Raphael Katschke

1994872

BrainWasheD

Einleitung

Das Thema der Projektidee befasst sich mit der Datenanalyse von Tweets in Hinblick auf Häufigkeiten, Schlagwort-, Erwähnungs- und Hashtagverteilungen zu dem Release der World of Warcraft Erweiterung: Legion von Blizzard Entertainment.

World of Warcraft (WoW) kam erstmals am 23. November 2004 in Neuseeland, Australien, Mexiko, Kanada und den USA auf den Markt und erschien europaweit am 11. Februar 2005. Das Spiel zählt zu den erfolgreichsten Computerspielen der Welt und erreichte Ende 2010 ein Hoch von 12 Millionen Abonnenten, welche jedoch bis 2015 auf 5.5 Millionen gesunken sind. Am 8. September 2016 wurde berichtet, dass die Anzahl der gleichzeitig aktiven Spieler in der Veröffentlichungswoche von Legion so hoch gewesen ist wie seit 2010 nicht mehr.

Das Spiel baut in seiner Geschichte auf der RTS (Real-time strategy) Spielereihe Warcraft auf wovon das prominenteste Spiel Warcraft III (+The Frozen Throne Erweiterung) ist und erweiterte sie in bis dato 22 Romanen und einem Kinofilm. Es ist öfter Teil von Diskussionen rund um Computerspielesucht und wurde in einer Folge

der Sendung Southpark parodiert. Das Entwicklerstudio Blizzard Entertainment ist regelmäßig auf Computerspielemessen vertreten und organisiert selbst eine eigene Messe zu allen Blizzard Entertainment Spielen (BlizzCon). Die Ankündigung der 6. Erweiterung Legion erfolgte 2015 auf der Gamescom und wurde Live auf Twitch von ~250.000 Menschen verfolgt.

Die großen thematischen Hauptpunkte des Spieles sind der stetige Krieg zwischen den Fraktionen Horde und Allianz sowie den Lichtwesen Naaru und der Burning Legion. Diese – geformt um den verdorbenen Titanen Sargeras – hat sich zum Ziel gesetzt alles Leben im Universum zu vernichten um eine Manifestation einer noch böseren Macht zu verhindern. Unzählige Welten sind der schier endlosen Macht der dämonischen Armee zugrunde gefallen und die Welt Azeroth spielt eine zentrale Rolle im Widerstand. Mehr als 10.000 Jahre nach der ersten gescheiterten Invasion versucht die Legion erneut Azeroth zu vernichten und bündelt hierzu ein riesiges Portal auf den Broken Shores, wo der besiegte Avatar von Sargeras begraben liegt. Die Erweiterung beginnt mit dem verzweifelten Versuch von Horde und Allianz in einem vereinten Angriff die Dämonen zurückzuschlagen, was jedoch kläglich scheitert.

Thematisch befasst sich Legion mit den bis dato abgeschotteten Nightelves aus der antiken Stadt Suramar, der erneuten Invasion der „Burning Legion“ und einem der beliebtesten Charaktere von Warcraft: Illidan Stormrage. Ein Charakter der in einem ständigen inneren Konflikt zwischen Machtgier, Rache und dem verzweifelten Versuch die Legion zu vernichten steckt. Von seinem Volk als Verräter gebranntmarkt und seinem Bruder verstoßen wurde er in der ersten Erweiterung von den Abenteurern niedergestreckt, sein Körper und seine Seele eingesperrt. Die Erweiterung Legion soll Illidan auf eine Erlösungsgeschichte leiten.

WoW gilt als König der MMORPGs und es ist bislang keinem Spiel gelungen, WoW zu „entthronen“. Es ist eine der letzten Bastionen der Abonnementmodelle im MMORPG Genre zusammen mit Final Fantasy 14: A Realm Reborn (5 Millionen registrierte Accounts). Der mit dem Spiel verbundene Hype(train) soll anhand von Häufigkeitsdistributionen und Wortzusammenhängen genauer analysiert werden. Im Fokus stehen hier die meist genutzten Begriffe und die mit denen zusammen vorkommenden Worte, sowie eine Stimmungsanalyse aller Tweets in Form von Positiv, Neutral und Negativ.

Projektplan

Der Plan ist in fünf Schritte nach Komplexität aufgeteilt.

Schritt 1 sieht vor, dass der Twittercrawler geschrieben und angepasst wird. Da bereits ein Clientcrawler auf vorherigen Projekten vorhanden war, musste dieser lediglich in den Key-Attributes angepasst werden, sodass nach den richtigen Hashtags gesucht und die benötigten Daten gespeichert werden. Dieser Crawler zieht die aktuellsten 3000 Tweets zu einem gegebenen Thema raus.

Ein funktionstüchtiger Streamlistener war nicht gegeben, lediglich ein Grundgerüst. Dieser musste also neu geschrieben und angepasst werden. Durch Testläufe konnten Abstürze und falsch-gespeicherte Daten ausfindig gemacht und behoben werden. Der Streamlistener startet sich automatisch selbst neu, sollte er abstürzen, sodass ein reibungsloser Datenfang gegeben ist.

Schritt 2 sieht vor, dass der vorhandene CSVReader auf die neuen Bedürfnisse angepasst wird. Er liest die angegebene CSV Datei ein und verteilt die Daten in

verschiedene Listen und Dictionaries für den Text, die erkannte Sprache, die Tweetanzahl zu jeder Uhrzeit, die TweetID, Koordinaten und das Land. Da der Reader universal für Thorsten und mich funktionieren sollte, haben wir alle Daten für uns beide gespeichert.

Schritt 3 geht in die Vorbereitung für die spätere Analyse. Hier soll eine Stoppwortliste auf die Tweettexte angewandt werden um später leichter die Worte trennen zu können. Benutzt wurden dann schließlich 10 Stoppwortlisten der 10 häufigsten Sprachen in WoW (Anhand von Serversprachen ausgewählt) um möglichst effektiv Stoppworte zu entfernen.

Daraufhin sollen Listen für Schlagworte, Erwähnungen und Hashtags erstellt werden, aber eine zusätzlich, die alle vorherigen vereint ist dazugekommen. Der letzte Punkt deckt sich mit dem nächsten Schritt.

Schritt 4 behandelt die Analyse der aufbereiteten Daten. Zuerst werden Toplisten für Keywords, Hashtags und Erwähnungen erstellt, dann zusätzlich eine Topliste wo alle drei vereint werden. Darauf aufbauend werden dann gewisse Tendenzen herausgearbeitet. So wird die Anzahl der Erwähnungen in Relation zur Anzahl aller Tweets gestellt, und die einzelnen Erwähnungen bis zu einer gewählten Grenze (Ab 25 Erwähnungen) in einer weiteren Kuchengrafik dargestellt. Aus den beiden Ergebnissen wird dann auf der Homepage der prozentuale Anteil an Kontaktaufnahmen zu den Entwicklern/ Support schriftlich festgehalten. Weiterhin wird eine Sentimentanalyse mit Hilfe von Listen positiver und negativer Worte durchgeführt. Hierfür wurden zusätzlich die emotional aussagekräftigsten Begriffe des WoW-Vokabulars hinzugefügt um möglichst flächendeckend Emotionen aufzugreifen. Zuletzt laufen die Top25 Begriffe – in diesem Falle die vereinten – einzeln durch alle Tweets und kriegen alle zusammen vorkommenden Worte zugewiesen. Aus den entstandenen Dictionaries wird eine Häufigkeitsverteilung angefertigt, aus welcher Wortcluster in Form von Wordclouds erstellt werden. On-Top lassen sich alle Resultate auch Sprachspezifisch erstellen, obwohl ursprünglich die einzelnen Sprachen nicht weiter berücksichtigt werden sollten.

