Published in Towards Data Science

Daniel Foley Follow
Oct 21, 2018 · 8 min read ★ · Listen

Building an ETL Pipeline in Python

Introduction

In my last post, I discussed how we could set up a script to connect to the Twitter API and stream data directly into a database. Today, I am going to show you how we can access this data and do some analysis with it, in effect creating a complete data pipeline from start to finish. Broadly, I plan to extract the raw data from our database, clean it and finally do some simple analysis using word clouds and an NLP Python library.

Let's think about how we would implement something like this. To make the analysis as general as possible I am going to take an object oriented approach by creating a **TweetObject** class and methods of this class to perform the tasks above. There are a few important libraries that we will use such as the NLTK (Natural Language Toolkit)

Let's take our first look at the python code.

```
import mysql.connector
from mysql.connector import Error
import os
import re
import pandas as pd
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
import nltk
from wordcloud import WordCloud, STOPWORDS
import numpy as np
import matplotlib.pyplot as plt
from textblob import TextBlob


class TweetObject():

        def __init__(self, host, database, user):
                self.password = os.environ['PASSWORD']
                self.host = host
                self.database = database
                self.user = user



        def MySQLConnect(self,query):
                """
                Connects to database and extracts
                raw tweets and any other columns we
                need
                Parameters:
                ----------------
                arg1: string: SQL query
                Returns: pandas dataframr
                ----------------
                """

                try:
                        con = mysql.connector.connect(host = self.host, database = self.database, \
                                user = self.user, password = self.password, charset = 'utf8')

                        if con.is_connected():
                                print("Successfully connected to database")

                                cursor = con.cursor()
                                query = query
                                cursor.execute(query)

```

```
        except Error as e:
            print(e)

        cursor.close()
        con.close()


        return df
```

part5_Twitter.py hosted with ❤ by GitHub view raw

First off, we import the necessary libraries. Like my previous post, we need to import the mysql-connector library to connect to our database. The TweetObject class will initialise some important parameters allowing us to connect to our database. The **MySQLConnect** method takes in a SQL query, executes it and returns a pandas data frame. Pandas is a really great library for any data analysis tasks and makes manipulating data really easy so I would recommend any aspiring data analysts/scientists get familiar with this library.

Natural Language Processing: Cleaning the Tweets

Now that we have a method that queries our database and returns the data what are we going to do with it? Let's take a moment to talk a little bit about natural language processing (NLP). In general, text data requires some pre-processing before we can feed it to a machine learning algorithm. We need to put it into a format that an algorithm can understand. This generally involves some of the following tasks.

Pre-Processing steps in NLP:

1. Normalisation.

4. Lemmatisation

5. TF-IDF.

This is certainly not an exhaustive list but these are the kinds of techniques that would apply to most NLP tasks. OK, let's explain what some of these concepts are and why we need to use them. Standardising words into lowercase is a normal step in NLP as we do not want our algorithm to consider PYTHON and python as two different words. Converting all words into the same case avoids these issues. Another step we want to take is to remove any irrelevant material from our text. One way to do this is to remove punctuation such as commas, full stops and stop words such as 'i', 'is', 'the'. A more detailed list of the stop words in the NLTK package can be seen here. We should also think about **Tokenisation** which is a process that essentially splits up text into meaningful chunks or tokens (TextBlob does this for us automatically). The final pre-processing technique that we will use is **Lemmatisation**. This essentially converts a word into its 'canonical form'. In other words pythons will become python and walked becomes walk.

One technique that is quite useful but we do not use here is Term Frequency-Inverse Document Frequency or **TF-IDF.** It is an information retrieval technique which allows us to identify the relative importance of words in a document. Broadly speaking, if a word occurs many times in a document it is likely to be important. However, if it also appears frequently across multiple documents then it may just be a common word and not in fact very meaningful. TF-IDF tries to account for this and returns an overall score of importance for each word. It is a widely used technique when trying to quantify what a document is about and tends to be used with algorithms such as **Gaussian Mixture Models (GMM)**, **K-means** or **Latent Dirichlet Allocation (LDA).**

There are many other techniques that we can employ that might improve our results such as stemming and n-grams for example but I will not go into these here. There are a number of resources online if you want to take a deeper dive into NLP.

