Technical Manual

Project Name:	Analyzing Reddit Behaviour
Author:	Rachel Solomon
Student No:	14345381
Supervisor:	Suzanne Little
Date Completed:	20/05/2018

Abstract

‘Analyzing Reddit Behaviour’ [ARB] is a web application which aims to highlight and contrast the differences between users’ behaviour within the various subreddits of ‘www.reddit.com’.

It acquires Reddit comments and performs statistical and sentiment analysis upon the data in order to extract meaningful conclusions. It can offer an insight into the types of users that would frequent both Reddit and different subreddits inclusively. ARB applies machine learning predictive models in an attempt to train them to categorize Reddit comments into appropriate subreddits and to also predict their score.

This tool can be used for research and educational purposes but also for learning more about the public’s latest reaction to various topics.

Table of contents

Motivation

Project ambition

This project revolves around analyzing data coming from the website ‘www.reddit.com’. For those who are unfamiliar with Reddit, Reddit is a social media site where users can come together and have forum-like discussions. Reddit is essentially made up of discussion threads in user-created areas of interest known as ‘Subreddits’.

Analyzing Reddit Behaviour is a web application which showcases the contrast between user behaviour throughout the various Subreddits. It highlights the overlap between negative and positive user interactions as well as a range of aspects such as popular language style. Furthermore, it also incorporates machine learning predictive models.

Personal Interest

The motivation for this project stems from both a familiarity with being an active user of the website and also from a curiosity to examine whether or not having an insight into a website with such a large user interaction, and somewhat controversial content, would provide us any valuable insights.

Being an avid user of Reddit with a keen interest in learning more about data analytics and machine learning motivated me to pursue and develop this project idea.This interest began when I was introduced to Twitter analysis for my third year project and I studied the Data Mining and Warehouses last semester. Whilst studying this module, I was introduced to using machine learning as a means for prediction and for trend analysis. As a result, I am quite eager and interested in expanding my knowledge in the realm of data analysis especially with regards to with social media.

Purpose

Business Need
Social media in recent times has become a goldmine for enabling businesses and corporations to catch a glimpse into the minds of their current and/or potential customers. Due to this fact, I believed that it would be interesting to see what conclusions can be drawn from user interactivity on Reddit and felt that there would be some stark differences in behaviour and response amongst the various communities.

Using this, a brand could potentially view user response to new products such as games or movies through Reddit discussions.

Educational Purpose
This web application can also be useful for both research and education purposes. In terms of education, it shows how much positive and/or negative behaviour exists within Reddit and which Subreddits are more prone to negative user behaviour.

Research

Existing Tools

I researched currently existing tools that would have the same or, at least, similar functionality to this project. While I didn't come across any tools that would have the exact same functionality, there are a few similar projects out there which work with Reddit data. For instance, there are tools that focus on analyzing specific users' behaviours based off their username and their posting history.

Reddit API & Gathering Data

In order to obtain the data from Reddit, initial research indicated that there were a couple of options as to do this. One of the options was to utilize a predefined dataset which contained all of the publicly available Reddit comments. Naturally, considering this contained all of the publicly available Reddit comments (4TB), it was quite large and, realistically, could only be used in chunks. The second option was to scrape and preprocess the data making use of the Reddit API via the use of a Python wrapper called *Praw*.

Research was required in order to learn more about *Praw* and how it worked. It was important to understand the various components and potential limitations that *Praw* could face. Once understood, it was clear that using *Praw* would make it easier to access and extract data from Reddit API and also enabled the use of gathering data based off of timestamp.

Database

When working with data, it is important that you are making use of the most appropriate database for the given application. During my research, I looked into what database would suit the project best and narrowed it down to between SQLite and MongoDB. I was interested in learning more about MongoDB as it is quite a popular choice for data analytics but I was more familiar with the SQL query language.
Eventually I decided that I would use SQLite as I had more experience with complex SQL queries and it was easier as well as being more light-weight compared to MongoDB. This suited the time constraints that came with the project's implementation.

Sentiment Analysis

As Sentiment Analysis played a major part in the main implementation idea, it was necessary to ensure that enough research was done in order to have an overview as to how it should be developed. Although I had a brief introduction to sentiment analysis last year in my third year project, I required more in-depth research as what we produced was quite basic and only appropriate for that of a third year project.

For my research, I read a number of papers relating to Natural Language Processing and analyzing the sentiment of speech. I also researched more about the Natural Language

Toolkit (NLTK) Python package which would provide me with helpful ways for preprocessing data. As a result of my research, I opted to use a Bag of Words approach.

Machine Learning

Whilst I had a brief introduction to Machine Learning, it was necessary to conduct more research into machine learning techniques for predictive modelling as I would be using it with textual data rather than that of numerical data.

Textual data requires more preprocessing and specific algorithms tend to perform better than others. I researched various algorithms and how to preprocess text data with the likes of using a TF-IDF Vectorizer and Count Vectorizer.

Count Vectorizer

	also	love	programming
love programming	0	1	1
programming also love	1	1	1

TF-IDF Vectorizer

	also	love	programming
love programming	0.000000	0.707107	0.707107
programming also love	0.704909	0.501549	0.501549

Count Vectorizer

Count Vectorizer works by giving equal weight to all of the words in a sentence. This means that a word is converted to a column, for instance in a dataframe, and for each document it is equal to 1 if it is present in that document. If it is not, it is equal to 0.

TF-IDF Vectorizer

TF-IDF differs from Count Vectorizer as it reflects how important that word that word is to the document with respect to the entire dataset. It is made up of two parts, Term Frequency (TF) and Inverse Document Frequency (IDF). TF represents the frequency count for a unique word in each document and IDF takes that unique word and counts how many times it appears in all documents.

Algorithms

I decided to make use of Naive Bayes and a Linear SVM model for predicting the Subreddit of a comment. For predicting the score, I decided I would opt for AdaBoost Classifier, RandomForestClassifier and Linear SVC. Extensive research was required in order to learn how to incorporate these algorithms correctly for the best results.

Front-End Development

For the framework of the web application, I investigated as to what would be the most ideal and appropriate framework for this project. Having experienced using the Django framework, I decided that I would use Flask this time instead. Research regarding the various components of Flask and how they interact was necessary as it was a completely new tool to me. It was necessary to learn how to use it and how to run it locally.

I also decided that I would want to implement the visual aspects of the analysis through JavaScript and as I had very little experience and knowledge with using JavaScript, research

was necessary in order to provide me with a basic understanding of how it worked. I personally wanted to gain more experience working with JavaScript as it is both very powerful and popular in modern web development.

For the overall design, I decided to make use of the Bootstrap UI framework in order to develop an appealing interface for the web application. I had little experience working with Bootstrap before, so to get a sense of how it worked and how it would be applied it was necessary to read the documentation.

Web server

In order to have a fully functioning web application, it was important to consider the various possible web servers that could host it. The main three that were considered were Amazon Web Services, Heroku and PythonAnywhere.
Research was required in order to determine which of the three should be implemented. Reading up on the three of them allowed for the comparison of the pros and cons.

Design

High Level Design

As the aim of the project was to create a fully functioning online web application, it make sense that I would consider that when initially designing the system. While it remains similar to the design that was introduced within the Functional Specification, the end product has a few changes which can be seen in the diagrams below.

The concept of the system was revised which renders it slightly different to what was outlined within the Functional Specification. For instance, the very first design within the Functional Specification was focused solely on analyzing offensive behaviour whereas the new revised design has more of a generalized approach which can highlight both positive and negative areas of Reddit as well as overall analysis.

The system is broken down into various different components: the main application which was built using the Flask and Bootstrap frameworks, the SQLite database which the application communicates with, the Reddit API via *praw* and the PythonAnywhere server on which is is hosted.

