not in row[4]): #checks if coords are given or not

    self.temp = row[4].replace('[[', '')
    self.temp = self.temp.replace(']]', '')
    self.temp = self.temp.replace('[', '')
    self.temp = self.temp.replace(']', '')
    self.temp = self.temp.replace(' ', '')

    self.temp_coords = self.temp.split(',')

    self.temp_lat = (float(self.temp_coords[0])+float(self.temp_coords[4])) /2
    self.temp_lat = round(self.temp_lat, 6)
    self.temp_long = (float(self.temp_coords[1])+float(self.temp_coords[5])) /2
    self.temp_long = round(self.temp_long, 6)

    self.coords.append(self.temp_lat)
    self.coords.append(self.temp_long)

    self.coordinates[row[0]] = self.coords

if ('NaN' not in row[5]):
    self.country[row[0]] = row[5]

```

Die datareader.py übernimmt in diesem Fall die Überprüfung von mitgelieferten Koordinaten. Sind Koordinaten bei einem Tweet vorhanden, wird mit Hilfe der Formel der Schnittpunkt ermittelt und via Basemap ein Schnittpunkt in Form eines Punktes auf der Karte markiert. Damit steht nun fest von wo der tweet kommt. Jetzt möchte ich aber noch wissen, in welcher Sprache der Tweet getwittert wurde. Grund hierfür sind Länder wie Schweiz oder Kanada, in denen nicht nur deutsch oder englisch, sondern auf französisch gesprochen wird. Des Weiteren können Menschen natürlich auch aus dem Urlaub oder von Geschäftsreisen twittern. Vorher habe ich mich für die Sprachen Englisch, Deutsch, Französisch und Spanisch entschieden. Diese Sprachen sollen mit Hilfe der geo.py auf der Karte mit verschiedenen Farben markiert werden. Hierbei werden sämtliche Koordinaten überprüft und zusätzlich mit den gelieferten Sprachinformationen abgeglichen. Dafür sorgt der folgende Abschnitt:

```

21
22
23
24
25
26
27
28
29
30
31
32
33
34

for elem in coordinates:
    self.x, self.y = map(coordinates[elem][0], coordinates[elem][1])
    if language[elem] == 'en':
        map.plot(self.x, self.y, 'ro', markersize=3)
    elif language[elem] == 'de':
        map.plot(self.x, self.y, 'go', markersize=3)
    elif language[elem] == 'fr':
        map.plot(self.x, self.y, 'bo', markersize=3)
    elif language[elem] == 'es':
        map.plot(self.x, self.y, 'yo', markersize=3)
    elif language[elem] == 'und':
        map.plot(self.x, self.y, 'wo', markersize=3)
    plt.draw()

```

Hier wird dafür gesorgt, dass die Koordinaten mit den Ergebnissen der Spracherkennung abgeglichen werden. Gibt es für beide einen Treffer, wird der Punkt farbig je nach Sprache markiert.

3. Ergebnisse

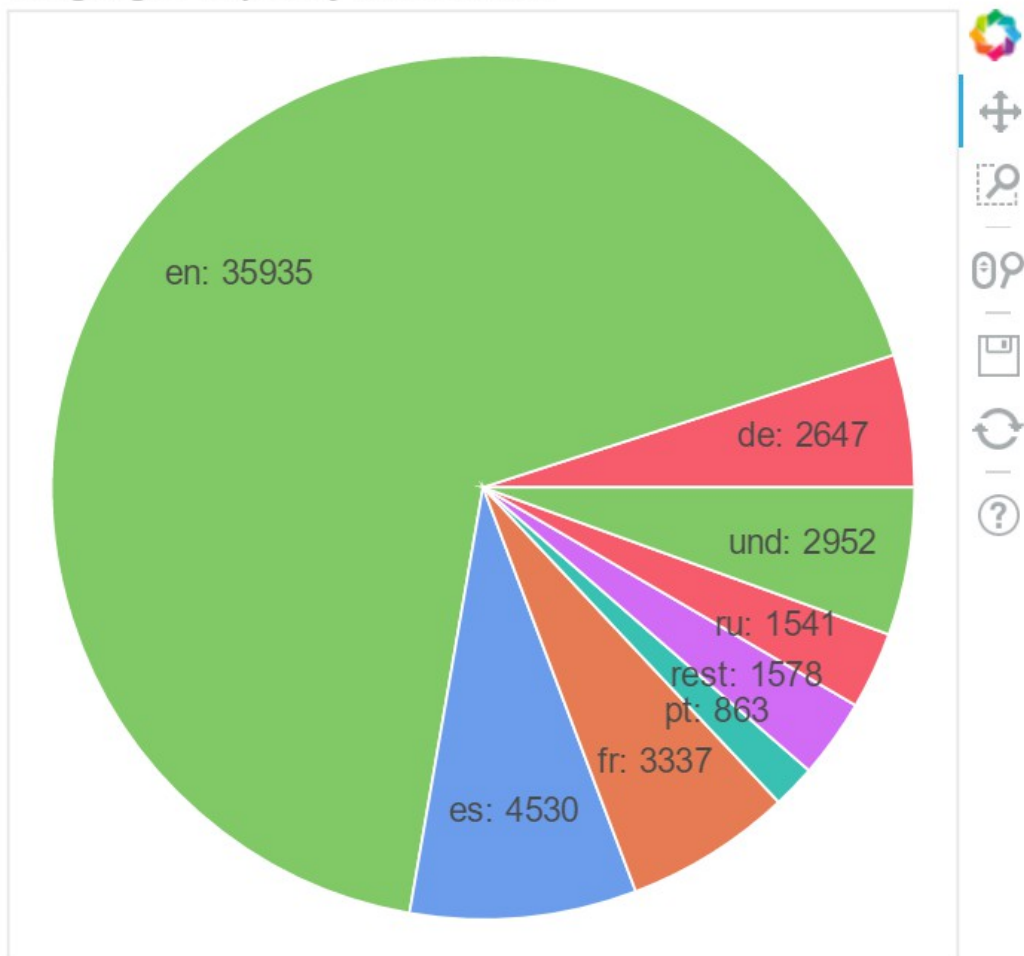
Insgesamt kam ich auf eine stolze Summe von 53.383 tweets, voller Datensalat.