Schritt 5 fasst die grafische Darstellung der Analyse noch einmal zusammen, so werden Balkendiagramme für die Häufigkeitsverteilungen der Hashtags, Keywords und Mentions erstellt, eine Multilinechart für die Timeline, Kuchengrafiken für Mentions und Sentimentergebnisse, sowie Wordclouds für die Clusterbildung der Wortzusammenhänge der einzelnen Topterme.

Die Ergebnisse werden auf einer Homepage veröffentlicht. Aufgrund der vielen Grafiken – allein 25 Wordclouds – werden nur die wichtigsten/ interessantesten und aussagekräftigsten Grafiken in den Artikel eingebunden. Die restlichen Grafiken werden im dort verlinkten Github Repository hochgeladen.

Zusätzlich gibt es einen Scatterplot, der alle Wordclouds zusammengefasst und interaktiv darstellt.

Methoden

Für das Sammeln der Daten wurde ein CustomStreamListener der Tweepy Python Library verwendet, welcher die ID, created_at, text, lang, coordinates und user.location in einer .csv Datei abspeichert.

```
class TwitterStreamListener(tweepy.StreamListener):

    tweet_count = 0
    #Streamlistener to fetch the tweets
    def on_status(self, tweet):
        if (not tweet.retweeted) and ('RT @' not in tweet.text):
            tweet.text = tweet.text.replace('|', ' ')
            tweet.text = tweet.text.replace('\n', ' ')

            TwitterStreamListener.tweet_count += 1

            with open('result.csv', 'ab') as resultFile:
                writer = csv.writer(resultFile,
                                    delimiter=";",
                                    lineterminator="\r\n",
                                    encoding='utf-8')

                if tweet.place and tweet.user.location:
                    writer.writerow([tweet.id_str, tweet.created_at, tweet.text, tweet.lang, tweet.place.bounding_box.coordinates, tweet.user.location])
                elif ((not tweet.place) and (tweet.user.location)):
                    writer.writerow([tweet.id_str, tweet.created_at, tweet.text, tweet.lang, 'NaN', tweet.user.location])
                elif ((tweet.place) and (not tweet.user.location)):
                    writer.writerow([tweet.id_str, tweet.created_at, tweet.text, tweet.lang, tweet.place.bounding_box.coordinates, 'NaN'])
                else:
                    writer.writerow([tweet.id_str, tweet.created_at, tweet.text, tweet.lang, 'NaN', 'NaN'])

            print (TwitterStreamListener.tweet_count)
        return
```

Für den Fall, dass der Stream unterbrochen wird, wurden die drei wichtigsten Fehlermeldungen in entsprechenden Funktionen abgefangen. So werden bei Abstürzen, Zeit- und Limitüberschreitungen Benachrichtigungen in der Konsole ausgegeben und der Stream sofort neugestartet.

```
#Restartfunction for errorpopups
def on_error(self, status_code):
    print ('Error: ' +repr(status_code) +'restarting...')
    lambda e:self.start_stream()
    return

#Restartfunction for timeouts
def on_timeout(self):
    print ("Timeout... restarting...")
    lambda e:self.start_stream()
    return True

#Restartfunction for getting too much information at the same time
def on_limit(self, track):
    print ("Limit reached... restarting...")
    lambda e:self.start_stream()
    return
```

Die enstandene .csv Datei wird von einem csvreader eingelesen und in ihre einzelnen Datenteile aufgespalten, welche dann in einem ersten Durchlauf für die weitere Bearbeitung aufbereitet und in Dictionaries abgespeichert werden.

```
with open(fileName, 'rb') as data:
    reader = csv.reader(data, delimiter=";", lineterminator="\r\n", encoding='utf-8')

    # reading row for row
    for row in reader:

        self.split_date = row[1].split() # Splitting Date and Time
        self.split_place = []
        self.regex = re.compile('^\d{18}$')
        self.coords = []

        if (self.regex.match(row[0]) and (row[0] not in self.temp_id)):

            self.temp_id.append(row[0])
            self.unique_tweets += 1

            self.text_Data[row[0]] = row[2].lower()  # save text into dictionary[tweet_id]
            self.lang_Data[row[0]] = row[3].lower()

            self.new_time = self.split_date[1].split(':') # hours, min. und sec.
            self.temp_time_of_day.append(self.new_time[0]) # saving only hours

            if ('NaN' not in row[4]):

                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]

    self.tweets_per_hour = Counter(self.temp_time_of_day) # counting tweets per hour
    self.tweets_per_hour = OrderedDict(sorted(self.tweets_per_hour.items(), key=lambda t: t[0]))
    data.close()
return [self.text_Data, self.lang_Data,self.tweets_per_hour, self.unique_tweets, self.coordinates, self.country]
```

Daraufhin werden die Textdaten an die stopwords.py übergeben, in welcher die Texte durch Stoppwortlisten von den 10 meist genutzten Sprachen in World of Warcraft durchgejagt werden um möglichst viele Stoppworte zu erfassen und entfernen.

```
self.en_Stops = set(stopwords.words('english'))
self.de_Stops = set(stopwords.words('german'))
self.fr_Stops = set(stopwords.words('french'))
self.es_Stops = set(stopwords.words('spanish'))
self.ru_Stops = set(stopwords.words('russian'))

self.fi_Stops = set(stopwords.words('finnish'))
self.no_Stops = set(stopwords.words('norwegian'))
self.sv_Stops = set(stopwords.words('swedish'))
self.nl_Stops = set(stopwords.words('dutch'))
self.it_Stops = set(stopwords.words('italian'))
```

Die Sprachen wurden anhand der Serversprachen und zusätzlich verfügbare Stoppwortlisten der Pythonlibrary NLTK gewählt.

Desweiteren werden die häufigsten nicht von den Stoppwortlisten erfassten Affixe entfernt.

```
self.replaces = ["'re", "'ve", "'ll", "'m", "'t", "d'", "<", "]",
                 "[", ">", ",", ".", ";", "+", "*", "~", "&amp;", '"',
                 "!", "?", "|", "(", ")", ":"]
```

Zuletzt teilt stopwords.py die Daten noch in entsprechende Listen nach Sprache und Texte | Hashtags | Erwähnungen | Schlagworte ein.

```
self.full = [self.full_Text, self.full_Hashtag, self.full_Mentions, self.full_Stripped]

self.english = [self.en_Text, self.en_Hashtag, self.en_Mentions, self.en_Stripped]
self.german = [self.de_Text, self.de_Hashtag, self.de_Mentions, self.de_Stripped]
self.french = [self.fr_Text, self.fr_Hashtag, self.fr_Mentions, self.fr_Stripped]
self.spanish = [self.es_Text, self.es_Hashtag, self.es_Mentions, self.es_Stripped]
self.russian = [self.ru_Text, self.ru_Hashtag, self.ru_Mentions, self.ru_Stripped]

self.finnish = [self.fi_Text, self.fi_Hashtag, self.fi_Mentions, self.fi_Stripped]
self.norwegian = [self.no_Text, self.no_Hashtag, self.no_Mentions, self.no_Stripped]
self.swedish = [self.sv_Text, self.sv_Hashtag, self.sv_Mentions, self.sv_Stripped]
self.dutch = [self.nl_Text, self.nl_Hashtag, self.nl_Mentions, self.nl_Stripped]
self.italian = [self.it_Text, self.it_Hashtag, self.it_Mentions, self.it_Stripped]

return [self.full, self.english, self.german, self.french, self.spanish,
        self.russian, self.finnish, self.norwegian, self.swedish, self.dutch, self.italian]
```