Now let's get back to our code. The **clean_tweets** method below implements some of the techniques mentioned above. In particular, it normalises all words into lower case, splits the words, lemmatises, and only keeps words that are not in the stop words list. It also gets rid of some common HTML that tends to appear in tweets that won't really help our analysis.

```
def clean_tweets(self, df):

                """
                Takes raw tweets and cleans them
                so we can carry out analysis
                remove stopwords, punctuation,
                lower case, html, emoticons.
                This will be done using Regex
                ? means option so colou?r matches
                both color and colour.
                """

                # text preprocessing
```

```
        df['len'] = None
        for i in range(0,len(df['tweet'])):
                # get rid of anything that isnt a letter

                exclusion_list = ['[^a-zA-Z]','rt', 'http', 'co', 'RT']
                exclusions = '|'.join(exclusion_list)
                text = re.sub(exclusions, ' ' , df['tweet'][i])
                text = text.lower()
                words = text.split()
                words = [wordnet_lemmatizer.lemmatize(word) for word in words if not word in stopword_list]
                 # only use stem of word
                #words = [ps.stem(word) for word in words]
                df['clean_tweets'][i] = ' '.join(words)


        # Create column with data length
        df['len'] = np.array([len(tweet) for tweet in data["clean_tweets"]])



        return df
```

part6_Twitter.py hosted with ❤ by GitHub view raw

Calculating Sentiment

Now that we have a method to clean our tweets, we can create a method to calculate sentiment. The TextBlob library makes sentiment analysis really simple in Python. All we need to do is pass our text into our TextBlob class, call the **sentiment.polarity** method on the object and it returns either 1, 0 or -1 corresponding to positive, neutral and negative sentiment respectively. The TextBlob class implements a number of text processing functions by default if no additional parameters are passed to it so let's have a quick look at these to understand what is going on under the hood.

There are four parameters, the **tokenizer**, **np_extractor**, **pos_tagger** and **analyser** that if left blank, default to certain methods. The tokenizer defaults to the **WordTokenizer** method which splits the text up into a list of words. The **FastNPExtractor** method is called if we leave np_extractor blank which according to the documentation returns a list of noun phrases which can be quite helpful in identifying what a particular sentence or tweet is about.

Next, the NLTK_tagger is called which identifies parts of speech (POS) such as whether a word is a noun, verb or adjective. Finally, the analyser defaults to the pattern analyser which returns our polarity score. As we can see, the code for this sentiment calculation is really concise. The polarity score is returned on a scale from -1 to 1 which we convert into a sentiment score based on its value.

We also create methods which save our results to a CSV and create a word cloud. I really like using word clouds as they are an effective way of summarising text data. They provide a pretty visualisation of word counts in the text where the bigger words correspond to a higher count (word appears more often). This makes it much easier to get a feel for the kinds of things people are tweeting about.

```
def sentiment(self, tweet):
                """
                This function calculates sentiment
                from our base on our cleaned tweets.
                Uses textblob to calculate polarity.
                Parameters:
                ----------------
                arg1: takes in a tweet (row of dataframe)
                ----------------
                Returns:
                        Sentiment:
                        1 is Positive
                        0 is Neutral
                       -1 is Negative
                """

                analysis = TextBlob(tweet)
                if analysis.sentiment.polarity > 0:
                        return 1
                elif analysis.sentiment.polarity == 0:
                        return 0
                else:
                        return -1




        def save_to_csv(self, df):
                """
                Save cleaned data to a csv for further
                analysis.
                Parameters:
                ----------------
                arg1: Pandas dataframe
                """
                try:
                        df.to_csv("clean_tweets.csv")
                        print("\n")
                        print("csv successfully saved. \n")
```

```



    def word_cloud(self, df):
        """
        Takes in dataframe and plots a wordclous using matplotlib
        """
        plt.subplots(figsize = (12,10))
        wordcloud = WordCloud(
                                        background_color = 'white',
                                        width = 1000,
                                        height = 800).generate(" ".join(df['clean_tweets']))
        plt.imshow(wordcloud)
        plt.axis('off')
        plt.show()
```