```
53350 770775417040932864;2016-08-31 00:09:14;Done and done! Thanks @XXRacer for making a freaking cool chair! Time to #Warcraft https://t.co/ISquFBIMRb;en;NaN;38° 37' 38 N / 90° 11' 52 W
53359 770775429464657920;2016-08-31 00:09:17;DK 109! #Warcraft! https://t.co/K5v8dkg9I;und;NaN;Draenor, US
53361 770775444803104768;2016-08-31 00:09:21;I just earned the [Eye of Azshara] Achievement! #Warcraft https://t.co/LopSssRwmn;en;NaN;NaN
53362 770775458484850688;2016-08-31 00:09:24;Check out my screenshot from World of #Warcraft! @thorh13 https://t.co/T7cP8igyKQ;en;NaN;NaN
53363 770775478081870017;2016-08-31 00:09:29;@lt;3 World of #Warcraft! https://t.co/XV9YTOGdpJ;en;NaN;Buffalo, NY
53364 770775487140524032;2016-08-31 00:09:31;MiRa mi captura de pantalla de World of #Warcraft! https://t.co/8YZV00CB3i;und;NaN;Alicante
53365 770775504114909188;2016-08-31 00:09:35;RT Warcraft: .BlizzHeroes Thank you. They were delectable. https://t.co/E4dYHnHq1 https://t.co/6qZ7ZvxA6 #Warcraft #blizzard;en;NaN;Cincinnati, OH
53366 770775515099762688;2016-08-31 00:09:37;Mage 109! #Warcraft! https://t.co/ulkOLJ4Q9i;en;NaN;Draenor, US
53367 770775510869266288;2016-08-31 00:09:38;10 Months of testing and they release a bugged quest. #Legion https://t.co/DwdfnGY24B;en;NaN;NaN
53368 770775562545668096;2016-08-31 00:09:40;#worldofwarcraft;und;NaN;Porto Alegre - RS
53369 770775573606068225;2016-08-31 00:09:51;There's always time for a fishing break! #Warcraft! https://t.co/YnUHoESMqH;en;NaN;Ruins of Lordaeron
53370 770775575258599424;2016-08-31 00:09:52;Regardez l'objet que je viens de recevoir : [Chouette pygmée] https://t.co/fGaweb5S1 #Warcraft;fr;NaN;Bordeaux, Aquitaine
53371 770775581642413568;2016-08-31 00:09:53;Check out my screenshot from World of #Warcraft! https://t.co/lur1GzzYnB;en;NaN;Planet Skydolf
53372 770775604295643136;2016-08-31 00:09:59;This game is amazing. standing at 49.7 86.6 stormheim. #wow #Warcraft! https://t.co/g1X8RH5yK;en;NaN;Azeroth
53373 770775607110172672;2016-08-31 00:09:59;yes #Warcraft! https://t.co/jjgiYgs13b;und;NaN;West Chicago, IL
53374 770775612843708416;2016-08-31 00:10:01;@Warcraft @WarcraftDevs #worldofwarcraft #Legion https://t.co/mxggytccn;und;NaN;USA
53375 77077563897690625;2016-08-31 00:10:07;Check out my screenshot from World of #Warcraft! https://t.co/Mlw79uzlrm;en;NaN;Londres, Inglaterra
53376 77077564632690608;2016-08-31 00:10:09;Je viens d'accomplir le haut fait [Azuna matata] : #Warcraft https://t.co/kd6M12sQhB;fr;NaN;Bordeaux, Aquitaine
53377 770775658247163905;2016-08-31 00:10:11;Olha só o item que acabei de ganhar! [Crematória] https://t.co/YIW9cf07y #Warcraft;pt;NaN;Azralon
53378 770775658804940801;2016-08-31 00:10:12;I have joined the fight against the #Legion! WHO WILL JOIN ME? https://t.co/vJ0Hgys021;en;NaN;New York
53379 770775668460249088;2016-08-31 00:10:14;Aw yeah. Affliction locking! #Warcraft https://t.co/T1bmBthciE;en;NaN;Rochester, NY
53380 770775715969040576;2016-08-31 00:10:25;Druid done #Artifact #Legion #Warcraft https://t.co/p0vU9QvPLd;en;NaN;Gilneas, Elune
53381 770775735036440577;2016-08-31 00:10:30;¡Acabo de conseguir el logro [Nivel 110]! #Warcraft POR.FIN https://t.co/v8yQWCEcn;es;NaN;En una montaña por Aeffkspene.