Initial System Diagram

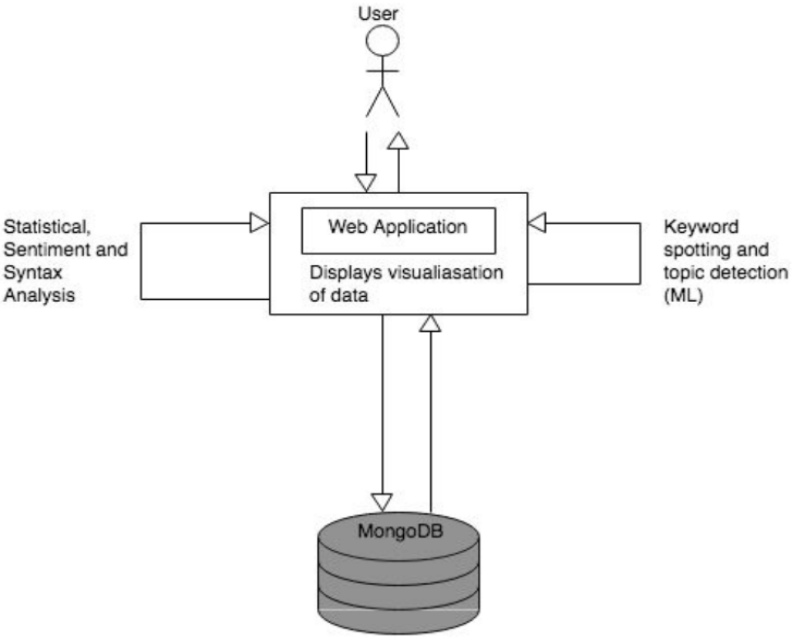

Fig. 1.1 Initial System Diagram

Revised System Architecture Diagram

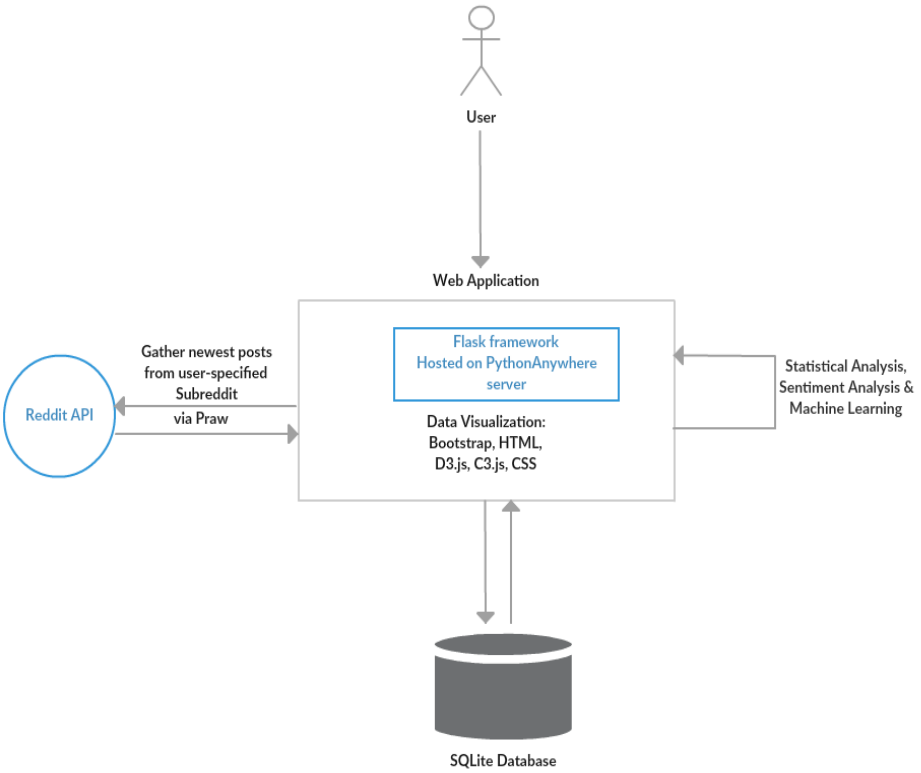

Fig. 1.2 Revised System Diagram

The web application was designed to accept user input in order to query specific data via interacting with the Reddit API. In the previous design, within the Functional Specification, the original design was that there would be no facility for user control. As this design is quite limited, a change was in order. Thus, the concept of allowing the user to query by timestamp was introduced.

This user input was then processed, i.e. converted from user readable format into a UNIX Epoch timestamp and then passed into *Praw*. The web application read from a list of the top 100 subreddits to extract comments from the submissions within. For each comment in each submission within the Subreddit, the comment's body, author and score was extracted and then its sentiment was calculated before being inserted into an SQLite database. It should be noted that this is a new change as in the Functional Specification, it was originally designed to incorporate a MongoDB.

Once the data has been acquired, the system cleans the data and then performs elements of statistical analysis in order to draw results. With these acquired, it then generates word-clouds and charts to display the results in a clearer manner for the user. In the previous design, it outlined the potential of keyword spotting and topic detection as well as sentiment analysis. The new revised design strays away from keyword spotting as the main purpose is no longer to focus solely on offensive behaviour but using a more general approach.

The idea of querying by user inserted timestamp had to be revised once Reddit announced that they were making changes to their site and API which would no longer facilitate this functionality. This is outlined in more detail in the Problems Solved section of this document. This new design added the functionality of enabling the user of querying by Subreddit and being presented with more Subreddit-focused analysis. It then uses this data to train a machine learning model in an attempt for the system to learn about user behaviour patterns such as comment features for a successful public response. It also learns about the features of a comments that would belong in specific subreddits which could further be used in categorization.

The analysis and machine learning aspect results are then inserted into the Flask framework using a combination of HTML/CSS, Javascript and Bootstrap to create an appealing and interactive user input.

Implementation

I made use of a number of OpenSource Python packages for the likes of data preprocessing and for the machine learning aspect, i.e. pandas, numpy, nltk, scikit-learn. For the first instance of development, I made use of the Python package TextBlob for calculating sentiment polarity in order to establish a working prototype on which to further build. I then implemented a sentiment analysis algorithm to replace the package once I had ensured that it would perform as a suitable replacement.

Gathering the data

The first stage of implementation was gathering and acquiring the data. As outlined previously, the initial approach was to make use of predefined dataset of Reddit comments but once cleaned, it was soon discovered that it would not provide a lot of usability. As an alternative, the Python package Praw was introduced as a wrapper for interacting with the Reddit API.

Praw helped retrieve the required data and sentiment analysis was performed upon the data in order to calculate a comments sentiment and also to calculate the overall sentiment of a subreddit based on the total comment sentiment in relation to total number of comments (i.e. total sentiment of comments/ total # of comments).

Web application

Whilst working on cleaning and preprocessing the data so it would be suitable for analysis, I also worked on implementing a mock up UI using Bootstrap so I could get an idea as to how the format of the web application should be structured in terms of routes and page linking.

I established the basic skeleton of the Flask web application and included the completed analysis into the application in order to achieve a basic functional web application which ran on the localhost. Flask is made up of templates and a main file which runs the application, which is `app.py` in this project. The templates get rendered to provide the user with a user interface and also works as a way to pass variables from `app.py` into the HTML.

The Flask web application interacts with both the Reddit API via Praw and also with an SQLite database. It makes use of the SQLite database as a resource to store data on which to work. Within this database, there are three tables that are used by the web application. One of them contains the larger amount of Reddit data that was gathered initially before Reddit made changes to its API. This data is used for the analysis of the overall dataset. The other tables consist of Subreddits with their overall Sentiment and then a table which holds temporary data which comes from the users specified subreddit query. The database is called `reddit_comments.db` can be found in the `src/FlaskApp` directory.

Front-End

To create the User Interface, I incorporated the framework Bootstrap in order to ensure that there was a consistent design throughout the application and to also provide the web application with an aesthetically appealing appearance. Most of the User Interface is developed from the HTML and CSS.

In order to facilitate for dynamic and responsive user interaction, classic JavaScript is used. In particular, Javascript powers the loading screens, the clickable fun-facts and is what is behind the user feedback aspect of the machine learning page. It is also used for the autocomplete form for generating word-clouds for individual subreddits.

The Javascript libraries D3.js and CS.js are also used in creating the visual graphs. C3.js is what powers and produces all the graphical representations and D3.js is what is behind displaying the word-clouds for both the overall dataset and the individual subreddits.

Statistical Analysis

Once the data had been preprocessed and cleaned, statistical analysis could be developed in order to draw conclusions from the data. These statistical analysis algorithms consist of the following: finding the most frequently used words within the data, the top positive and negative subreddits, the most active users and the proportion of self-declared sarcasm.

Finding the most frequently used words within the dataset was required in order to generate the word-clouds and also to provide an overview at a glance as to what subject matter was being discussed the most. It did this by tokenizing and filtering out non-alphabetical strings. It then removes stop-words which are words that are commonly used which do not provide any meaningful representation of topics such as “the”,“and” and “or”. Once stop-words were removed, it finds the Frequency Distributions of the words. The Frequency Distributions essentially provide us with the frequency of each vocabulary item in the text. From there, it finds the 75 most common.

To find the Top 5 Positive and the Top 5 Negative subreddits within the data, it performs sentiment analysis on Reddit comments. It then finds the overall relative sentiment score of each subreddit and can easily determine the two ends of the spectrum of positivity and negativity. More information on how the sentiment analysis algorithm works is outlined in the following section.

The Most Active Users within the dataset refers to the users who have posted more than 10 times amongst the entire corpus. I thought this would be very interesting to derive as it is not out of the ordinary for the most active users within a Subreddit to be Reddit bots. When initially testing this algorithm on a small sample of data, there were evident bots visible such as /u/WritingPromptsRobot and /u/autotldr. I acknowledged this and discussed it in the blog. To find these, the algorithm makes an SQL query call to the database in order to find the relevant comments.

I was very interested in the idea of experimenting to see the complexity and issues behind detecting sarcasm from textual data. As a result, there is a part of the statistical analysis that incorporates the proportion of self-declared sarcasm. As detecting sarcasm is quite a complex task, which is still being subjected to research, I made use of a common characteristic of Reddit comments where the author of the comment incorporates the token ‘\s’ or ‘\sarcasm’ to make it clear that their comment is not to be taken literally. With this knowledge in mind, the web application can query the database of comments for those comments which contain this token. Due to the nature of not detecting all possible cases of sarcasm, I christened this as detecting ‘self-declared’ sarcasm.

Sentiment Analysis

The Sentiment Analysis algorithm works off of a basic sequence of events and incorporates a variation of the Bag of Words approach. It first preprocesses the text by splitting the Reddit comment into a series of sentences and then splits these sentences into tokens. Once tokenized it can then add a Part Of Speech (POS) tag in order to categorise the token. It adds a POS tag by making use of the `nltk` Python library. Categorising the token with a POS tag helps the application to understand the sentence structure and makes it easier to follow the meaning of a sentence based off of the words. When the words have been tagged, they have a structure that is similar to the following: [‘word’, ‘stem’, ‘pos-tag’].

From here, these tokens are tagged based off of whether they are positive, negative, incremental and decremental. This is done by comparing them to a dictionary containing tagged vocabulary and if present, they receive the corresponding tag. This dictionary can be found in the `sent_tags.yml` file.