part7_Twitter.py hosted with ❤ by GitHub view raw

Main Method

Next up, we have our main method which creates our object and calls the appropriate methods. We first create an object of the TweetObject class and connect to our database, we then call our **clean_tweets** method which does all of our pre-processing steps. The final steps create 3 lists with our sentiment and use these to get the overall percentage of tweets that are positive, negative and neutral. Before we run this we need to make sure our **SQL server in running**. Otherwise, we won't be able to connect to the database and retrieve our data.

```
if __name__ == '__main__':

    t = TweetObject( host = 'localhost', database = 'twitterdb', user = 'root')

    data  = t.MySQLConnect("SELECT created_at, tweet FROM `TwitterDB`.`Golf`;")
```

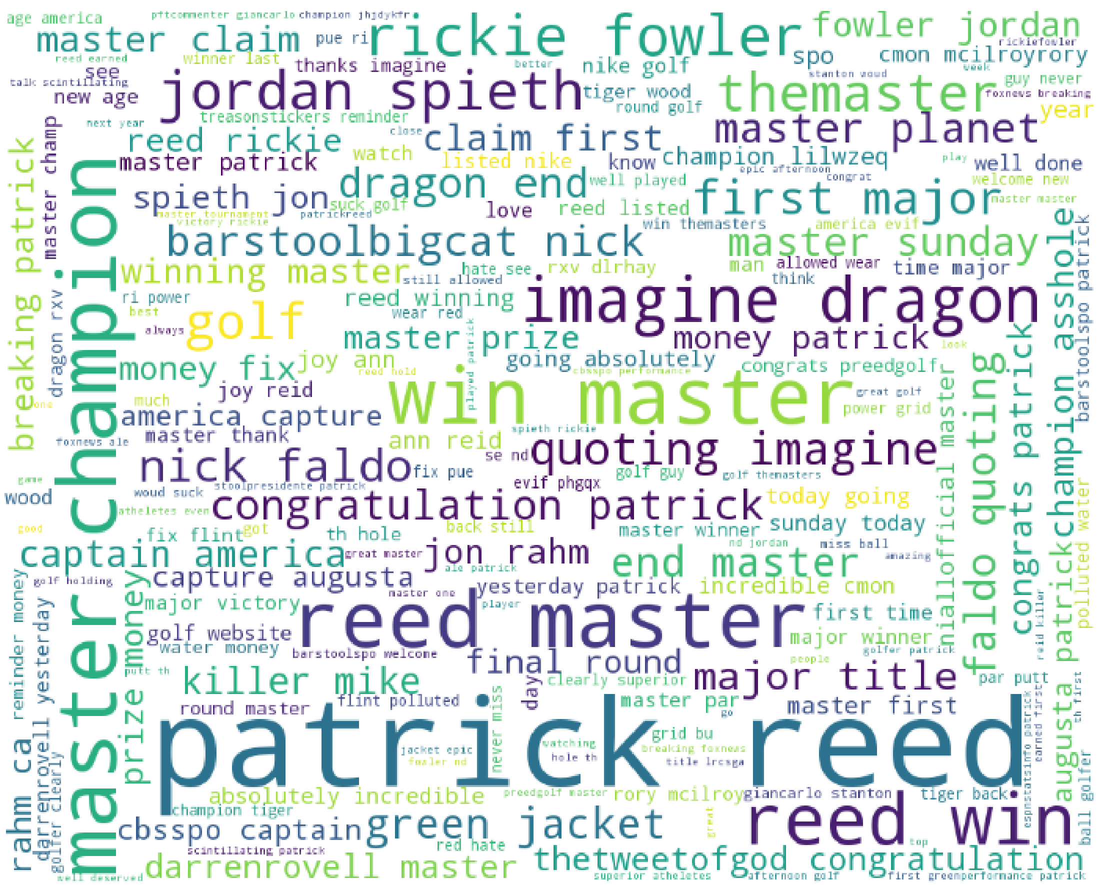
```python
        pos_tweets = [tweet for index, tweet in enumerate(data["clean_tweets"]) if data["Sentiment"][index] > 0]
        neg_tweets = [tweet for index, tweet in enumerate(data["clean_tweets"]) if data["Sentiment"][index] < 0]
        neu_tweets = [tweet for index, tweet in enumerate(data["clean_tweets"]) if data["Sentiment"][index] == 0]

        #Print results
        print("percentage of positive tweets: {}%".format(100*(len(pos_tweets)/len(data['clean_tweets']))))
        print("percentage of negative tweets: {}%".format(100*(len(neg_tweets)/len(data['clean_tweets']))))
        print("percentage of neutral tweets: {}%".format(100*(len(neu_tweets)/len(data['clean_tweets']))))
```