53382 77077573626015616;2016-08-31 00:10:30;That's my shaman at 110, yay #Legion;en;NaN;Antwerpen, België
53383 770775759103283200;2016-08-31 00:10:35;I have joined the fight against the #Legion! WHO WILL JOIN ME? https://t.co/lFu6Lg8r7f;en;NaN;日本 東京都
```

Dieser musste nun gefiltert und die Tweets unterschieden werden.

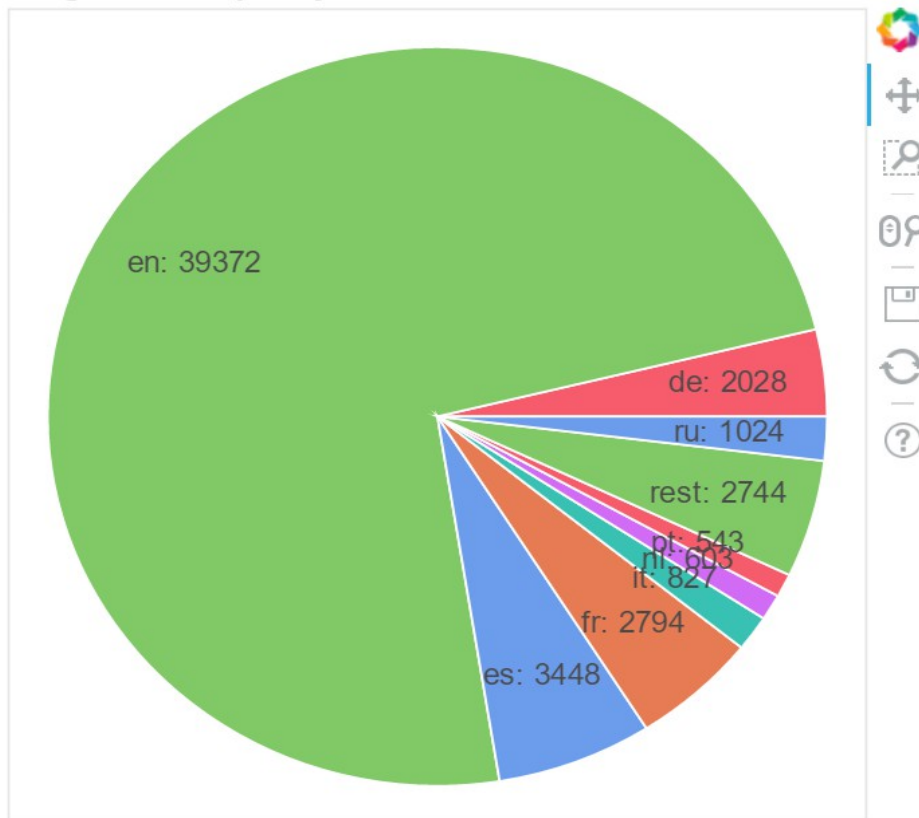
Zum einen hatte ich das Ergebnis der Twitter-Sprachanalyse, die „Abbildung der Language Frequency Distribution“

Language Frequency Distribution



und zum anderen die Analyse über Langdetect von Python, die „Abbildung der Langdetect Frequency Distribution“

Langdetect Frequency Distribution



Beides wurde mit Hilfe von Bokeh im Kuchen- bzw. „Donut“- Diagramm dargestellt.

Anfangs wollte ich für diese Darstellung Networkx benutzen, allerdings ist mir bei genauer Betrachtung der Masse der Ergebnisse schwindelig geworden. Wenn man es nun mit Hilfe von Networkx darstellen möchte, sieht man nur noch einen bunten Knubbel an Farben und keinerlei Hinweise auf das Gesamte wie bei einem Kuchendiagramm.

Bokeh hat nicht nur den Vorteil, dass ich die Ergebnisse interaktiv via HTML ausgeben lassen kann, sondern es ist deutlich übersichtlicher. Dies bedeutet, dass ich auch noch im HTML Format zoomen, speichern und verschieben kann. Um nun zu schauen, welches der Beiden Ergebnisse nun präziser ist, müsste ich alle tweets per Hand analysieren und überprüfen, was aber bei 53.383 tweets den Zeitrahmen dieses Seminars deutlich gesprengt hätte. Allerdings ist mir bei den Ergebnissen beider Vorgänge nur ein Unterschied der Zahlen aufgefallen. Die Tendenzen sind nicht extrem verzerrt. Bei beiden Analysen ist deutlich zu erkennen, welche Sprachen am dominantesten sind.