Nach der Stoppwortentfernung und Listenzuweisung werden die entsprechenden Daten in der toplists.py auf ihre Häufigkeitsverteilung analysiert. Hierfür wird sich

wieder der NLTK Lib bedient, um Frequency Distributions und Tokenizer mit Hilfe von Regex zu verwenden. Das Programm erkennt zwei Modi, welche entscheiden ob zwischen Hashtags und Schlagworten unterschieden wird oder nicht. Wird der Modus „remove_hashtag“ angegeben werden bspw. #DemonHunter und DemonHunter zusammengezählt – dieser Modus ist essenziell, bedenkt man die häufige Verwendung von Hashtags in Tweets – der semantische Wert ist oft der gleiche.

```
self.retok = nltk.RegexpTokenizer('\w+|\#?\w+|\@?\w+')

self.full_words = []
self.freq= []
self._fullString = ''

for dicts in args:

    self.lg = ''
    self._String = ''

    for elem in dicts:

        self.temps = []
        self.lg = lang[elem]

        if modus == 'remove_hashtag':
            for word in dicts[elem]:
                if word != 's':
                    self.temps.append(word.replace('#', ''))

        elif modus == 'keep_hashtag':
            for word in dicts[elem]:
                if word != 's':
                    self.temps.append(word)

        self.temp_String = ' '.join(self.temps)
        self._String = self._String + ' ' + self.temp_String


    if self.lg == 'en':
        self.words = self.retok.tokenize(self._String)
        self.words = [x for x in self.words if x != 's']
        self.words = [x for x in self.words if x != 'of']
        self.en_freq = FreqDist(self.words).most_common(25)
        self.full_words.append(self.words)
```

Auch hier lassen sich die Toplisten wieder On-Top bei Bedarf nach Sprachen erstellen. Ausgegeben werden am Ende Listen von Tupeln, die die FrequencyDistributions beinhalten.

Um die Häufigkeitsverteilungen der mit den aus den Toplisten gewonnenen Begriffen zusammenhängenden Worte zu erhalten, wird die entsprechende Topliste an das

Programm wordpairs.py weitergegeben. Um nun die Verteilung zu erarbeiten, werden in einer for-Schleife sämtliche Tweets nacheinander nach dem jeweiligen Begriff durchsucht. Enthält ein Tweet einen solchen Begriff, werden alle Worte anhand einer Liste dem Begriff zugeordnet. Nachdem alle Tweets einer Iterationsrunde durchlaufen sind, werden die gesammelten Worte gezählt und in einem OrderedDict in absteigender Reihenfolge sortiert gespeichert. Nachdem alle 25 Begriffe abgearbeitet wurden, werden die gesammelten Dictionaries in einem übergeordneten Dictionary ausgegeben.

```
def pairings(self, given_words, texts):
    print ('pairing words...')
    self.pairs = {}
    for word in given_words:
        self.elems = {}
        self.mc = []
        self.temp_words = []
        for text in texts.values():
            if (str(word[0]) in text) or ('#'+str(word[0]) in text):
                for elem in text:
                    if (elem != str(word[0])) and (elem != ('#'+str(word[0]))):
                        self.temp_word = str(elem).replace('#', '')
                        self.temp_words.append(self.temp_word)
        self.mc = Counter(self.temp_words).most_common(75)
        for element in self.mc:
            self.elems[element[0]] = element[1]
        self.elems = OrderedDict(sorted(self.elems.items(), key=lambda t: t[1], reverse=True))
        self.pairs[word[0]] = self.elems
    return self.pairs
```

Für die Sentimentanalyse wurden positive und negative Wörterlisten hinzugezogen und nachträglich bearbeitet um die gängigsten emotional behafteten Begriffe des WoW-Jargon aufzufangen. Anhand der Listen werden alle Tweets nach den entsprechend emotional zugeordneten Begriffen durchsucht und diese gezählt. Sollten mehr positive als negative Begriffe vorkommen, wird der entsprechende Tweet als positiv gewertet und vice versa. Im Falle dass positive und negative Begriffe gleichermaßen häufig vorkommen, wird er als neutral betrachtet. Aufgrund der begrenzten Zeichenanzahl bei Twitter und der Durchschnittslänge eines englischen Wortes wurde auf eine genauere Unterscheidung/ Gewichtung verzichtet.

```
    if self.pos_Count_Sent > self.neg_Count_Sent:
        self.pos_Count += 1
    elif self.neg_Count_Sent > self.pos_Count_Sent:
        self.neg_Count += 1
    elif self.pos_Count_Sent == self.neg_Count_Sent:
        self.neut_Count += 1
```

Für die Timeline wurde eine simple Multilinechart ausgewählt, welche die Anzahl der Tweets zur jeweiligen Uhrzeit darstellt. Hierfür wurden mehrere Tage zusammengezählt und auf 24 Stunden verteilt. Eine zweite Linie zeigt den gesamten Durchschnitt der stündlichen Tweets.

```
self.index = [int(x) for x in hours.keys()]
self.values = [float(v) for v in hours.values()]
self.avg = 0

for x in self.values:
    self.avg += x
self.avg = (self.avg/24)

self.fig, self.ax = plt.subplots(1)
self.ax.plot(self.index, self.values, 'k*', label = 'Tweets at hour X')
self.ax.set_xlim(0, 23)
self.ax.fill_between(self.index, 0, self.values, facecolor = 'black', alpha = '0.7')

plt.axhline(y = self.avg, color = 'c', marker = '.', label = 'Average Tweets in an hour')
```

Die Toplisten werden in einer minimalistischen Barchart angezeigt und können Hashtags, Keywords, Mentions und die Zusammenführung derer für einzelne Sprachen übersichtlich darstellen.

Die Ausnahme bilden die Erwähnungen, welche noch zusätzlich für eine intellektuelle Schlussfolgerung der Tendenzen bezüglich der Relation Kontaktaufnahme/ Tweetanzahl zwei Kuchengrafiken erhalten.

```
self.regex = re.compile('(?<=^|(?<=[^a-zA-Z0-9-\.]))@([A-Za-z0-9_]+)')

for elem in tweets:
    self.temp = ' '.join(tweets[elem])
    if re.match(self.regex, self.temp) is not None:
        self.count += 1

self.perc_mentions = round(((100/len(tweets))*self.count), 2)

self.labels = []
self.sizes = []

for elem in mentions:
    (self.key, self.value) = elem
    self.labels.append(self.key+': '+str(self.value))
    self.sizes.append(self.value)

self.colors = cm.Set1(np.arange(25)/25.)
self.fig = plt.figure(1)
self.ax = self.fig.add_subplot(111)

self.patches, self.texts = self.ax.pie(self.sizes, colors=self.colors, shadow=True, startangle=90)

self.lgd = self.ax.legend(self.patches, self.labels, loc='center right', bbox_to_anchor=(1.5, 0.5))
self.ax.axis('equal')
self.fig.savefig(path.join(path.dirname(__file__), 'RESULTS/mentions.png'), transparent=True, bbox_extra
plt.close()
```

Die erste Kuchengrafik stellt die Anzahl der Tweets mit einer oder mehr Erwähnungen in Relation zu allen Tweets dar. Die Zweite zeigt die 25 meist genutzten Erwähnungen auf, welche zusätzlich mindestens 25 mal gezählt wurden. Hieraus werden intellektuell

die Entwickler- und Supportaccounts gelesen und gezählt um diese ebenfalls in Relation zu allen Tweets zu stellen.

