**Trump Tweets vs The Markets**

Final Report for CS39440 Major Project

*Author*: Mateusz Stankiewicz ([mas15@aber.ac.uk](mailto:mas15@aber.ac.uk))

*Supervisor*: Dr. Neil Mac Parthaláin (ncm@aber.ac.uk)

26th April 2018

Version 1.0 (Release)

This report is submitted as partial fulfilment of a BSc degree in  
Computer Science (G401)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

**Declaration of originality**

I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the regulations on Unacceptable Academic Practice from the University’s Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
* In submitting this work I understand and agree to abide by the University’s regulations governing these issues.

Name …………………………………………

Date ……………………………………………

**Consent to share this work**

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name …………………………………………

Date ……………………………………………

**Acknowledgements**

I am grateful to…

I’d like to thank StackOverflow community for my Degree.

**Abstract**

no more than 300 words.

“World events often have a great influence over international markets. Political uncertainty can often drive commodities up or down in value depending on where it occurs in the world. “[1]. Politicians of countries with the biggest markets have got a strong impact on the value of currencies and trading commodities. A simple message such as "With Mexico being one of the highest crime Nations in the world, we must have THE WALL. Mexico will pay for it…"[2] sent from the USA president's account can drop down Mexican Peso value. Trump’s infamous Twitter account, which is followed by almost 50 million people, can be an effective tool to influence the markets.

The goal of this project is to develop a system which considers the sentiment of tweets and can predict whether a stock index will increase or decrease depending on the current index, words, phrases and the sentiment of the tweet.

Python, Scikit-learn and NLTK (Natural Language Toolkit) are used to process the data in this project. The web interface is created using Flask framework.

The results show that when using a Naïve Bayes classifier, the accuracy of predicting the USD Index change is 53.6%, whereas the base rate of the three-class problem (up, down, no change) is 41%.

**Contents**

1. Background, Analysis & Process 3

1.1. Background 3

1.2. Analysis 3

1.3. Process 3

2. Design 3

2.1. Overall Architecture 3

2.2. Detailed Design 3

2.2.1. Even More Detail 3

2.3. User Interface Design 3

2.4. Other Relevant Sections 3

3. Implementation 3

4. Testing 3

4.1. Overall Approach to Testing 3

4.2. Automated Testing 3

4.2.1. Unit Tests 3

4.2.2. User Interface Testing 3

4.2.3. Stress Testing 3

4.2.4. Other Types of Testing 3

4.3. Integration Testing 3

4.4. User Testing 3

5. Critical Evaluation 3

6. Appendices 3

A. Third-Party Code and Libraries 3

B. Ethics Submission 3

C. Code Samples 3

Annotated Bibliography 3

# Background, Analysis & Process

This section should discuss your preparation for the project, including background reading, your analysis of the problem and the process or method you have followed to help structure your work. It is likely that you will reuse part of your outline project specification, but at this point in the project you should have more to talk about.

**Notes:**

* All of the sections and text in this example are for illustration purposes. The main Chapters are a good starting point, but the content and actual sections that you include are likely to be different.
* Look at the document on the MMP: Final Report and Technical Work [7] for additional guidance.

## Background

What was your background preparation for the project? What similar systems did you assess? What was your motivation and interest in this project?

## Analysis

Taking into account the problem and what you learned from the background work, what was your analysis of the problem? How did your analysis help to decompose the problem into the main tasks that you would undertake? Were there alternative approaches? Why did you choose one approach compared to the alternatives?

There should be a clear statement of the objectives of the work, which you will evaluate at the end of the work.

In most cases, the agreed objectives or requirements will be the result of a compromise between what would ideally have been produced and what was felt to be possible in the time available. A discussion of the process of arriving at the final list is usually appropriate.

As mentioned in the lectures, think about possible security issues for the project topic. Whilst these might not be relevant for all projects, do consider if there are relevant for your project. Where there are relevant security issues, discuss how they will this affect the work that you are doing. Carry forward this discussion into relevant areas for design, implementation and testing.