1. Englisch (35935 / 39372)
2. Spanisch (4530 / 3448)
3. Französisch (3337 / 2794)
4. Deutsch (2647 / 2028)
5. Russisch (1541 / 1024)
6. Portugisisch (863 / 543)
7. rest/undefined (2952 / 2744)

Besonders interessant sind hier die Sprachen „Niederländisch“, sowie „Italienisch“. Bei Twitter wird Niederländisch anscheinend der deutschen Sprache zugeordnet und Italienisch der spanischen Sprache. Italienisch ähnelt der spanischen Sprache sehr, ebenso wie Niederländisch der deutschen und wird von Twitter anscheinend nicht korrekt unterschieden. Twitter ist mit der Spracherkennung sehr schnell, wobei langdetect bei meinen Ergebnissen knappe 15 Minuten für die Analyse und Ausgabe gebraucht hat, dafür aber präziser im Linguistik-Algorithmus ist. Auffällig hierbei ist aber auch die große Menge an nicht identifizierten Sprachen. Dies liegt hauptsächlich daran, dass sehr viele tweets nur Screenshots und Bilder, sowie links oder hashtags beinhalten, aber keinerlei verwertbare Sprache. Außerdem fallen viele Sprachen aufgrund der Kodierung durchs Raster, besonders asiatische sowie arabische Sprachen wie koreanisch, chinesisches, russisch (kyrillisch). Eine Idee zu Beginn war es die Wörterbücher von langdetect auf den „gamespezifischen slang“ anzupassen. Diese Idee verwarf ich relativ schnell wieder, da dies nicht so eindeutig war wie ich zuerst dachte. Wie auch im Seminar besprochen denkt man zuerst bei so was wie „dudu op“ an einen deutschen tweet. Aber weit gefehlt...

Als Beispiel „Dudu“:

```
19075 770354220688150529;2016-08-29 20:15:33;Queda poco para #Legion #TodosContraLaLegion mi dudu está listo #ForTheHorde @Warcraft ES;es;NaN;NaN
21621 770380477551218690;2016-08-29 21:59:53;Dat Dudu Stuff :Df #Warcraft! https://t.co/fzb7NUJBNI;en;NaN;Gallifrey
```

Dudu kommt nämlich auch mal gerne in der spanischen oder englischen Sprache vor. Ebenso wie das Beispiel „Drood“:

```
47947 770719486148218880;2016-08-30 20:26:59;Je n'aime pas l'arme prodigieuse des #drood Équi : j'ai l'impression d'être la Mort - - #WoW #Legion #WorldOfWarcraft https://t.co/mvucWCaGs;fr;NaN;Biarritz
18861 770350575477264385;2016-08-29 20:01:04;"Minuit approche &amp; je ne suis tjrs pas décidé : j'up en 1er ma Mage HZ récolteuse ou ma Drood Ally Alchi - - #WoW #Legion #worldofwarcraft";fr;NaN;Biarritz
```

Ebenfalls kommt „Drood“ nicht nur in der englischen, sondern auch in der französischen Sprache vor. Da ich bei solchen Treffern gegen einen Sprachenkonflikt anlaufen würde, wäre mir auch mit einer Erweiterung der Wörterbücher für diese Sonderfälle nicht geholfen und da ich ein „dudu op“ in keinsten Weise klar definieren kann, fällt es unter „undefined“. Aus diesen Gründen habe ich mich dann gegen die Erweiterung der Wörterbücher entschieden. Bei meinen Recherchen bin ich auf eine Arbeit von Shyo Nakatani gestoßen, der sich speziell auf die „Short Text Language Detection with Infinity-Gram“ konzentriert hat. Nach kurzer Zeit musste ich feststellen, dass seine Arbeit nicht nur meinen linguistischen Horizont, sondern auch die Zeit für dieses Projekt deutlich „überspringt“.

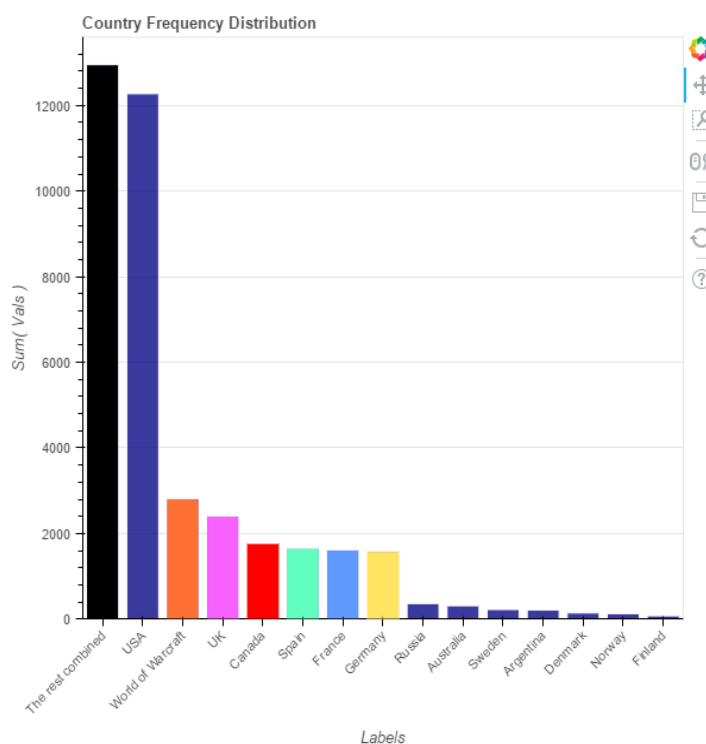
Auch die erste Länderanalyse habe ich mit Bokeh darstellen lassen, dieses mal allerdings im „Bar-Plot“ Diagramm. Um an die Ergebnisse zu kommen, musste ich (wie bereits in den Methoden erwähnt) eine Menge an Vorarbeit leisten. Ich habe mir bereits vorhandene Listen der größten Städte aller bedeutender Länder besorgt, diese formatiert und miteinander verschmolzen.