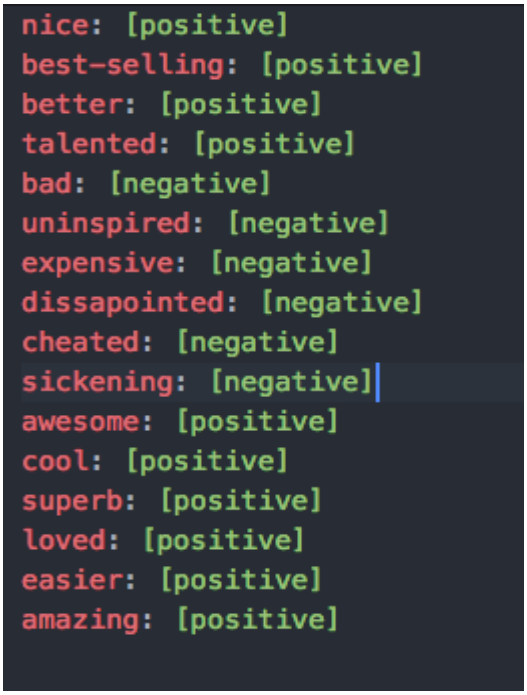

Fig. 1.3 Tagged dictionary words

The new structure becomes [‘word’, ‘stem’, [‘sentiment’, ‘pos-tag’]]. With these tags in place, it is just a matter of calculating the frequency of each sentiment tag to define its sentiment score. If it detects a positive tag, it returns 1 but returns -1 if it detects a negative tag. As this is a quite a naive approach, it also takes into account sentiment tags which can increment or decrement the sentiment value from words such as “very”,“too” or “little”.

These words are tagged with a corresponding increment or decrement tag and if detected will multiply or divide the current sentiment score by 2 which can be seen in the screenshot below.

```
if 'inc' in prev_tag:
    token_score *= 2.0
elif 'dec' in prev_tag:
    token_score /= 2.0
```

Fig. 1.4 Code Snippet for Incrementing/Decrementing Sentiment Score

The algorithm finishes up by calculating the sum of the sentiment of all words within each sentence and the sum of all sentences within a Reddit comment. When the algorithm was initially developed, its output was compared to that of its previous placeholder TextBlob to see if there was any inconsistency between the two. .Results of this have been outlined within the Testing Results section.The aspect of TextBlob has been removed and replaced by this algorithm as mentioned in previous sections

Graphical Representation

The web application contains visual aids in representing data statistics in the form of Bar charts, Pie charts and Donut charts. These charts visualise the following: total number of comments per each subreddit, the distribution of active users per subreddit, the distribution of positive, neutral and negative comments and lastly, which language style is likely to achieve a higher comment score.

Detecting popular language style was a very interesting concept to me. I was curious to see if there was a particular style or type of language seen within comments that resulted in other users being more likely to upvote said comment. To experiment further, I derived 5 categories of deterministic language style such as Aggressive language, Assertive language, Meme/Popular media references, Passive language and added in the concept of self-declared sarcasm which was mentioned in an earlier section of this document.

Furthermore, I created a series of word lists which included language of each style and found the correlation between the average score of a comment and the frequency of words of each style within the comment.

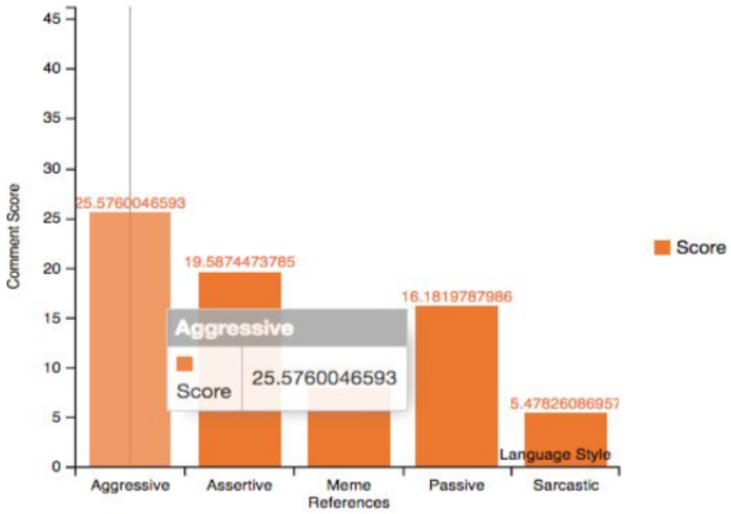

Fig. 1.5. Language Style Graph

Another graph I was interested in depicting was the spread of the distribution of the most active users amongst subreddits that fit into the category of being one of the Top 5 Positive, Negative or Sarcastic. If the user was not present within any of these Top 5 lists, they were added to Other.

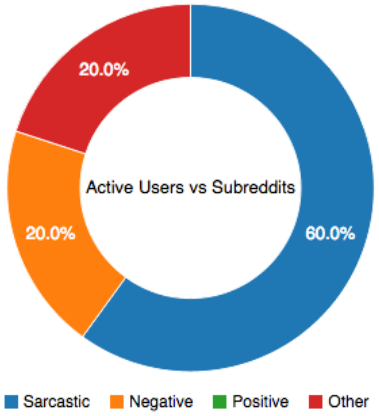

Fig. 1.6 Active Users vs Subreddits Donut Chart

Machine Learning

For the Machine Learning aspect of this project, the main goal was to create two predictive models that can successfully predict the Subreddit of a Reddit comment and the Score of a Reddit comment.

This required preprocessing the text data in order to make it usable as machine learning works only with numerical data. In order to preprocess each Reddit comment, I created a category ID which would represent the Subreddit as a number.

```
# add categories
data_df['category_id'] = data_df['subreddit'].factorize()[0]
category_id_df = data_df[['subreddit', 'category_id']].drop_duplicates().sort_values('category_id')
category_to_id = dict(category_id_df.values)
id_to_category = dict(category_id_df[['category_id','subreddit']].values)
```

Fig. 1.7 Code Snippet outlining the creation of Subreddit category ID's

As well as this, I incorporated the use of a TF-IDF Vectorizer and Transformer as well as a Count Vectorizer which have already been explained within the Research section of this document. These were used to transform the documents to feature vectors in order to extract features to train the model on. Count Vectorizer was used to gather counts of word n-grams/consecutive characters in order to build a dictionary of feature indices.TF-IDF was used in order to minimize the weights for words that occur in many documents which are therefore less informative than those that occur only in a smaller portion.

Once the features had been extracted, the two models were then trained on various Machine Learning algorithms in order to achieve the best accuracy overall. For predicting the Subreddit of a comment, I experimented with Naive Bayes, using a Linear SVM with an SGDClassifier. For predicting the Score of a comment I made use of Linear SVC, Adaboost Classifier and also RandomForestClassifier. These algorithms were selected with the help of scikit-learn's "Choosing the Right Path Estimator". When using these algorithms, it is essential that the data is separated into a training and a testing set.

Predicting the Subreddit

Naive Bayes Classifiers are based on applying Bayes' theorem which takes the "naive" assumption of independence between every pair of features. It was implemented making use of the Pipeline feature from scikit-learn. Naive Bayes is known for being quite fast in comparison to other methods but is also regarded to be not the best of estimators, which explains the poor results it provided when compared against using a Linear SVM model. Specifically the system makes use of Multinomial Bayes which is more suitable for text documents. The model only achieved an accuracy of 56-60%.

```
# Naive Bayes
comment_clf = Pipeline([('vect', CountVectorizer(stop_words='english')),
                        ('tfidf', TfidfTransformer()),
                        ('clf', MultinomialNB()),
                        ])

comment_clf = comment_clf.fit(data_df.comment, data_df.subreddit)
# predict
predicted = comment_clf.predict(test_data_df.comment)
prediction_accuracy = (np.mean(predicted == test_data_df.subreddit) * 100)
```

Fig. 1.8 Code Snippet for creating Naive Bayes pipeline

The second technique I incorporated was that of a Linear SVM classifier with SGD training. This combines regularized linear models with stochastic gradient descent learning. This means that the gradient of loss is estimated per sample and the model is updated along the way with a decreasing learning rate. This implementation worked well with the data at hand because the data was represented as sparse arrays. This model achieved a greater accuracy than that of Naive Bayes, with a score of approximately 79%. The confusion matrices for these two algorithms can be seen within the Testing section of this document.

```
# SGD CLASSIFIER
text_clf_svm = Pipeline([('vect', CountVectorizer(stop_words='english')),
                         ('tfidf', TfidfTransformer()),
                         ('clf-svm', SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3, n_iter=5, random_state=42)),])

                                  # (X_train, Y_train)
text_clf_svm = text_clf_svm.fit(data_df.comment, data_df.subreddit)
```

Fig. 1.9 Code Snippet for creating Linear SVM with SGD Classifier

Predicting the Score

For the model of predicting the score, I made use of three different algorithms to examine which would perform the best. These three were Linear SVC, AdaBoost Classifier and Random Forest Classification. They all produced similar results but the best out of the three were the Linear SVC and AdaBoost Classifier therefore I decided to include them into the web application.

Random Forest Classification works by fitting a number of decision tree classifiers on sub-samples of the data and uses averaging in order to improve the predictive accuracy. I had some exposure to using Random Forest before in my Data Mining and Warehouse module which I was why I was curious to see how it performed with such a different dataset.