Auch für die Sentimentanalyse stellte sich die Kuchengrafik als optimal dar, da hier lediglich jeweils 3 Ergebnisse in Relation zueinander gestellt werden mussten. Positiv, Neutral, Negativ. Die Sentimentanalyse lässt sich ebenfalls On-Top für 3 Sprachen sowie alle zusammen darstellen.

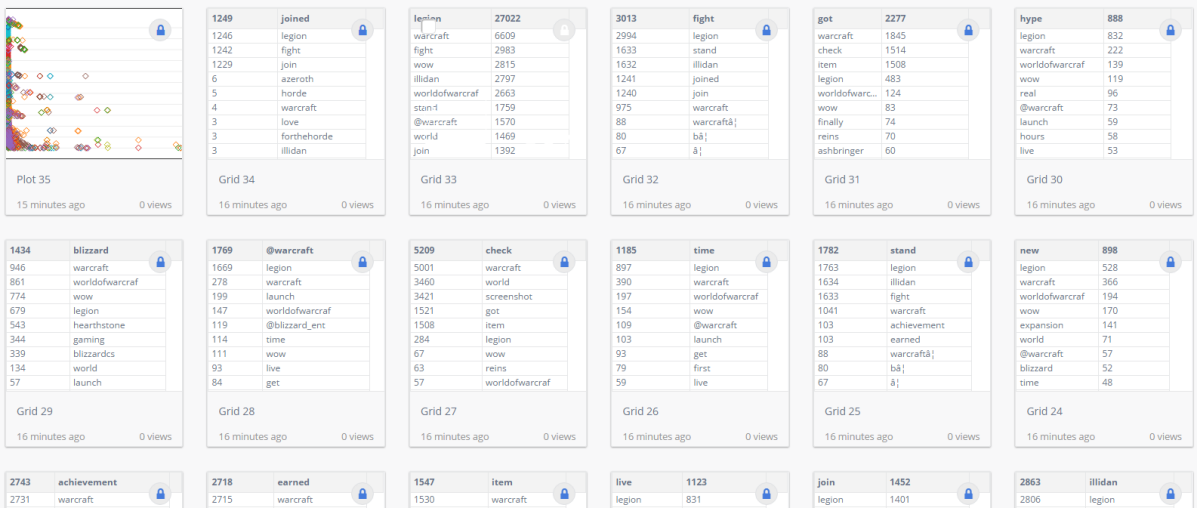
Die Wortzusammenhänge und -häufigkeiten werden in Form mehrerer Wordclouds dargestellt, hierzu werden die Dictionaries in eine Unterfunktion der Wordcloud Lib geschoben und zusätzlich das Twitterlogo als Mask sowie eine passende Font geladen.

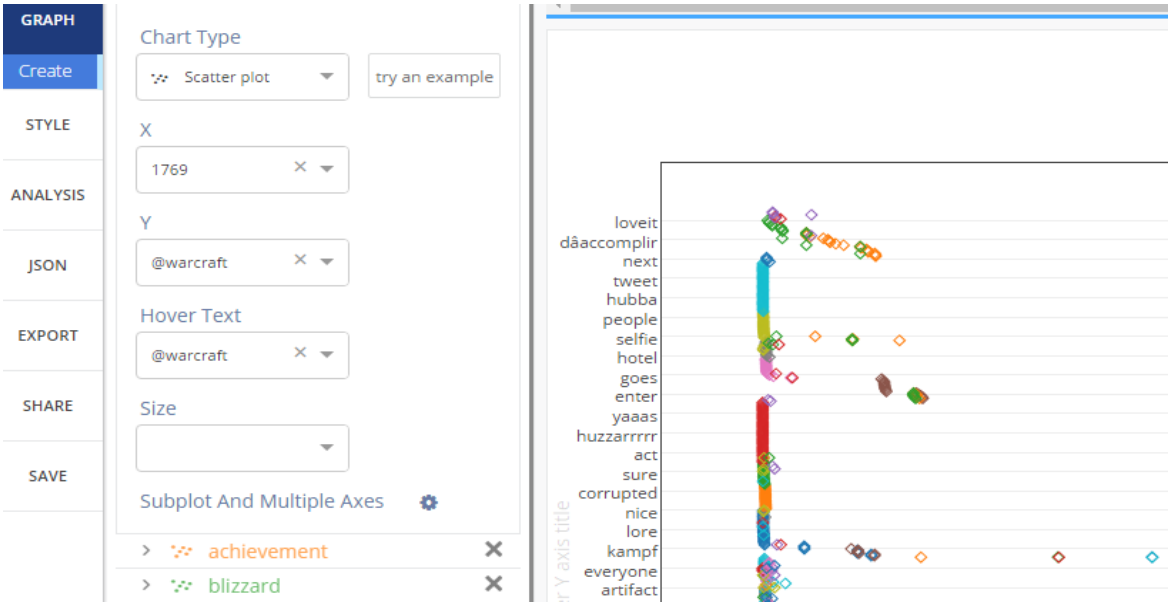
```
#Takes the results from wordpairs and creates a fitting wordcluster/cloud and saves it into WORDPAIRS
def create_Cloud(self, data):
    print ('creating wordpair graph...')
    self.twitter_mask = np.array(Image.open(path.join(path.dirname(__file__), 'MASK/twitter_mask.png')))
    for word in data:
        wordcloud = WordCloud(font_path=path.join(path.dirname(__file__), 'FONT/CabinSketch-Bold.ttf'), re
        wordcloud.generate_from_frequencies(list(data[word].items()))
        wordcloud.to_file(path.join(path.dirname(__file__), 'WORDPAIRS/'+word+'.png'))

    return
```

Zusätzlich zu den Wordclouds wurde ein interaktiver Scatterplot angelegt, welcher alle Worte den Toptermen zugeordnet in einer Grafik anzeigt.

Hierzu wurden die Daten der Wordclouds in eine .csv Datei gespeichert und manuell auf einzelne Dateien aufgeteilt um diese dann in die Seite plot.ly einzuspeisen und zu konfigurieren. Das Ergebnis wird als html Datei gespeichert und ist somit interaktiv in Form von zoomen, verschieben und skalieren verfügbar.