Dieser Ausschnitt der Liste USA zeigt, wie ich vorgegangen bin. Bei der USA habe ich alle Abkürzungen des Staates dem Staat angehängt und gleichzeitig noch die Metropolen hinzugefügt. USA war besonders schwierig zu handhaben, da verschieden abgekürzt wird.

Zum einen gibt es Benennungen wie New York, N.Y., New York, NY. oder auch New York, New York. Wie bereits in den Methoden vorweggenommen wird hier eine spezielle Liste mit postalcodes sowie Hauptstädten der verschiedenen Staaten genommen, da quasi jeder US-Staat wie ein Land fungiert.

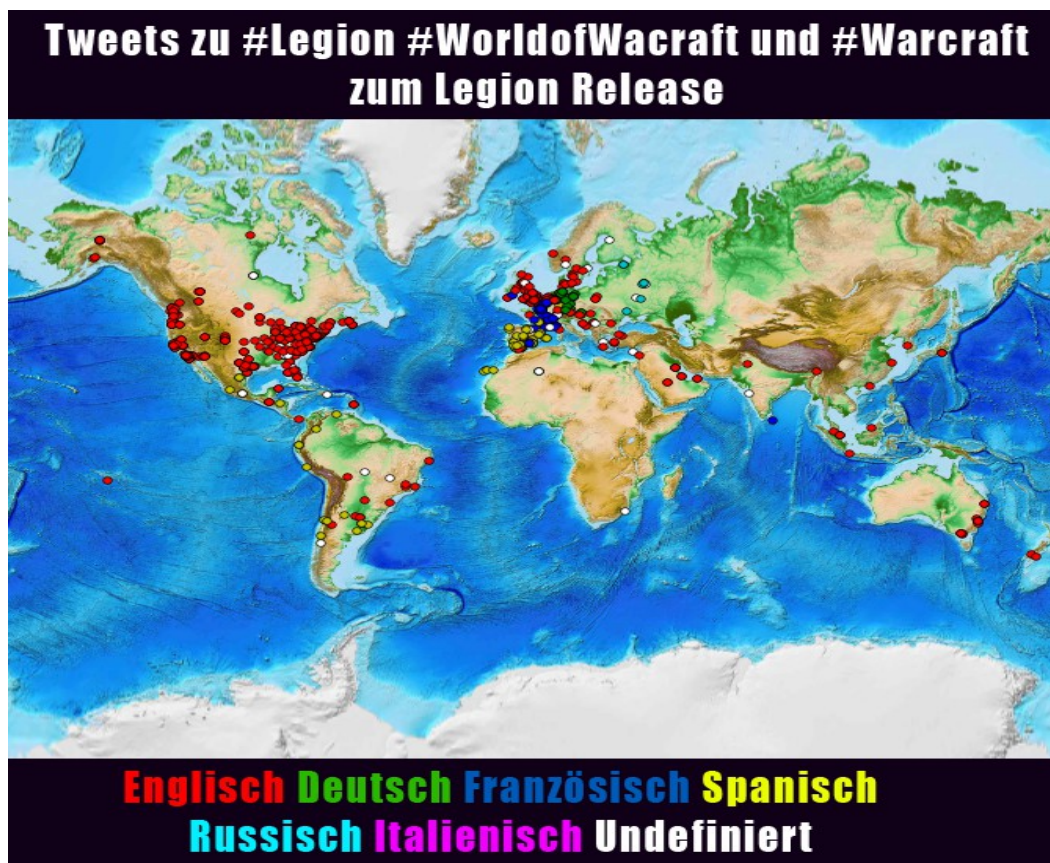
	A	B	C	D	E
1	Alabama	Ala	AL	Montgomery	
2	Alaska	Alaska	AK	Juneau	
3	Arizona	Ariz	AZ	Phoenix	
4	Arkansas	Ark	AR	Little Rock	
5	California	Calif	CA	Sacramento	
6	Colorado	Colo	CO	Denver	
7	Connecticut	Conn	CT	Hartford	
8	Delaware	Del	DE	Dover	
9	Florida	Fla	FL	Tallahassee	
10	Georgia	Ga	GA	Atlanta	
11	Hawaii	Hawaii	HI	Honolulu	
12	Idaho	Idaho	ID	Boise	
13	Illinois	Ill	IL	Springfield	
14	Indiana	Ind	IN	Indianapolis	
15	Iowa	Iowa	IA	Des Moines	
16	Kansas	Kans	KS	Topeka	
17	Kentucky	Ky	KY	Frankfort	
18	Louisiana	La	LA	Baton Rouge	
19	Maine	Maine	ME	Augusta	
20	Maryland	Md	MD	Annapolis	
21	Massachusetts	Mass	MA	Boston	
22	Michigan	Mich	MI	Lansing	
23	Minnesota	Minn	MN	St. Paul	
24	Mississippi	Miss	MS	Jackson	
25	Missouri	Mo	MO	Jefferson City	

Nicht wirklich verwunderlich ist auch das Ergebnis an sich. Es fällt deutlich auf, dass die meisten Tweets aus „The rest combined“ bestehen. Das sind vor allem gar keine Einträge, fehlerhafte Einträge und oder Einträge die von den Listen nicht abgedeckt werden. Ich habe jetzt keine Listen gefunden, die jeden kleinen Kurort namens Meerbusch oder Neuss oder Dorfunter beinhalten. Diese Orte fallen dann natürlich unter „Rest“. Ebenfalls nicht so überraschend ist es, dass die USA weit vorne liegt was tweets angeht. Twitter hat in der USA einen deutlich höheren Stellenwert als in anderen Ländern.