## Process

You need to describe briefly the life cycle model or research method that you used. You do not need to write about all of the different process models that you are aware of. Focus on the process model that you have used. It is possible that you needed to adapt an existing process model to suit your project; clearly identify what you used and how you adapted it for your needs.

# Design

## Overall Architecture

### 2.1.1 Tools and Technologies

Very important part of staring the project is to select programming language and libraries that will be used. More about blabla

2.1.1.1 Programming Language

When choosing a programming language, main criterias were usability for the project, its libraries, ease of use and experience. The language should provide:

* Machine learning and natural language processing libraries
* Simple Web framework
* Ease of experimenting
* Simple tools to gather the data from web APIs
* Simple tools to do data manipulations and analysis

In this project Python was the best choise. It provides very popular and in-depth machine learning libraries (NLTK, Scikit-learn, Textblob), web frameworks (Flask, Django, Pyramid) and Interpreter which is useful to do quick, ad hoc experimenting. It also provides a Pandas library that allows to form data into DataFrames what is very handy in data analysis and provides many build-in data processing functions. This language was also used by the student during the Industrial Year. Python is also very popular and has got documentation and there is many tutorials and snippets available in the Internet.

Other languages that were took into account are:

* **R** - has good ML and data processing libraries but has not got any libraries providing a way to create a web interface.
* **Ruby** – provides good web framework (Ruby on Rails) but does not provide good libraries/gems to do ML. There is Weka for JRuby or other ways to use Java Weka library in Ruby.
* **Java** – Provides many web framworks such as Spring, JSF or Vaadin. XX also can use Weka (Waikato Environment for Knowledge Analysis) that is a very popular software suite.

Moreover choosing Python is a good opportunity to develop skills that are used in the industry and meet the [needs of the labour market](http://context.reverso.net/tłumaczenie/angielski-polski/needs+of+the+labour+market).

In the project was used the most recent Python version what is 3.6.

2.1.1.2 Libraries

To do a language processing, firstly was used Textblob librarty which is build on top of NLTK but unfornatelly due to the poor documenion and lack of its capabilities there was made a deicision to move into NLTK.

To build a classificator which predicts market changes there was used Scikit-learn. It has got good documentation with many examples and there is also many topics about it on sites such as StackOverflow.

Other possibility would be to use Weka but it would require to use either Jython(implementation of the Python language for the Java platform[3]) or run weka library using wrappers around JNI calls such as javabridge(package that allows Python to interact with JVM[4]) but both ways are too complicated.

The web interface of the program is very simple so I chose Flask. It is a micro web framework using Jinja2 template engine. It allows to create simple pages in very easy way. I did not use Django because it is rather better for more complex webpages. Pyramid is also good for creating simple webpages but is much less popular, what makes it harder to find some solutions ETC in the Internet.

I also used Mlxtend library to sue Apriori algorithm, Nose to do unittests and Flask-SQLAlchemy to quickly create a simple database in Flask.

To do data processing I also used Pandas and NumPy which are the most popular Python libraries providing …BLLBLB

To scrap tweets from Twitter API I used a Tweepy library which is the most popular and very simple.

## Detailed Design

The project can be split into main two parts: market analisys and web interface. Therefore it was split the into three packages separating project concerns:

* Markets package
* Webpage package
* Tests package

**2.2.1 Markets package**

All the code in “markets” package was split into modules that group together logically related code.

DODAC ZE PYTHON TO NIE TYLKO OOP I TO BARDZIEJ TPYOWE DLA PYTHONA

**Dataset** – a module with a TweetsDataSet class that wraps a pandas Dataframe and represents a set of tweets with their features, sentiments and market affects.

**Tweeter Scrapping** – a module used to scrap tweets from Tweeter.

**Phrases Extraction** – contains a PhrasesExtractor class which builds a vocabulary of the XXX phrases and words ZZZ from set of texts and then extracts those features from POJEDYSCzych tweets.