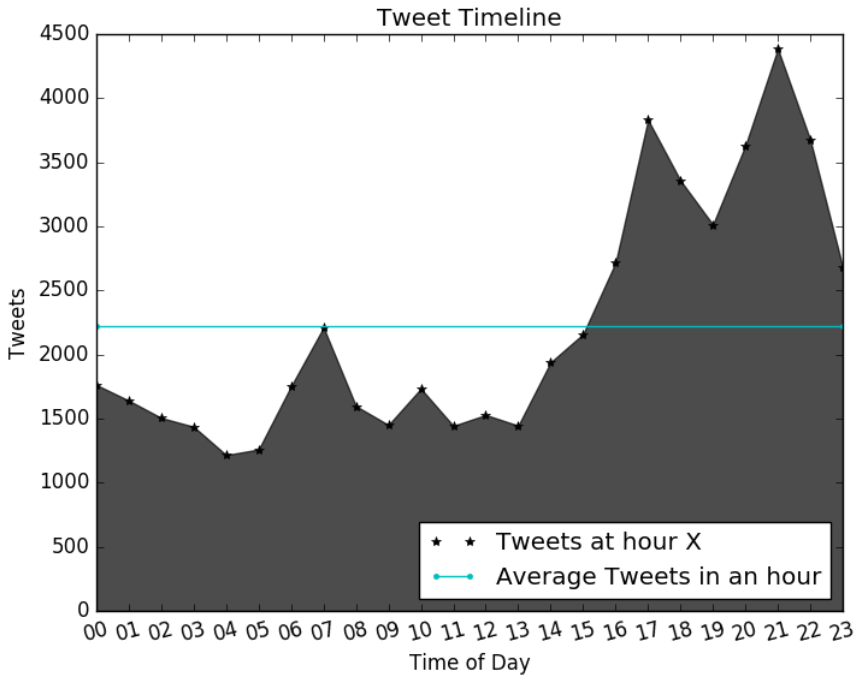




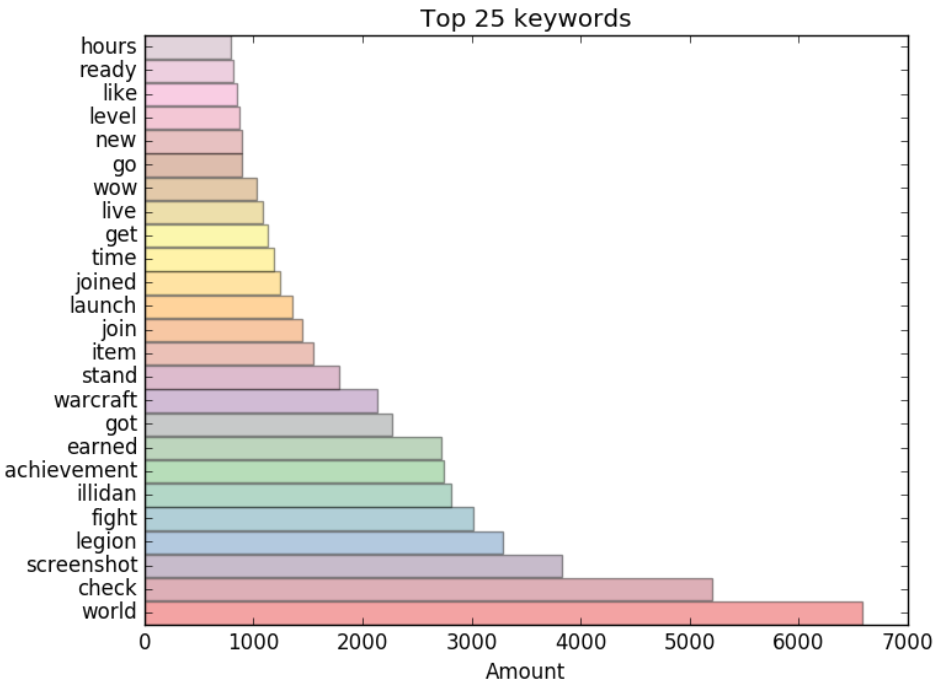
Ein weiteres Ziel im Projektplan war die Einbindung der Ergebnisse via PHP/HTML auf unserer Community-Seite **www.Cede-Maiori.info**. In Zusammenarbeit mit Thorsten Brückner wurde dies folgender maßen umgesetzt: Die Ergebnisse wurden in „News-Form“ auf der Frontpage einbunden. Dieses ermöglich es, dass auch Besucher der Seite direkt die Ergebnisse von uns einsehen können. Kompakt, gut lesbar und immer an oberster Stelle sticky gesetzt. Wir haben die Bilder grafisch der Seite angepasst und interaktive Bokeh Links hinzugefügt. Diese wurde mittels HTML in den PHP Source Code eingefügt auf dem die Seite beruht.

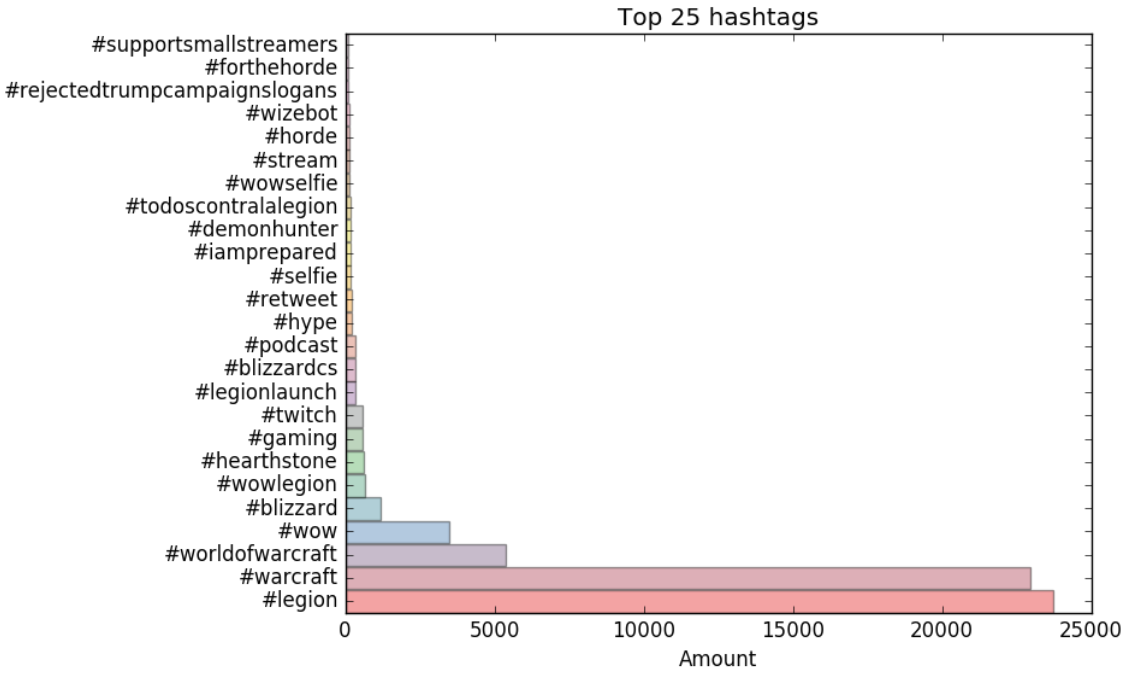
Die Wordpairs mitsamt Worthäufigkeiten werden in einer nach dem Twitterlogo angeordneten Grafik dargestellt. Je größer das Wort ist, desto häufiger kommt es mit dem Suchterm (Dateiname) zusammen in Tweets vor. Es wurde ganz bewusst der Suchterm aus dem Bild gelassen, da der Term sonst als zusammenhängendes Wort verwechselt werden könnte, denn es gab keine Möglichkeit diesen hervorzuheben.
Desweiteren werden Tagclouds erstellt, in denen sämtliche Topbegriffe entfernt wurde. So kann bspw. „legion“ was einer der Top 25 Begriffe ist, nicht in den Grafiken auftauchen. Da diese Top Begriffe sonst in jeder Grafik aufgrund ihrer schieren Anzahl erscheinen aber dennoch zu den zusammenhängenden Begriffen zählen, wurden Grafiken sowohl mit als auch ohne Top-Begriffe angelegt.