Was allerdings negativ auffällt ist das Problem der kyrillischen Ortschaften. Sehr viele Russische Städte und Dörfer sind anscheinend durchs Raster gefallen, weil das kyrillisch in den Listen nicht mit den Angaben in Twitter übereinstimmt. Viele User geben aber auch erst gar keine Informationen über ihren Ort bekannt, da es bei Twitter kein „Muss“ ist, einen Wohnort anzugeben, bzw. man ihn verstecken kann. Ebenso geben sehr viele Leute Fantasieorte an. Passend zu WOW natürlich, Azeroth, Kalimdor, Stormwind, Orgrimmar usw.. Durch meine eigene Warcraft-Erfahrung konnte ich dazu allerdings eine Recht gute Liste entwerfen, um den groß-teil dieser Ortschaften als „World of Warcraft – Orte“ zu identifizieren.

Der letzte Teil der Analyse und Darstellung ist das Geo-Mapping. Dieser Teil war mir besonders wichtig, weil man hier sehr schön sehen kann, von wo und vor allem in welcher Sprache getwittert wurde. Entschieden habe ich mich am Ende doch für die Kombination Matplotlib und Basemap, weil dieses neben GEOJson einfach die gängigste Methode ist. Plotly fiel bei mir ziemlich schnell raus, da dies einfach nur füttern von Informationen beinhaltet, aber keinerlei programmiertechnische Eigenleistung. Des Weiteren wird Plotly ab einem bestimmten Punkt auch kostenpflichtig.

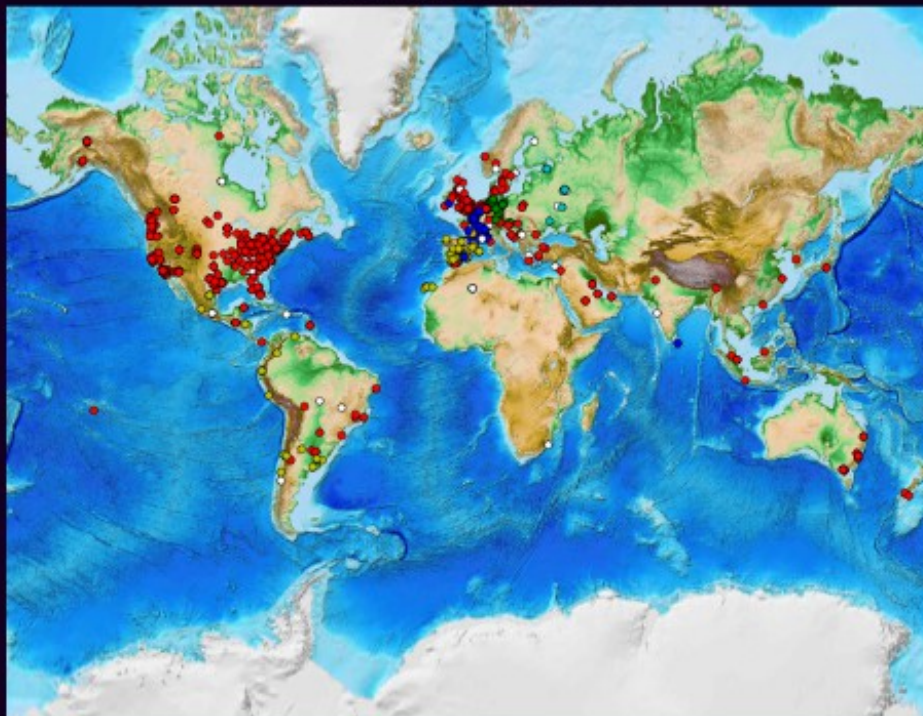


Auch hier ist es nicht verwunderlich, dass die meisten Geo-Tags aus dem Raum Nordamerika kommen und englischen Tweets angehören. Auch hier finden sich wieder undefinierte Sprachen, größtenteils arabischer, kyrillischer oder HTML/Media Herkunft sind. Überraschen ist auch der rote Punkt im pazifischen Ozean. Dabei handelt es sich entweder um eine sehr kleine Insel, eine fehlerhafte Koordinate oder eine Person auf einer längeren Schiffsfahrt, die es kaum erwarten kann Legion zu spielen.