Linear SVC seemed like it would be a good choice as it supports both dense and sparse input which I had noticed were evident in the data. Lastly, AdaBoost Classifier works by fitting a classifier on the dataset but then fits additional copies of the classifier on the same dataset where the weights of incorrectly classified instances are adjusted. These were both fitted with a bag of words which were a result of the Count Vectorizer.

```
# predicting score w AdaBoostClassifier
vect = CountVectorizer(min_df=1, stop_words='english')
random_data_sample = data_df.sample(frac=0.1, random_state=87824, axis=0)
bow = vect.fit_transform(random_data_sample['comment'])
ab = AdaBoostClassifier(n_estimators=200)
ab.fit(bow, random_data_sample['score'])

test_data = data_df.sample(frac = 0.1, random_state=824, axis=0)
ab_prediction = ab.predict(vect.transform(test_data['comment']))
ab_pred_acc = np.mean((ab_prediction == test_data.score) * 100)

# Linear SVC
svc = LinearSVC()
svc.fit(bow, random_data_sample['score'])
svc_test_data = data_df.sample(frac=0.1, random_state=424, axis=0)
svc_pred = svc.predict(vect.transform(svc_test_data['comment']))
svc_pred_acc = np.mean((svc_pred == svc_test_data.score) * 100)
```

Fig. 1.10 Code Snippet for predicting the score with AdaBoostClassfier and LinearSVC

The system had some trouble predicting the score of the comment due to a couple of reasons and held an accuracy of between 20-30%. Due to the ambiguity of language, it was unable to find any correlation between the words that were present and the comment score. In an attempt to improve the accuracy, I removed stop-words and set a threshold comment score of 5 which meant it would only work on predicting scores less than 5. Doing so increased the accuracy to 35-40%.

Based off of these models, the web application also implemented a way for the user to interact with the Machine Learning aspect. There is a facility for the user to input a desired sentence and the system will take this input and apply the Linear SVM (with SGD Classifier) model to this new input data in an attempt to predict what subreddit it would ideally belong to. In the future, this could possibly be used for auto-categorization of comments that may be unrelated to the subreddit that they are posted in.

User Specified Subreddit Analysis

The web application incorporates the ability for a user to retrieve and view analysis performed upon the newest comments in a subreddit that they have inputted.
This is present on the homepage of the web application and provides a clearer insight into comments based on a specific topic from the Subreddit.

This part of the web application provides the user with an overall sentiment score for the Subreddit, the number of newest comments that have been added that day, the most popular language style and also provides the proportion of user contribution to the comment set. This can be used to investigate as to whether or not a certain user has hijacked all the threads within the newest posts within the Subreddit.

```
top_posters = pd.DataFrame(posters)
top_posters.columns = ['user', 'commentCount']
#print top_posters
top_commenters = top_posters
top_commenters = list(top_commenters.values.flatten())
poster_freq = dict([(k, v) for k,v in zip (top_commenters[::2], top_commenters[1::2])])
#print length
user_activity = {}
for key, value in poster_freq.iteritems():
    f_length = float(length)
    f_val = float(value)
    proportion_of_comments = f_val/f_length
    proportion = 100 * proportion_of_comments
    proportion = float(("%0.4f"%proportion_of_comments))
    user_activity.update({key : proportion})
return user_activity
```

Fig. 1.11 Code Snippet for finding the User contribution

Web Server

The web application is hosted on a PythonAnywhere server and can be accessed at www.analyzingredditbehaviour.com. There was the possibility of hosting it on Heroku, but that did not provide the necessary support for hosting an SQLite database.

Problems Solved

As part of the development process, there were a number of problems encountered which required. These problems vary from some small issues to quite substantial ones. The sections below will outline these further.

Data

Initially, the plan was to work off of a large dataset of publicly available Reddit comments that another user had scraped and uploaded to everyone to use. However, since this dataset contained all of the publicly available comments on Reddit it was quite large in size. Due to its size, it was only usable in small chunks such as comments per month. After cleaning this data, it was discovered that it had some hidden problems of its own and once cleaned there wasn't enough data to work with to draw meaningful results so it was rendered to be

unusable. To counteract this, I decided that I would scrape the Reddit comments using the Reddits API Python Wrapper, Praw. With Praw, you are able to query by specific timestamps so I figured it would be an excellent way to encourage user engagement and use their desired timeframe to extract the data they were interested in.

Word2vec

Towards the beginning of deciding which various techniques of statistical analysis and natural language processing should be implemented, I came across the concept of implementing word embedding through the use of a word2vec model. These models are shallow neural networks that are trained to reconstruct linguistic contexts of words. I thought it would be interesting to have the ability to derive word associations from Reddit data but unfortunately, trying to implement it as part of the overall project was unsuccessful. The associations that it produced weren't very meaningful and due to the time constraints of the project and trying to implement all of the other parts, it meant that I couldn't afford to dedicate more time to fine-tune the model. This will be discussed more in the Future Improvements section.

Machine Learning

An aspect of this project involves using machine learning to make predictions on the score and the subreddit of a comment. This involves some experimentation in regards to finding the most ideal machine learning algorithm that was best suited to the data at hand.

A problem was encountered here when trying to predict the score of a comment. Due to the lack of context behind the comments, the machine learning algorithms had some trouble predicting the score of a comment and resulted in poor results of accuracy at about 25%. In order to try to mitigate this issue, I decided to experiment with removing stop-words that were present during preprocessing and also assigning category ID's to Subreddits in an attempt to provide more numerical data for the system to train off of. Whilst this did improve the accuracy and raise it to 35%, it was still a poor performance. It was then that I decided to see if limiting the range of comment scores would add some improvements. I altered the range of score to between 0-5, and in doing so increased the accuracy to approximately 39-40%.

While this is still quite a low score, I am able to understand the reason behind it. There is possibly an element of humour or context that a user takes into account when the decide to upvote a comment which is quite hard to detect through text classification.

Reddit API change

Quite a substantial problem to overcome was that of the change to the Reddit API. As mentioned in previous sections, the initial functionality depended on the user inputting their desired start and end time to query the data between.

Reddit updated their website in order to update their current Search algorithm and unfortunately, this included making changes to their API's functionality. In doing so, this

removed the ability to query data by timestamp which meant that the web application could no longer fetch new data corresponding to the users desired timeframe.

In an attempt to overcome this, I decided that an appropriate workaround would be to allow the user to input a particular Subreddit they were interested in focusing solely on. When searching for their subreddit, the web application fetches the newest posts of the day in said subreddit and performs the analysis on that set of data.

In a way, while this is not what I set out to achieve, it seems to be a better alternative in the long run. For instance, now it is a lot clearer to track what Reddit users think of a certain topic. If a new episode of a television show comes out on a weekly basis, you can clearly see the Reddit communities reaction to and whether it be a positive or negative one at that.

Data display delay

Due to the capacity of the data and the various analysis algorithms that are performed upon it, the web application did suffer in terms of delaying when processing and presenting the results to the user. This was not ideal, because if the data was quite substantial there would be a significant delay which would cause the user to become disinterested.

To overcome this issue, I made use of Flask's caching ability. This allows for the user to seamlessly move in between the menu tabs, taking in the results of the data. However, this does result in a slight delay when first interacting with the web application.

Results

Testing Results

This section outlines and records the tests that have been carried out on the system in terms of Unit Tests, Functionality Tests and various Acceptance Level Tests. In parallel with the UI and User testing, this aims to provide a detailed account of how the system fared against various circumstances. This section is divided by the different testing procedures.

Back-End Testing

Unit Tests were incorporated to ensure that each component worked as was expected and there would be no exceptions thrown or errors.

Unit Tests

Sentiment Analysis Algorithm

File: `sentiment_tests.py`
Tested: `getSentiment(), splitString(), tagValue(), makeTag(), sentiment_score()`
Result: PASS

Fig. 2.1 Terminal output for Sentiment Analysis unit tests

File: `getSentiment(), TextBlob.sentiment.polarity()`
Tested: `Comparing getSentiment() (own algorithm) against TextBlob.sentiment.polarity() (package sentiment analysis algorithm)`
Info: Values > 1 are positive; Values >= 0 && Values <= 1 are neutral; Values < 0 are negative
Results: see table below

Sentence	Own	TextBlob
"I hate that movie, it's so bad"	-2.0	-0.75
"wow, dude I think you should calm down"	1.0	0.0814814815
"words can't express how much I love that!"	0.0	0.625
"to be honest, I don't really care too much about it"	0.0	0.333333333
"Mods must be asleep. They would've taken my shit down quick with some bs. "	-2.0	-0.00740740740741
"there is a lack of integrity"	0.0	0.0
"I love dogs so much, they are so cute"	1.0	0.4
",.[)@]+"	0.0	0.0
"6789998212"	0.0	0.0
“ “	0	0.0

```
[MacHome1:FlaskApp blackstar138$ python sentiment_test_textblob.py
I hate that movie, it's so bad
Own Sentiment:  -2.0
TextBlob's Sentiment:  -0.75

wow, dude I think you should calm down
Own Sentiment:  1.0
TextBlob's Sentiment:  0.0814814814815

words can't express how much I love that!
Own Sentiment:  0.0
TextBlob's Sentiment:  0.625

to be honest, I don't really care too much about it
Own Sentiment:  0.0
TextBlob's Sentiment:  0.333333333333

Mods must be asleep. They would've taken my shit down quick with some bs.
Own Sentiment:  -2.0
TextBlob's Sentiment:  -0.00740740740741

there is a lack of integrity
Own Sentiment:  0.0
TextBlob's Sentiment:  0.0

I love dogs so much, they are so cute
Own Sentiment:  1.0
TextBlob's Sentiment:  0.4

,.[)@]+
Own Sentiment:  0.0
TextBlob's Sentiment:  0.0

6789998212
Own Sentiment:  0.0
TextBlob's Sentiment:  0.0


Own Sentiment:  0
TextBlob's Sentiment:  0.0
```