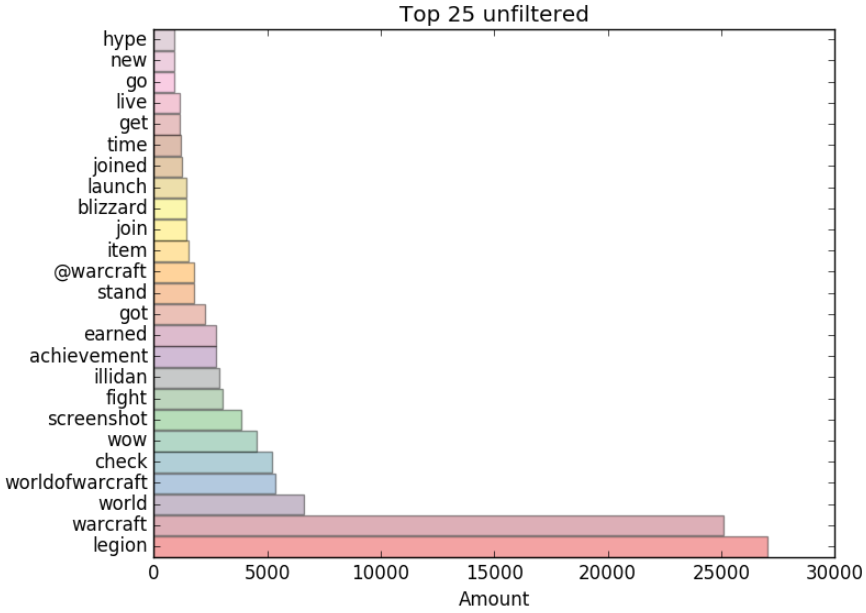
Resultate



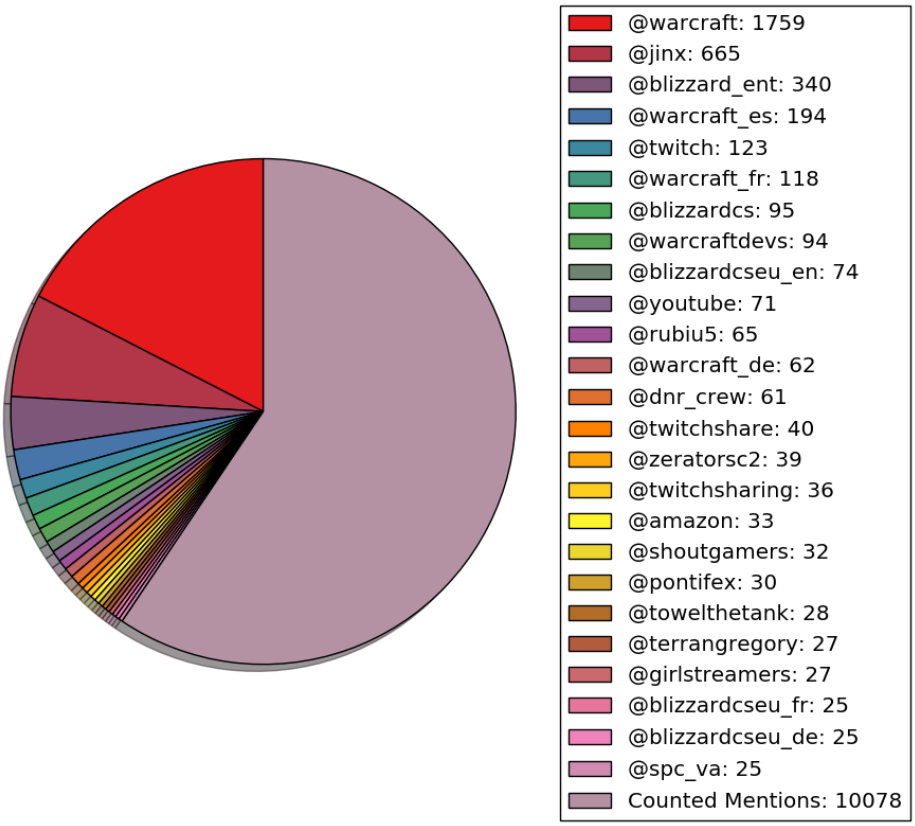
Die Timeline stellt eine Tweetverteilung auf 24 Stunden Basis dar. Im Verlaufe einer Woche wurden 53000 Tweets gesammelt. Im Verlaufe einer Woche wurden 53.000 Tweets gesammelt und die created_at Daten auf ihre Uhrzeit –genauer die Stunden– reduziert. Zusätzlich zeigt die cyanfarbene Linie die durchschnittliche stündliche Tweetanzahl.



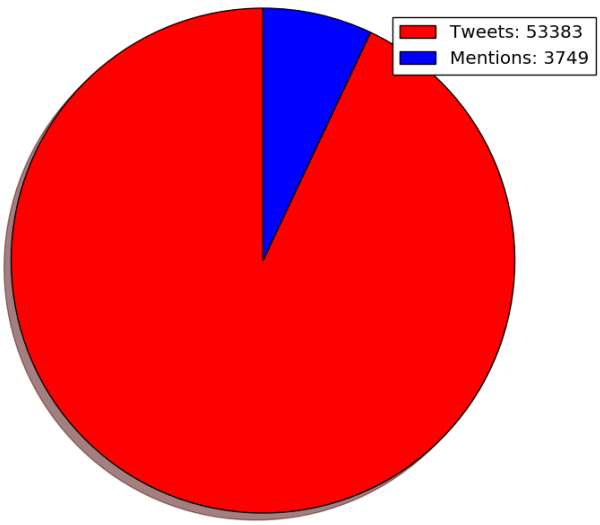




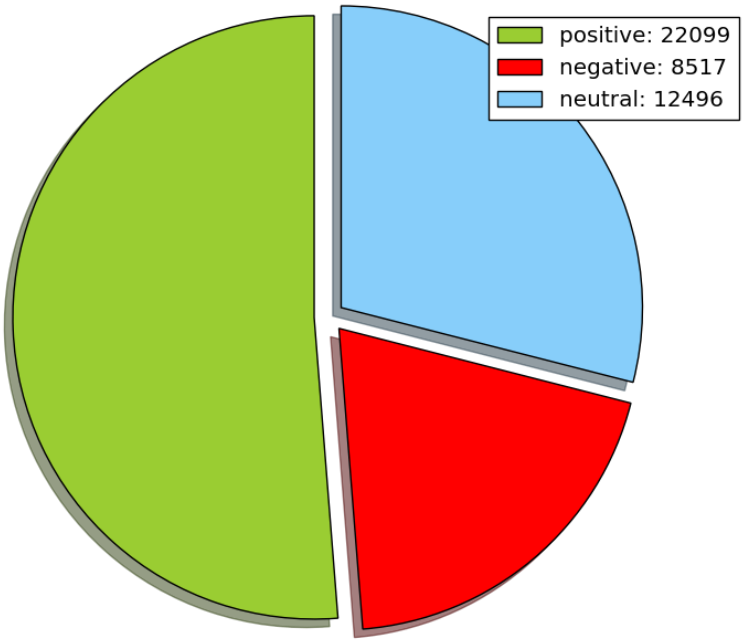
Die Barcharts zeigen die verschiedenen Toplisten von den jeweils 25 meisten Vorkommnissen. Während in der unfiltered Grafik Hashtags und Keywords verschmolzen sind, sodass bpsw. „Legion“ und „#Legion“ zusammengezählt werden, wurden diese für die Hashtag und Keywordgrafiken explizit getrennt. Es lässt sich ebenfalls eine Grafik für Erwähnungen erstellen, da diese aber bereits zwei Piecharts bekommt, wird diese hier ausgelassen.