Der Release von Legion stand vor der Tür und im Rahmen eines Uni-Projekts haben Neciferia und ich den Hype via Twitter ein wenig unter die Lupe genommen. Wir haben während der letzten Woche vor dem Release inkl. dem Legionstart sämtlich tweets zu den hashtags #Legion #Worldofwarcraft und #Warcraft aufgefangen + analysiert und möchten euch diese kurz und knapp präsentieren

Tweets zu #Legion #WorldofWacraft und #Warcraft zum Legion Release



Englisch Deutsch Französisch Spanisch
Russisch Italienisch undefiniert

Die Weltkarte zeigt sämtlichen Tweets die mit Hilfe von Twitter Geo-Tags identifiziert werden konnten. Die verschiedenen farblichen Punkte entsprechen den unterschiedlichen Sprachen der verschiedenen Länder, aus denen getwittert wurde. Nicht verwunderlich ist hierbei die Häufigkeit in den USA sowie der englischen Sprache

(Screenshot eines Teils der Präsentation via Homepage mit PHP)

Als „on Top“ Ziel haben wir im Projektplan die Einbindung und Darstellung der Ergebnisse via PHP/HTML auf unserer Community-Seite www.Cede-Maiori.info abgesprochen. In Zusammenarbeit mit Raphael Katschke wurde dies auch umgesetzt. Da wir parallel am selben Thema, allerdings an zwei eigenständigen Projekten, gearbeitet haben, durften wir gemeinsam den Twitter-Crawler, sowie die Homepage benutzen um unsere beiden Ergebnisse am Ende bestmöglich zusammen zu präsentieren. Wir haben uns darauf geeinigt, dass wir die Ergebnisse in „News-Form“ auf der Frontpage einbinden. Grund hierfür ist einfach, dass wir auch den Mitgliedern von „Cede Maiori“, sowie sämtlichen Besuchern Einblick gewähren wollten. Um dies zu realisieren, wurden die Ergebnisse homepagefreundlich angepasst, vereinheitlicht und statisch als Bilder auf den Server kopiert und in Kombination mit Texten zur kurzen Erläuterung via PHP-Code eingebunden. Separat haben wir noch die Ausgaben von Bokeh (welche ja interaktiv als HTML Format ausgegeben wurden) hinzugefügt. Als Beispiel ein kleiner Screenshot der Homepage mit einem Auszug der Ergebnisse.

3. Benötigte Programme, Libs und Anwendung

Was benötigt ihr um das Projekt laufen zu lassen?

Als erstes wird eine Verbindung zur Twitter-API benötigt um den Crawler zu bedienen, dafür bedarf es einem Twitter-Account sowie bereits in Methoden beschrieben, generierte Consumer Keys und Tokens, die dem Twitter-Account hinzugefügt werden. Allerdings kann auch die von mir beigelegte „results.csv“ benutzt werden, um die Main.py zu starten und die Ergebnisse auszugeben.

Welche Libs sind nötig?

Damit der Code einwandfrei läuft sind folgende **imports** absolut notwendig.

- `tweepy`
- `re`
- `collections`
- `unicodcsv`
- `nltk`
- `bokeh`
- `matplotlib`
- `mpl_toolkits.basemap`
- `os`
- `langdetect`

Geschrieben wurde das Projekt mit Python 2.7, daher ist es notwendig, dies auch mit 2.7 laufen zu lassen. 3.x kann zu Problemen führen.

Der komplette Ablauf lautet wie folgt:

1. Twitter-Keys und Tokens in `twitter_keys.py` eintragen (nur wenn man selbst den Crawler laufen lassen möchte)
2. Die `Main.py` starten und laufen lassen (benötigt die von mir mitgelieferte `Results.csv`)
 1. Der Vorgang kann je nach Rechenleistung länger dauern.
3. Als erstes wird die Geo-Map via Matplotlib und Basemap ausgegeben. Erscheint diese, muss sie geschlossen werden. Es wird ein Ergebnis in Form eines Bildes und namens `Geomap.png` erstellt. Andernfalls kann sich der Ablauf stark verzögern.
4. Nun werden die Bokeh Diagramme gezeichnet und als interaktive HTML im Browser geöffnet.
 1. Insgesamt werden 3 Diagramme gezeichnet. 2 mal ein Kuchendiagramm und 1 mal ein Bar-Diagramm.
 2. Wie bereits erwähnt, kann dies mitunter bis zu 15 Minuten oder länger dauern, da `langdetect` alle 53.383 Tweets durchgeht und analysiert.