Fig. 2.2 Terminal output for Sentiment Analysis unit tests

Unit Tests

Statistical Analysis (Word Clouds)

File: `wordcloud_testing.py`
Tested: `test_wordcloudLength(self), test_wordcloudType(self), test_subFreqWords(self)`
Results: see table below

Test ID	Output	Result
def test_wordcloudLength(self)	Length = 50	PASS
def test_wordcloudType(self)	is_dataframe = true	PASS
def test_subFreqWords(self)	is_list = true	PASS

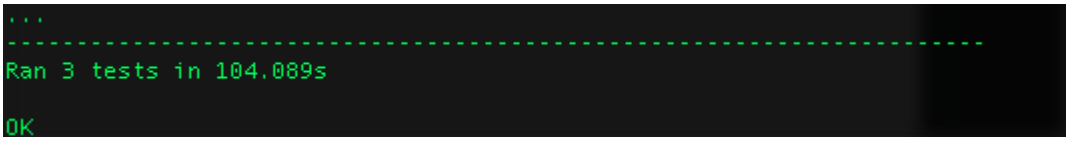

Fig. 2.3 Terminal output for Wordcloud unit tests

Unit Tests

Analysis algorithms

File: `analysis_testing.py`
Tested: `test_detectSaracsm(self), test_freqPosters(self), test_freqWords(self), test_top5neg(self), test_top5pos(self), test_createSubSeries(self), test_activeSubs(self)`
Results: see table below

Test ID	Input	Output	Result
def test_detectSaracsm(self)	sarcasm_proportion	0.0236	PASS
def test_freqPosters(self)	N/A	N/A	PASS
def test_freqWords(self)	length	10	PASS
def test_freqWords(self)	is_stopword	False	PASS
def test_top5neg(self)	N/A	N/A	PASS
def test_top5pos(self)	N/A	N/A	PASS
def test_createSubSeries(self)	is_dataframe	True	PASS
def test_activeSubs(self)	is_float	True	PASS
def test_activeSubs(self)	sarc_percentage	60.0	PASS
def test_activeSubs(self)	negative_percentage	20.0	PASS
def test_activeSubs(self)	positive_percentage	0.0	PASS
def test_activeSubs(self)	Other	20	PASS

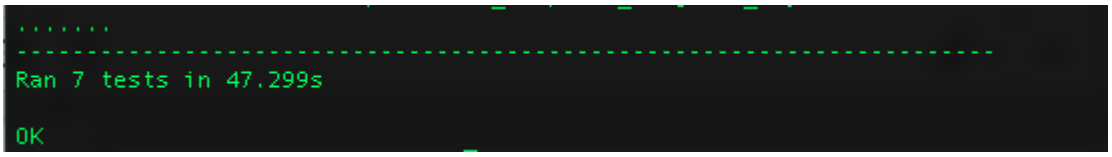

Fig. 2.4 Terminal output for Analysis unit tests

Unit Tests
Machine learning

File: `machine_learning_testing.py`
Tested: `test_predictSentence(self), test_predictedSubreddit(self), test_predictedScore(self)`
Results: see table below

Test ID	Input	Output	Result
test_predictSentence(self)	**“My favourite sport is basketball”**	**[u’nba’]**	**PASS**
test_predictSentence(self)	**“I really like to travel”**	**[u’travel’]**	**PASS**
test_predictedSubreddit(self) (Naive Bayes)	**Dataframe[‘comment’, ‘subreddit’, ‘score’, ‘sentiment’, ‘created_utc’]**	**Output > 30**	**PASS**
test_predictedSubreddit(self) (Linear SVM)	**Dataframe[‘comment’, ‘subreddit’, ‘score’, ‘sentiment’, ‘created_utc’]**	**Output > 65**	**PASS**
test_predictedScore(self) (Adabooster)	**Dataframe ['comment', 'subreddit', 'score', 'sentiment', 'created_utc']**	**Output > 30**	**PASS**
test_predictedScore(self) (Linear SVC)	**Dataframe ['comment', 'subreddit', 'score', 'sentiment', 'created_utc']**	**Output > 65**	**PASS**

Fig. 2.5 Terminal output for Machine Learning unit tests

Unit Tests for Querying by Subreddit

File: `test_querybysubs.py`
Tested: `test_frequentWordsinSub(self), test_overallSentiment(self), test_activePostersinSub(self), test_language_style_subreddit(self)`
Result: see table below

Test ID	Output	Result
test_frequentWordsinSub(self)	is_dataframe = true	**PASS**
test_overallSentiment(self)	is_float = true	**PASS**
test_activePostersinSub(self)	is_dictionary = true	**PASS**
test_language_style_subreddit(self)	is_Series = True	**PASS**
test_language_style_subreddit(self)	Len = 5	**PASS**
test_language_style_subreddit(self)	Len = 3	**PASS**