Die obenstehende Piechart zeigt die 25 meist genutzten Erwähnungen, unabhängig der Tweetanzahl. Hier können durchaus mehrere Erwähnungen in einem Tweet vorkommen. Die Top 25 fasst gesamt 4088 Erwähnungen (40,56%) von allen 10.078 gezählten Erwähnungen.

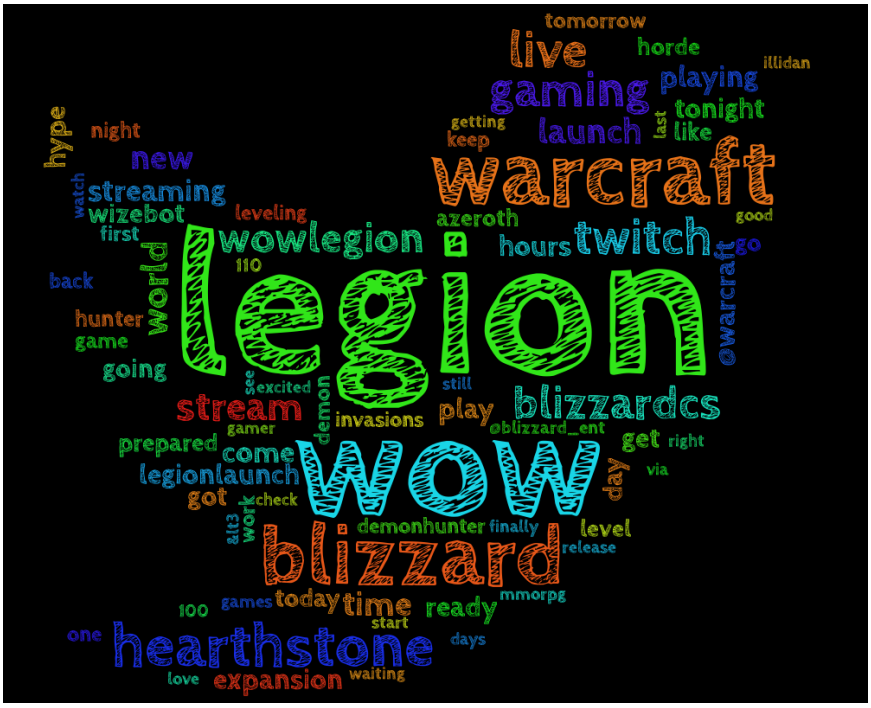


Die zweite Piechart zeigt die Anzahl der Tweets, die mindestens eine Erwähnung enthalten im Zusammenhang zur Anzahl aller Tweets. 7349 (7%) von 53383 Tweets enthalten mindestens eine Erwähnung.



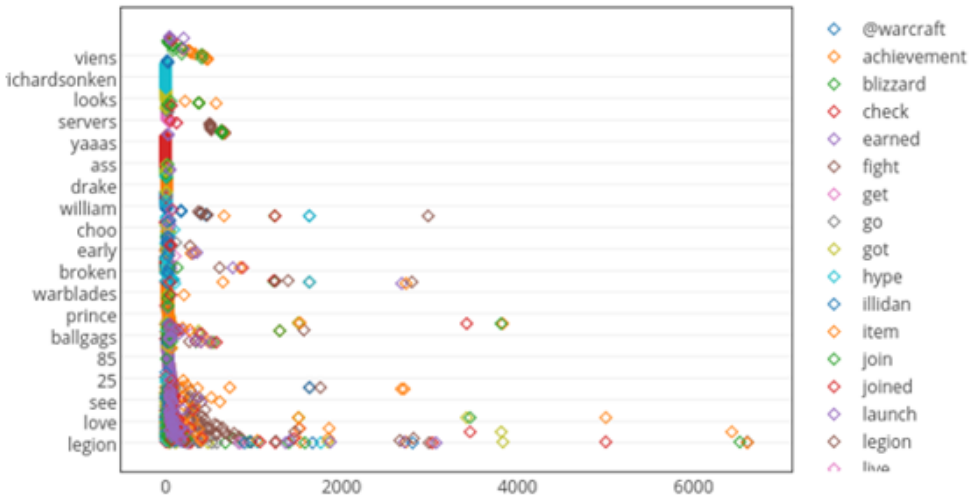
Die Sentimentanalyse wird ebenfalls anhand einer Piechart dargestellt. Zu sehen sind die positiven, negativen und neutralen Tweets der Sprachen Englisch, Deutsch und Spanisch. Diese umfassen mit 43.112 von 53383 Tweets eine prozentuale Abdeckung von 80,76%. Mit 22099 positiven Tweets haben wir eine positive Resonanz von 51,26%, 28,98% der abgedeckten Tweets waren neutral und lediglich 19,76% negativ.

Aufgrund dieser Ergebnisse kann man darauf schließen, dass die Erweiterung „Legion“ überwiegend positiv angekommen ist.



Die Wordpairs mitsamt Worthäufigkeiten werden in einer nach dem Twitterlogo angeordneten Grafik dargestellt. Je größer das Wort ist, desto häufiger kommt es mit dem Suchterm (Dateiname) zusammen in Tweets vor. Es wurde ganz bewusst der Suchterm aus dem Bild gelassen, da der Term sonst als zusammenhängendes Wort verwechselt werden könnte, denn es gab keine Möglichkeit diesen hervorzuheben.

Desweiteren werden Tagclouds erstellt, in denen sämtliche Topbegriffe entfernt wurde. So kann bspw „legion“ was einer der Top 25 Begriffe ist, nicht in den Grafiken auftauchen. Da diese Top Begriffe sonst in jeder Grafik aufgrund ihrer schieren Anzahl erscheinen aber dennoch zu den zusammenhängenden Begriffen zählen wurden die alternativen Grafiken im Ordner WORDPAIRS/ALTER abgelegt statt die Hauptgrafiken darzustellen.



Zusätzlich werden alle 25 Einzelergebnisse zusammengefasst in einem Scatterplot angezeigt, welcher interaktiv zoom- und scrollbar ist – auf die Häufigkeitsskalierung der einzelnen Punkte wurde bewusst verzichtet, da diese in den einzelnen Grafiken bereits abgehandelt wird.