5. Wichtig! Es befinden sich bereits Ergebnisse im Github, damit ihr es nicht extra durchlaufen lassen müsst. Wollt ihr das trotzdem, solltet ihr vorher die 3 HTML Dateien und die `geomap.png` entfernen, oder ihr überprüft danach das Änderungsdatum der Datei, denn diese werden überschrieben, wenn sie bereits vorhanden waren.

4. Fazit, Probleme und Lösungen

Grundlegend hat mir das Projekt viel Spaß gemacht, da mich das Thema natürlich sehr interessiert und ich den Hype live mitbekommen und verfolgt habe. Leider habe ich durch nicht ganz perfekter Vorarbeit ein wenig improvisieren müssen, bin aber der Meinung, dass ich gute, wenn nicht sogar bessere Libs für die Darstellung der Ergebnisse gewählt habe. Was mich ein wenig aus der Bahn geworfen hatte, war vor allem `Networkx`. `Networkx` sieht interessant aus, leider hat es bei der Installation arge Probleme bereitet und hat nicht den Zweck erfüllt, den ich eigentlich damit erreichen wollte. Daher war dies eine besonders schlechte Wahl meiner Meinung nach. `Networkx` liefert grafisch zwar Highlights was Layouts angeht (besonders `Rheingold`), allerdings haben diese meiner Meinung nach (vor allem bei einer Masse von 53.383 Tweets) keine bereitstellbaren Informationen der Statistiken die ich benötigt habe. Das Besprechen des Projektplans mit euch, hat allerdings dazu beigetragen, dass ich nicht das komplette Projekt kippen musste, sondern einfach nur eine andere Bibliothek zur Darstellung der Ergebnisse wählen musste. Python macht mir dies sehr einfach, da Python in dieser Hinsicht recht viel zur Auswahl hat. Ich bin dann auf `Bokeh` gestoßen, eine Lib zur graphischen Darstellung die ich bis dato nicht kannte, mich aber sehr angesprochen hat. `Bokeh` gibt den von mir erstellten Code als Diagramm in einer HTML Datei aus. Diese Datei ist interaktiv, also kann damit ein User rein zoomen, sie verschieben oder sogar als Bild auf seinem Rechner speichern. Leider fehlte die Zeit, mich weiter und intensiver mit `Bokeh` zu beschäftigen,

aber es hat genau das ausgegeben was ich für wichtig hielt, nämlich der Vergleich beider Sprachidentifizierungsmethoden, sowie ein gut designer Barplot für die Darstellung der Länderinformationen der Twitteruser. Ein wenig enttäuscht bin ich von Plot.ly gewesen. Die Bubble-Map für Verwendung der Geo-Tags sah gut aus, bei näherer Betrachtung aber fiel auf, dass Plot.ly extrem automatisiert ist und quasi nur eine copy&paste deinen Code Anwendung darstellt. Sinn und Zweck dieses Projektes war ja, dass wir die Ergebnisse mit Hilfe von Programmiersprachen erzielen, daher habe ich mich gegen Plot.ly und für Matplotlib in Kombination mit Basemap entschieden, da genau das, was ich vorher geplant hatte damit umsetzbar war. Des Weiteren wird Plot.py ab einem bestimmten Punkt „kostenpflichtig“ und benötigt die PRO Version. Das war ebenfalls ein absolutes „No go“ für mich. Erstaunt war ich allerdings, dass es kaum Probleme bei der Spracherkennung gab. Twitter sowie langdetect waren sich größtenteils einig, Twitter büßte allerdings an Präzision ein, ist dafür aber eindeutig schneller.

Probleme gab es vor allem Seitens Twitter. Twitter hat die Geo-Tags defaultmäßig deaktiviert, deswegen ist die Anzahl der Geo-Tags sehr gering, was leider nicht zu ändern ist. Daher habe ich mich dazu entschlossen, neben der Geo-Map noch die separaten Listen für alle interessanten Länder zu kreieren, um eine Übersicht aller Herkunftsorte ohne Geo-Tags zu erlangen. Von der Zeit her wurde es ziemlich knapp. Dies lag hauptsächlich daran, dass ich zwar schon mit Python und Twitter im Studium gearbeitet hatte, aber nie in dieser Art und Weise. Auch die Menge bzw. Größe des Projekts habe ich ein wenig unterschätzt und kam zum Ende hin deutlich ins Schwitzen, gerade weil es immer wieder bugs und Kodierungsfehler gab. Das Ausweichen auf andere Bibliotheken hat mir dabei natürlich auch nicht unbedingt in die Karten gespielt. Aus dem Projekt nehmen ich allerdings wichtige Punkte mit. Unter anderem sollte man sich gerade beim Programmieren nicht unbedingt auf bestimmte Bibliotheken verlassen und immer einen Notfallplan bereit haben, denn sonst steht einem das Wasser ganz schnell bis zum Hals, vor allem wenn man nicht so geübt ist wie wir Studenten. Abschließen würde ich sagen, dass mein Plan im großen und ganzen gut aufgegangen ist und hoffe das meine Ausgangsfrage zufriedenstellend für euch beantwortet wurde.