Accuracy Score

Confusion Matrix

Algorithm: **Linear SVM** for predicting Subreddits from comment

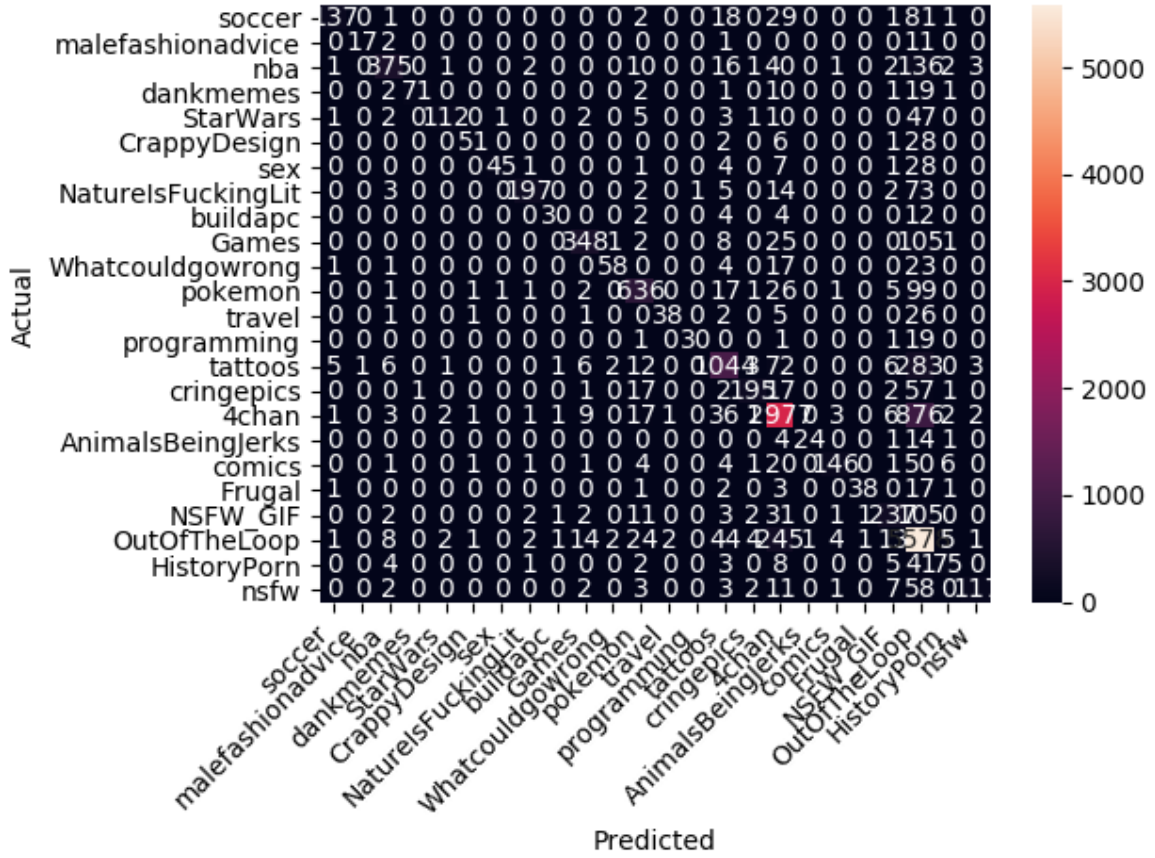

Fig. 2.6 Linear SVM Confusion Matrix

Accuracy Score

Confusion Matrix

Algorithm: **Naive Bayes** for predicting Subreddit of a comment

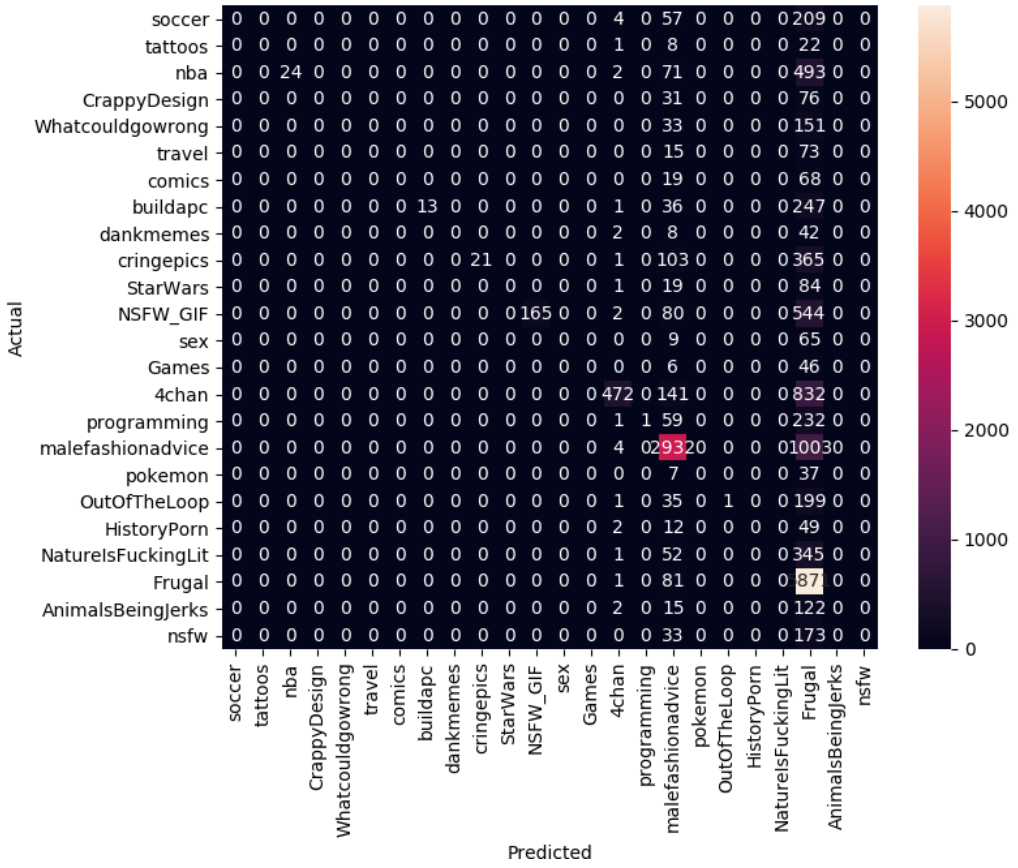

Fig. 2.7 Naive Bayes Confusion Matrix

Unit Tests

Database

File: `database_testing.py`
Tested: `test_sqlite_successfulConnect(self), test_sqlite_failConnect(self), test_connDB(self)`
Info: Unit Tests were implemented to ensure that the Database connected successfully.
Result: PASS

```
'''
----------------------------------------------------------------------
Ran 3 tests in 0.003s

OK
```

Fig. 2.7 Terminal Output for database tests

Functionality Tests

I conducted Functionality Tests to ensure that there were no dead links or routes within the Flask application and that each route returned the correct HTTP status code. I also tested that the GET and POST responses returned the correct HTTP Status code.

Flask Routes Functionality

File: `functional_tests.py`
Tested: `test_server_is_running(self), test_dashboard_response(self), test_charts_response(self), test_wordcloud_response(self), test_sub_wordcloud_response(self), test_wordcloud_sub_response(self):, test_ml_sentence_response(self), test_machine_learning_response(self), test_sub_analysis_response(self)`

Info: Testing to ensure that any and all HTML status codes that could occur are acknowledged per page/route and to check that only a HTML status code 200 (successful request) is returned for each page
Result: see table(s) below

Test ID	**Output**	**Result**
def test_server_is_running(self)	200	PASS
def test_charts_response(self)	200	PASS
def test_wordcloud_response(self)	200	PASS
def test_sub_wordcloud_response(self)	200	PASS
def test_sub_wordcloud_response(self)	200	PASS
def test_wordcloud_sub_response(self)	200	PASS
def test_ml_sentence_response(self)	200	PASS
def test_machine_learning_response(self)	200	PASS
def test_sub_analysis_response(self)	200	PASS
def test_sub_analysis_response(self) (GET)	405	PASS
def test_sub_analysis_response(self) (POST)	200	PASS

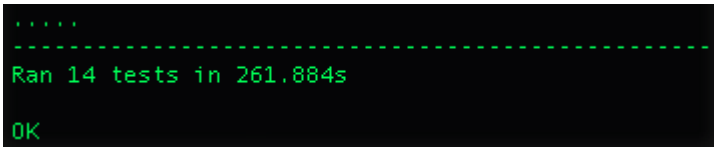

Fig. 2.8 Terminal output for functional tests

User Evaluation Results

Cognitive Walkthrough

A Cognitive Walkthrough was carried out as part of the User Evaluation testing. Cognitive Walkthroughs are used in order to understand the system's learnability for new users. They are conducted by one of more evaluators who work through a series of tasks from the perspective of the user. The results can be seen in the associated cognitive walkthrough file.

Heuristics

In order to fully evaluate my finished web application, I decided to compare it extensively against three separate concepts of design heuristics created by Donald Norman, Jakob Nielsen and Ben Shneiderman respectively.

Norman's 6 Principles of Design

Visibility

The application's user interface has strong levels of visibility throughout. There are clear instructions on the overall usage of the website on the homepage. Instructions as to how to use the homepage are written clearly in a large, clean font and the start function, "Let's Go", is displayed in the form of a large, eye-catching green button.

A navigation menu can be clearly seen to the left of the screen at all times, ensuring that the user can move between sections of the application with ease and has succinct menu tab descriptions which helps to prevent confusion.

The graphs are distinctly labeled and coloured appropriately. The 'Subreddit Word Cloud' page has a submit button to the immediate right of the search bar. The 'Machine Learning' page also has clear instructions written on the page to benefit users.

Feedback

System feedback can be seen and implement as the user proceeds throughout the application. A feedback loading symbol appears when the user initially clicks to view the pages containing graphs, word cloud or machine learning algorithm accuracies to inform the user that there is a short wait before the data will be displayed. When inputting a string to search for a specific subreddit, prompts of available subreddits for analysis appear for the user to select.

Constraints

There are a number of constraints which are visible to the application user. They consist of the following:

1. The user must wait for charts and word clouds to fully load before viewing them
2. All user input must be in English
3. When searching for word clouds per subreddit, the user must enter a string that matches at least one valid subreddit that appears in the prompted menu list.
4. The user must enter a valid non-numeric string in order to predict which subreddit it belongs to
5. The application is not offline so the user requires a stable internet connection

Mapping

The system has been designed responsively and can be viewed on a wide ranges of different screen sizes/devices as well as being able to respond appropriately to the user minimizing or maximizing the screen on a whim.

Consistency

The application's consistency can be seen in its responsive design and stylistic formatting. The system style format does not change and remains static as the user navigates through the page links. Due to the fact that the system has a responsive web design at its core, the layout will remain uniform no matter what device it is displayed on.

Affordance

Affordance can be seen in the instructions displayed on various sections of the application such as but not limited to: the index page; subreddit prediction page and the word cloud per subreddit page.

Nielsen's 10 User Interface Design Usability Heuristics

Visibility of the System Status

The system ensures to provides its users with information about its current status via a form of visual feedback. This feedback primarily takes the form of a loading icon and/or a string message, "Please be patient". It disappears once the backend task has been completed and the information presented to the user.

Breadcrumbs are also provided to ensure that the user doesn't lose their place or become confused whilst using the application - e.g. through the use of page headings.

Match between the System & the Real World

The application uses the same language as most of its user base which is, in this case, colloquial English. This ensures that its target users (English-speaking ones) can fully understand the system and any sort of task that is asked from them.

User Control & User Freedom

The application is designed to ensure that the user is in full control at all times. Should they accidentally click to view a section of the website that was not what they wanted, they can

easily select their correct page from the side menu bar. This menu is present at all times and is easily accessible for users.

The users can also refresh their browsers to get a clean page and resume their previous activities. On the ‘Subreddit Word Cloud’ and ‘Machine Learning’ pages, the user is able to undo any accidental errors made while entering a subreddit title or a sentence into the respective search bars by pressing the backspace button on their keyboard.

Consistency & Standards

There is a high level of consistency throughout the overall system. When each individual page is refreshed or if a new session begins, the styling and layout remain the same and do not suddenly change or alter. The application also looks the same on different devices or on different screen sizes.

The submittal buttons, “Submit”, “Let’s Go!”, etc., are a bright green colour and indicate to the user that they must be clicked in order to see a result.

Error Prevention

A high level of error prevention can be seen throughout the application. Error handling is apparent when the user’s actions result in a HTTP error (404, 403 & 500 respectively). The user can return to the home page from the error message page.

Form validation can also be seen on the homepage when the user tries to enter an invalid subreddit title into the search bar. An alert is thrown if the subreddit isn’t valid.

Recognition Rather than Recall

When a user wishes to view a word-cloud for a particular subreddit, they are given a set of precise instructions detailing which subreddit(s) can be searched for. Once they enter the first few letters of their chosen subreddit, they can see a finished prompt in the search bar. This allows them to simply select the subreddit once they recognise the title in the list provided.

The application provides a set of instructions for each individual section that requires a user action, (the homepage, Word Cloud per Subreddit & Machine Learning). These instructions remain in the same position at all times as the user navigates their way back and forth through the system. As such, they will be able to look for them instinctively as they become more familiar with the application

Flexibility & Efficiency

The system is extremely flexible as it is a cross-browser web application and can be accessed on a wide range of devices such as smartphone devices, tablets, laptops, etc..

Aesthetic & Minimalist Design

The application’s design is simplistic and does not bombard or overload the user with too much information. Any textual information such as instructions are short and to the point and

plenty of visual cues such as charts and simple tables are provided to ensure a clean visual layout.

Recognise, Diagnose & Recover from Errors
Users can recover from and recognise their mistakes with ease. They can return to the correct page or to the homepage via the sidebar navigation and they can also return to the homepage should their actions result in a HTTP error.