Zusammenfassend lässt sich sagen, dass die Erweiterung und der Hype in der Releasezeit überwiegend positiv angekommen ist. Dass die Begriffe „warcraft“, „worldofwarcraft“, „legion“, „hype“, „item“, „launch“ und „join“ in den Top 25 gelandet sind, überrascht nicht, und auch „Illidan“ ist dieser Charakter immerhin der Themenprotagonist der Erweiterung. Die großen herausstechenden zusammenhängenden Begriffe sind logisch nachzuvollziehen – einige davon gehören immerhin auch unter den Top 25 Begriffen – und häufig sind die Support- und Kommunikationsaccounts des Spieles darunter. In den alternativen Grafiken für zusammenhängende Worte stechen vor allem die Begriffe „level“, „100“, „blizzardcs“, „reins“, „deceiver“, „fel-infused“, „twinblades“, „ashbringer“, „followers“, „ready“, „play“, „finally“, „azeroth“, „horde“ und „expansion“ heraus. Ebenfalls keine Überraschung: Man startet mit level 100 in die Erweiterung und kann auf 110 leveln. BlizzardCS ist der CustomerSupport, Reins bezieht sich auf Zügel von Reittieren, fel-infused ist ein bestimmtes Equipmentset von dem Pre-Event. Deceiver ist ein Synonym für den Kontrahenten von Illidan, Kil’jaeden und die twinblades beziehen sich auf die Waffen von Illidan oder die Waffen der neuen Klasse Dämonenjäger. Ashbringer ist eine legendäre Waffe mit langer Geschichte in WoW, die nun an die Paladine weitergereicht wird. Followers sind NPCs, die man im Laufe der Kampagne rekrutiert und auf Missionen schicken kann und Azeroth ist die Hauptwelt, in der WoW spielt.

Bedienungsanleitung

Für die erfolgreiche Ausführung des Programmes werden die folgenden Libs benötigt:

- Codecs
- Collections
- Matplotlib
- NLTK
- Numpy
- OS
- PIL
- RE
- Tweepy
- Unicodecsv
- Wordcloud

Zum Starten des Streams müssen die Twitter-API Daten in die twitter_keys.py eingetragen werden.

```
# Consumer keys and access tokens, used for OAuth
# https://apps.twitter.com
# Erstelle Dir eine App und verwende die Keys hier
consumer_key = 'consumer key'
consumer_secret = 'consumer secret'
access_token = 'access token'
access_token_secret = 'access token secret'
```

Um mit dem Tweetseinsammeln zu beginnen, muss das Programm save_Tweets_Client.py oder save_Tweets_Stream.py ausgeführt werden.

Für alle anderen Funktionen des Projektes wird die main.py ausgeführt. Nicht erwünschte Aktionen können im Quellcode einfach mit einem Hashtag auskommentiert werden.

Reflexion

Wurde das Ziel erreicht?

Die zentrale Fragestellung „WoW-Hype. Welche Schlagworte, Hashtags und Erwähnungen zum WoW-Legion Release werden in Twitter am Häufigsten verwendet und welche kommen besonders häufig zusammen vor? Lassen sich Tendenzen herausfiltern?“ wurde mit nur geringen Abstrichen in der Darstellung zufriedenstellend beantwortet.

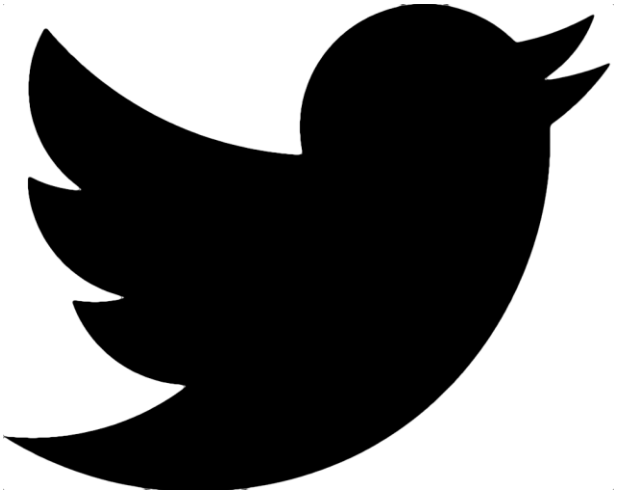
Was ist schiefgelaufen?

Für die Frage nach den Tendenzen im informationellen Bereich gelang es mir nicht, eine effektive Methode herauszuarbeiten, diesen abzudecken.

Die ursprüngliche Idee, die Wortzusammenhänge in networkx umzusetzen, ist an zwei Sachen gescheitert: 1. Ließ sich nur ein Layout ausfinding machen, das sinnvoll für die Darstellung der gegebenen Informationen ist 2. Ließ sich dieses Layout nicht anwenden. Das Layout wirft enorme Probleme mit der Library GraphViz auf, die trotz mehrer recherchierter Lösungsvorschläge nicht behoben werden konnten. Sogar Befehle aus den offiziellen installationsscripts ließen sich nicht ausführen, da diese nicht gefunden wurden.

Lösungen?

Als Alternative für die grafische Darstellung der Wortcluster wurde die Library WordCloud verwendet, welche großen Anklang in der Pythoncommunity zur Darstellung von zusammenhängenden Worten mitsamt ihrer Häufigkeiten für die Plattformen Reddit und Twitter findet. Allerdings wirkte diese Library vorerst als nicht zu gebrauchen, da sie eine eigene Stoppwortliste und FreqDist anwendet, die gegebenen Daten aber bereits Stoppworte entfernt haben und auf FreqDistlisten aufgeteilt sind. Nach einiger Suche innerhalb der Library kam dann eine Funktion auf, die sich direkt mit Frequencies speisen lässt und somit für die Ergebnisse verwendbar ist. Die WordCloud lässt sich zudem anhand von „Masks“, Wortanordnung und Fonts in ihrer Darstellungsart anpassen. Als Mask gewählt wurde passend zum Projekt das Twitterlogo:



Dargestellt werden die Ergebnisse in Form der eingespeisten Wörter, welche mit ihrer gegebenen Häufigkeit in Größe skalieren.

Da diese Darstellung allerdings in der sinnvollsten Variante (Topterm als Dateiname und nicht im Bild selbst – das Bild soll die zusammenhängenden Worte zu dem Topterm darstellen) in 25 Einzelbilder resultiert und diese nicht alle in einen Artikel der Homepage passen, wurde zusätzlich eine Scatterplot-Grafik mit der website plot.ly erstellt, welche alle 25 Ergebnisse in einer Grafik darstellt. Hier wurde jedoch auf Größenskalierung verzichtet um die Übersichtlichkeit in der schieren Masse und Unterschiedlichkeiten der Daten zu gewährleisten.

Fazit

Das Projekt stellte sich als aufwendiger heraus als ursprünglich geplant – besonders die grafische Darstellung bereitete hier und da Probleme und zwang zu Alternativen. Für die optionale Erweiterung der Wortzusammenhänge, sodass alle Worte auf ihre zusammenhängenden Worte analysiert werden, fehlte leider die Zeit. Alles in Allem wurde das Ziel jedoch zufriedenstellend nach dem Plan erfüllt. Die minimalen Ziele wurden abgesehen von der informationellen Tendenz erfüllt und teilweise On-top Features eingefügt. Durch das Seminar habe ich gelernt, dass ich mich in Zukunft besser vorbereiten muss in Form von Ersatzplänen. Es kostet unmengen Zeit, wenn man merkt, dass man eine gewählte Option nicht verwenden kann und erst im Nachhinein nach Alternativen suchen muss. Auch habe ich mögliche auftretende Komplikationen schlichtweg unterschätzt, was mein Zeitmanagement komplett durcheinander gebracht hat.