Results

OK, let’s see what this code produced. As I said in my previous post, I am a bit of a golf fan so I decided to collect tweets during the final round of the masters in 2018. As we can see from the word cloud below, Patrick Reed is the most frequent name popping up which is probably not surprising given he won the tournament. We also see a few other golfers names such as Jordan Speith and Ricky Fowler who finished right behind Reed in the leaderboard. Although this isn't the most insightful graph in the world I really am a fan of using word clouds to try and draw some initial insights from data. It is not entirely obvious what the general sentiment of these tweets are looking at this word cloud but that is why we have the TextBlob library. Below this, we have the results from our sentiment calculation. In particular, we calculated the percentage of tweets that fall into the three categories above.

Our results from the sentiment score indicate that the majority of tweets are positive at around 48%. There is also a large group of people who seem to be quite neutral at 39%. Overall we can glean from this that the tweets are broadly positive but not by much. One weakness of the approach here is that we may have inadvertently grabbed tweets that aren't at all related to the golf tournament but simply contain some of our keywords. This is one thing to be wary of when doing any analysis like this.




```
csv successfully saved.

Saved Sentiment CSV...
percentage of positive tweets: 48.35
percentage of negative tweets: 11.93
percentage of neutral tweets: 39.72
Dans-MacBook-Air:twitter-analysis danfoley$
```

Conclusion

This concludes our two-part series on making a ETL pipeline using SQL and Python. Although our analysis has some advantages and is quite simplistic, there are a few disadvantages to this approach as well. Like with all types of analysis, there are always tradeoffs to be made and pros and cons of using particular techniques over others. Ultimately this choice will be down to the analyst and these tradeoffs must be considered with respect to the type of

One disadvantage of the approach we have taken is that we have used an off the shelf algorithm. There is no guarantee that our results are very accurate and unfortunately no way to tell without going through the tweets we collected. One way we could handle this is to use a sentiment algorithm that was specifically trained on tweets which would likely give us improved results.

We could also take this analysis a step further and try some unsupervised learning methods such as clustering. This would help us find groups of tweets that are similar and could give us deeper insights into our dataset. GMM would also be an interesting technique to try. This algorithm attempts to do the same thing as clustering but has the added advantage of being more flexible by allowing us to assign tweets to multiple groups with certain probabilities. This is great as we can incorporate a level of uncertainty into our estimates. I will, however, leave these techniques for a future post.