If they enter an incorrect subreddit, an alert box will pop up onscreen to indicate that a mistake has been made and provides instructions on how to rectify this mistake.

Help & Documentation
Documentation is provided in the form of a technical manual and a user manual as well as instructions on each section of the application.

Shneiderman's 8 Golden Rules

Strive for System Consistency
The stylistic format of the application remains constant and does not change at any point. This means that the chosen colour format, font and image position do not change and ensure a strong level of consistency.

Seeking Universal Usability
The application can be used by both beginner and experienced users and is designed so that it can be easily understood by a wide range of people and viewed on a wide range of devices.

Offer Informative Feedback to the User
Informative feedback is provided to the user by means of loading icons and visual displays of the data such as charts, tables and brief sentences explaining the graph context.

Designing Dialogs to Yield Closure
The application is littered with closure dialogs which provide the user with a sense of accomplishment. Whenever they press 'Submit' on a certain page, a loading symbol (feedback) appears and disappears once the data is displayed.

The website itself has a small number of pages (5 in total) to ensure that the user does not become worried that they have to trawl through an infinite number of pages to see specific data.

Preventing Errors
Errors can be prevented thanks to the implemented form validation as mentioned above in Nielsen's 10 User Interface Design Usability Heuristics.

Permit an Easy Reversal of User Actions

Users can undo any mistake with relative ease as they can re-enter strings, move back and forth through the application at any given point and refresh/reload the application to start with a clean slate.

Keeping Users in Control

Users are in control of the application as soon as they begin their session to the moment that they close it. They can enter their own specific subreddits to view a corresponding word cloud and a specific string of their own choosing to see if it matches the correct subreddit.

They can also select a ‘YES’ or ‘NO’ option to tell the system if the predicted subreddit matches their string. E.g. If ‘Luke, I am your father’ matches a prediction of the ‘Star Wars’ subreddit.

Reducing Short Term Memory Load

The search bars that require the user to enter a string are identical to prevent confusion. Users can also select a specific subreddit from a prompted list once they have enter at least one character that matches the list of provided subreddits as seen in the Word Cloud per Subreddit page instructions.

Human User Evaluation - *Google Form Feedback*

A survey was sent out to a group of users of different demographics who opted in to take part. The survey was in regards to the usability of the web application from their point of view. The questions can be seen in the screenshots below, as well as the responses. The feedback from the form was taken into consideration and changes were made to the User Interface in order to make it more acceptable.

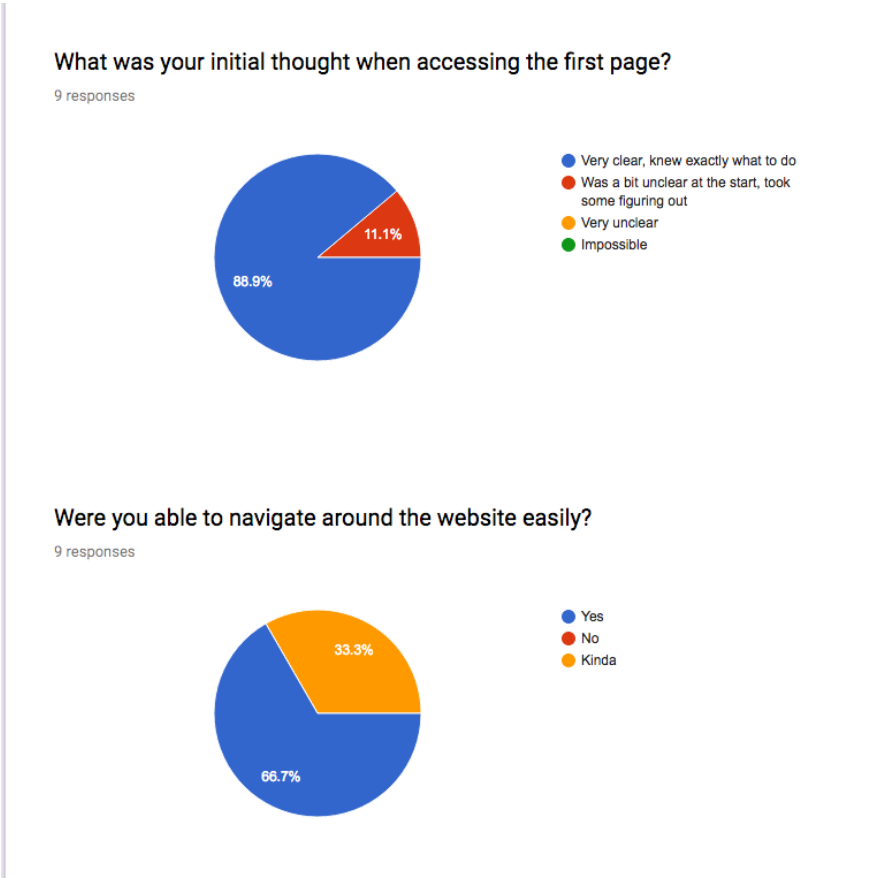

Fig. 3.1. Question 1 and 2 of User Feedback Form

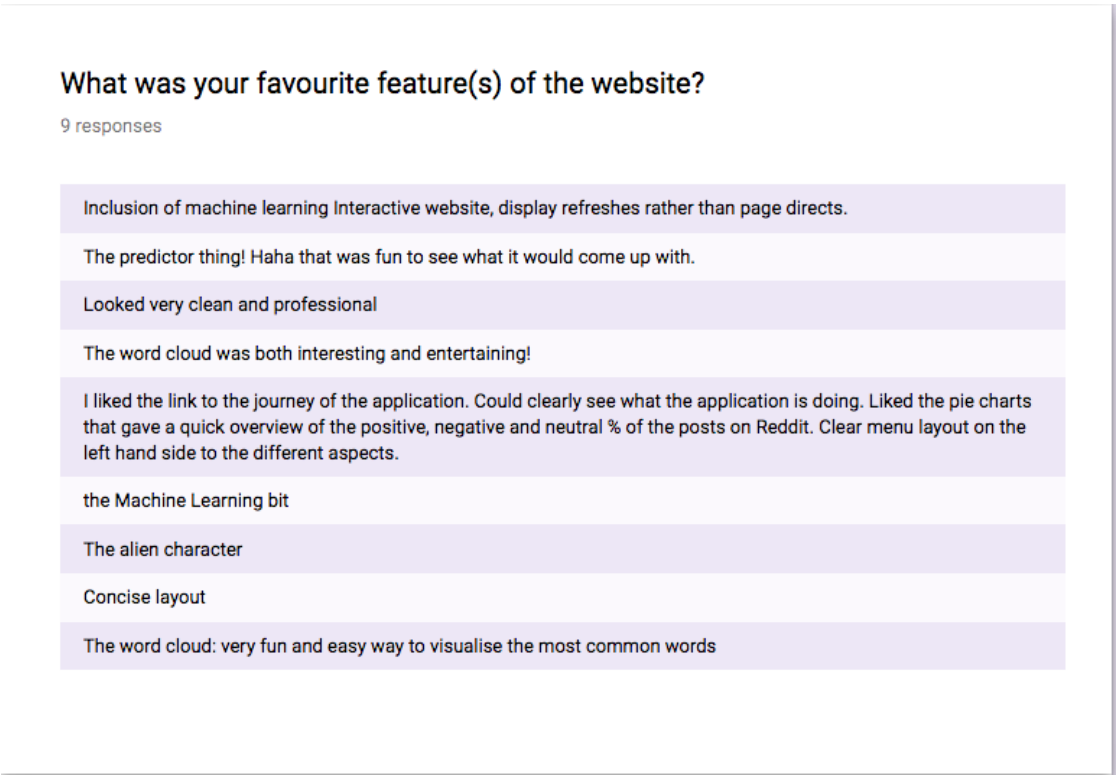

Fig. 3.2. Question 3 of User Feedback Form

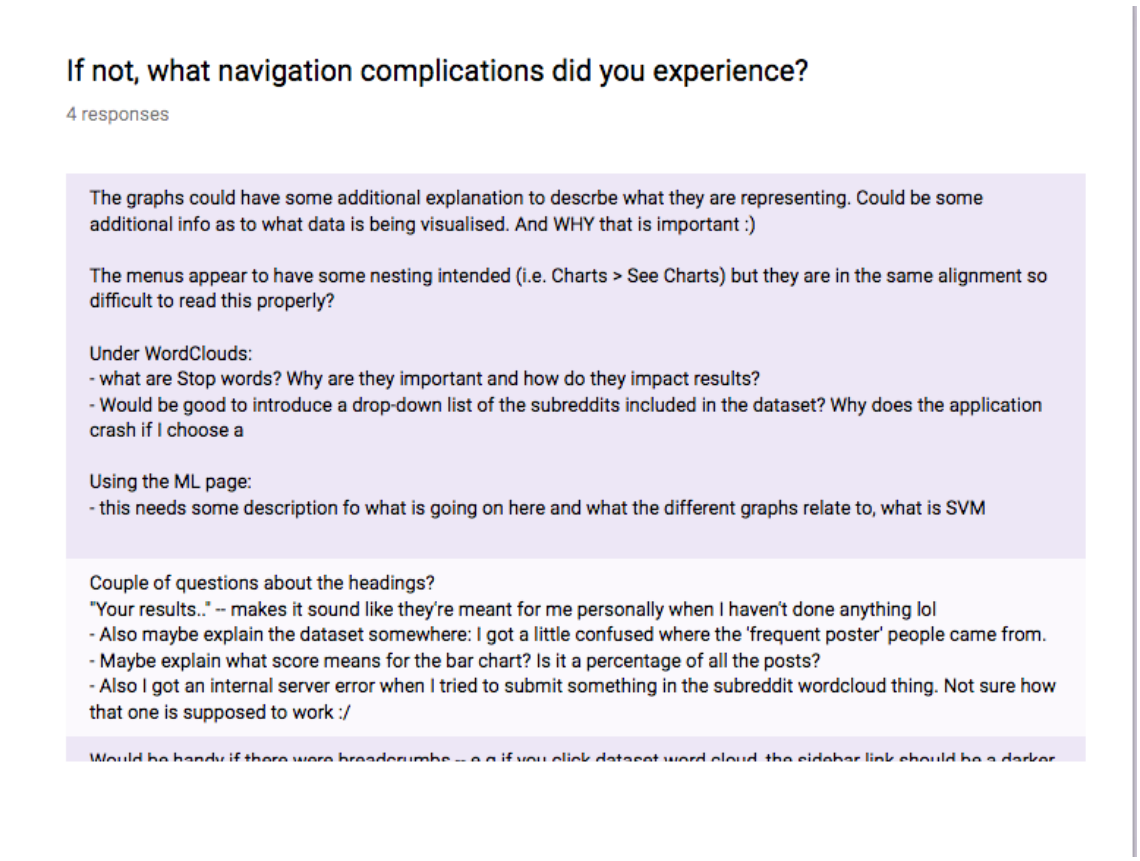

Fig. 3.3. Question 4 of User Feedback Form

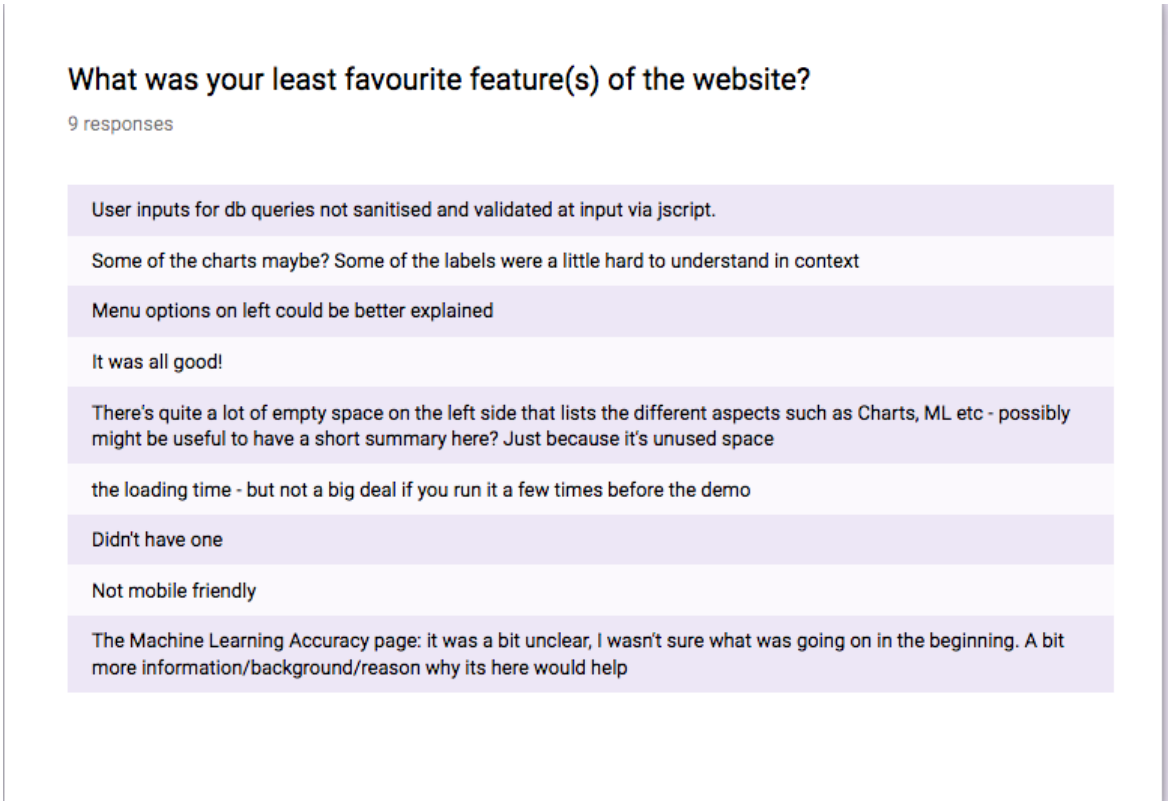

Fig. 3.4. Question 5 User Feedback Form

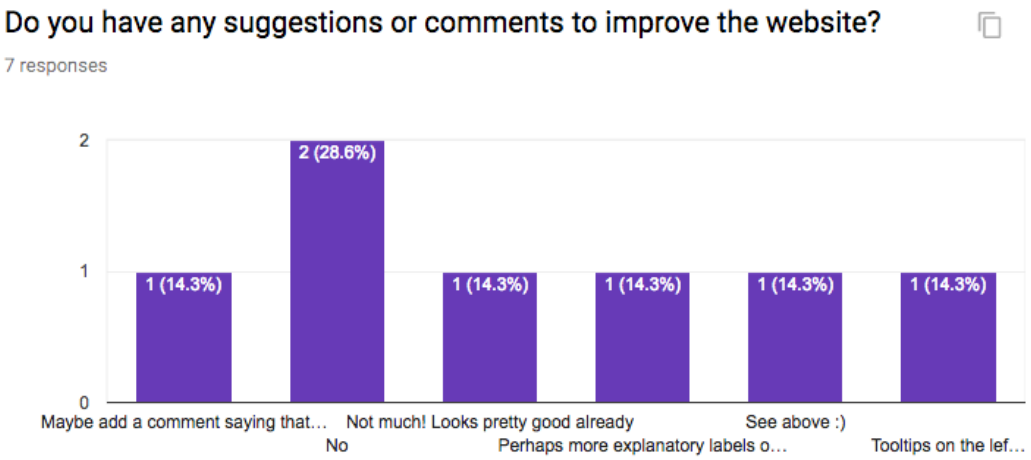

Fig. 3.5. Question 6 of User Feedback Form

Future Work

For future work and improvements, there are a couple of areas that could be be improved upon in the future.

Real-time data fetching

In order to avoid the delay in fetching, processing and presenting the data at all costs, implementing concurrency and real-time data fetching and processing would be a ideal option. Due to the unpredictable nature of Reddit, I deemed that having a web application that would gather and process Reddit data in real-time could end up being inappropriate for a university setting. By this I am referring to how Reddit, at times, can contain content that can be deemed as highly offensive. However, for a non-educational setting running the processes within the web application concurrently would be quite an improvement.

Machine Learning Feedback

As part of the machine learning aspect of this project, the user is able to give the system feedback as to how well they predicted the users text input. As it stands, the users' input does not currently impact the training of the data but instead updates the corresponding gauge chart as to benefit only the user.

An excellent improvement to this system would be to incorporate a way of feeding back in the users feedback back and use it to retrain the model so that the model is continuously learning and improving. Due to the time constraints of this project, implementing this was not possible at this time.

Incorporating an alternative database

This project currently uses an SQLite database, as outlined in previous sections. This type of database is perfectly suited for this type of application as a university project. However, were it to be improved and developed further it would be important to consider replacing the use of an sqlite database with perhaps one that is more powerful such as MongoDB.
For instance, to work with larger datasets and to implement more concurrency SQLite would no longer be the most appropriate choice for the project.

Host on Amazon Web Services (AWS)

The scope of this project did not require the additional features and support that Amazon Web Services provides so hosting it on PythonAnywhere was appropriate. However in future development, hosting the web application on AWS would be a necessity due to how scalable and durable AWS is.

References

Machine Learning

- http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
- http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
- https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a
- http://fastml.com/classifying-text-with-bag-of-words-a-tutorial/
- http://www.ritchieng.com/machine-learning-multinomial-naive-bayes-vectorization/
- http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
- http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
- http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
- http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
- http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Sentiment Analysis

- http://textminingonline.com/dive-into-nltk-part-iii-part-of-speech-tagging-and-pos-tagger
- http://fjavieralba.com/basic-sentiment-analysis-with-python.html
- https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html
- https://pythonspot.com/python-sentiment-analysis/
- https://stevenloria.com/simple-text-classification/

Flask Framework

- http://flask.pocoo.org/docs/1.0/quickstart/
- http://flask.pocoo.org/docs/0.12/testing/
- http://flask.pocoo.org/docs/0.12/errorhandling/#error-handlers

User Interface

- https://www.w3schools.com/howto/howto_js_autocomplete.asp
- https://getbootstrap.com/docs/3.3/getting-started/
- https://fontawesome.com/

Heuristics

- https://www.csun.edu/science/courses/671/bibliography/preece.html
- https://www.nngroup.com/articles/ten-usability-heuristics/
- https://www.interaction-design.org/literature/article/shneiderman-s-eight-golden-rules-will-help-you-design-better-interfaces

Testing

- https://www.usability.gov/what-and-why/usability-evaluation.html
- https://damyanon.net/post/flask-series-testing/
- https://jeffknupp.com/blog/2013/12/09/improve-your-python-understanding-unit-testing/
- http://www.patricksoftwareblog.com/unit-testing-a-flask-application/
- http://www.voidspace.org.uk/python/mock/