Recommended Course: Data Engineering, Big Data, and Machine Learning on GCP

Full Python Code

```
import mysql.connector
from mysql.connector import Error
import os
import re
import pandas as pd
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import nltk
from wordcloud import WordCloud, STOPWORDS
import numpy as np
import matplotlib.pyplot as plt
from textblob import TextBlob



class TweetObject():


        def __init__(self, host, database, user):
                self.password = os.environ['PASSWORD']
                self.host = host
                self.database = database
                self.user = user



        def MySQLConnect(self,query):
                """
                Connects to database and extracts
                raw tweets and any other columns we
                need
```

```
        """

        try:
            con = mysql.connector.connect(host = self.host, database = self.database, \
                user = self.user, password = self.password, charset = 'utf8')

            if con.is_connected():
                print("Successfully connected to database")

                cursor = con.cursor()
                query = query
                cursor.execute(query)

                data = cursor.fetchall()
                # store in dataframe
                df = pd.DataFrame(data,columns = ['date', 'tweet'])



        except Error as e:
            print(e)

        cursor.close()
        con.close()

        return df



    def clean_tweets(self, df):

        """
        Takes raw tweets and cleans them
        so we can carry out analysis
        remove stopwords, punctuation,
        lower case, html, emoticons.
        This will be done using Regex
        ? means option so colou?r matches
        both color and colour.
        """

        # Do some text preprocessing
        stopword_list = stopwords.words('english')
        ps = PorterStemmer()
        df["clean_tweets"] = None
        df['len'] = None
        for i in range(0,len(df['tweet'])):
            # get rid of anythin that isnt a letter

            exclusion_list = ['[^a-zA-Z]','rt', 'http', 'co', 'RT']
            exclusions = '|'.join(exclusion_list)
```

```
            #words = [ps.stem(word) for word in words]
            df['clean_tweets'][i] = ' '.join(words)


        # Create column with data length
        df['len'] = np.array([len(tweet) for tweet in data["clean_tweets"]])



        return df



    def sentiment(self, tweet):
        """
        This function calculates sentiment
        on our cleaned tweets.
        Uses textblob to calculate polarity.
        Parameters:
        ----------------
        arg1: takes in a tweet (row of dataframe)
        """

        # need to improce
        analysis = TextBlob(tweet)
        if analysis.sentiment.polarity > 0:
            return 1
        elif analysis.sentiment.polarity == 0:
            return 0
        else:
            return -1




    def save_to_csv(self, df):
        """
        Save cleaned data to a csv for further
        analysis.
        Parameters:
        ----------------
        arg1: Pandas dataframe
        """
        try:
            df.to_csv("clean_tweets.csv")
            print("\n")
            print("csv successfully saved. \n")


        except Error as e:
            print(e)
```

```
                plt.subplots(figsize = (12,10))
                wordcloud = WordCloud(
                                background_color = 'white',
                                width = 1000,
                                height = 800).generate(" ".join(df['clean_tweets']))
                plt.imshow(wordcloud)
                plt.axis('off')
                plt.show()





if __name__ == '__main__':

        t = TweetObject( host = 'localhost', database = 'twitterdb', user = 'root')

        data  = t.MySQLConnect("SELECT created_at, tweet FROM `TwitterDB`.`Golf`;")
        data = t.clean_tweets(data)
        data['Sentiment'] = np.array([t.sentiment(x) for x in data['clean_tweets']])
        t.word_cloud(data)
        t.save_to_csv(data)

        pos_tweets = [tweet for index, tweet in enumerate(data["clean_tweets"]) if data["Sentiment"][index] > 0]
        neg_tweets = [tweet for index, tweet in enumerate(data["clean_tweets"]) if data["Sentiment"][index] < 0]
        neu_tweets = [tweet for index, tweet in enumerate(data["clean_tweets"]) if data["Sentiment"][index] == 0]

        #Print results
        print("percentage of positive tweets: {}%".format(100*(len(pos_tweets)/len(data['clean_tweets']))))
        print("percentage of negative tweets: {}%".format(100*(len(neg_tweets)/len(data['clean_tweets']))))
        print("percentage of neutral tweets: {}%".format(100*(len(neu_tweets)/len(data['clean_tweets']))))
```

part9_Twitter.py hosted with ❤ by GitHub view raw

LinkedIn: https://www.linkedin.com/in/daniel-foley-1ab904a2/

some of the links in this post are affiliate links