**Sentiment Analysis** – module containing a class responsible for tweets sentiment analysis. It wraps NaiveBayesClassifier from NLTK library and uses a PhrasesExtractor to extract features from tweets which are then used to train a model or predict a value of the particular tweet. XXX was wrapped in a class because it is more convenient to load and save the Analyser and test it.

**Tweets Features Extraction** – contains all the functions used to extract features from the tweet such as sentiment and phrases/words in the tweet. To gather those informations uses SentimentAnalyser and PhrasesExtractor objects.

FeatureSelection – module containing functions responsible for selecting the most significative sets of features to obtain the best accuracy.

**Currency Analysis** – a main module that connects all the functionalities. There is a CurrencyAnalyser class that is used to analyse a CSV file of stock prices and provide results of this analysis such as assosiation rules, model to predict markets or the most coefficient features. It holds the functionality of reading files and saving the results.

**Association** – a module holding all the code that is responsible for reading stock prices from files and merging then with tweets datasets.

**Market** **Predicting**– contains all the code that is responsible for trainig a classifier that predicts stock changes. Contains 3 classes: Classifier representing a classifier model (MultinomialNB by default) and wrapping all its functionality, AnalysResult that represent a result of a single tweet analysis, MarketPredictingModel that contains two Classifier objects and decides which one to use to do a prediction.

**Rules** – contains functions used to do association rules learning.

The markets module has got also a “data” and “pickled\_models” direcotries. “data” folder stores all data used to do analysis such as a list of stopwords, scrapped tweets and CSV files generated by the application. “pickled\_models” stores saved sentiment analysis and market predicting models.

**2.2.2 Webpage package**

Webpage module has got a typical structure for Flask projects.

It is split into:

* Static folder – holding static files such as images or css styles
* Templates folder – holding Jinja2 templates that are filled with content by views.
* Views.py – this is where routes are defined. It defines routes for each currency and gathers data that is send to the templates and presented.
* models.py – holds Currency model. This model stores information about currencies such as its name and accuracy of the model in database.
* \_\_init\_\_.py – Initializes the application, sets up its configuration and database.
* Currencies.db – stores Currencies models that can be loaded when application runs.

**2.2.3 Others**

Apart from these 3 packages there is few more files typical for a Flask project:

* -manage.py – a script used to init/drop database, fill database with some sample data and to run the webpage.
* -requirements.txt – file containing a list of packages that are used in the project and have to be installed
* -README – file explaining how to run the program

## Tools used to develop the project

For my Python IDE I chose PyCharm made by JetBrains company. I used it during my Industiral year and I really liked it. It has got all code assistance features such as syntax and error highlighting. It supports Flask projects and many file extensions such as html, css, js, csv which I used in a project. It also has got integrated debugger which I used a lot.

To keep track on changes and have a backup of the work I set up a Github repository. Backups of the work were kept on the Github repository and two machines the student worked on. As a Git client for the machine with Windows OS was chosen GitKraken and command line git for the machine with Linux.

# Implementation

It is more likely that things were more complex than you first thought. In particular, were there any problems or difficulties that you found during implementation that you had to address? Did such problems simply delay you or were they more significant?

3.1 Data gathering

3.1.1 Tweets scrapping

First step was to gather Donald Trump tweets. For this purpose, there was used TweePy library that allows to fetch data from Twitter REST API. To communicate with the API it was necessary to create an account and obtain a customer key and access tokens.

The first task to do with tweets was a sentiment analysis, there had to be collected small amount (about 120) of tweets manually. Writing a scrapping script was very useful at this stage because it allowed to fetch particular tweets data from the API by the ID of the tweet. The data such as tweet id, creation date and text was saved to the CSV file.

Next step was to scrap all the tweets since the begin of 2017. It is a bit before Donal Trump become a president so his tweets began to have some influence on markets. Tweepy provides also a special function to get posts from users’ timeline. Unfortunately, it allows to get only 200 tweets at once so I had to do it sequentially. Twitter also allows only to scrap last 3240 of user tweets and hopefully there was obtained 2935 tweets from 01.2017 to 03.2018.

3.1.2 Stocks data

Obtaining stocks indices data was XX part of the project because all the webpages that archive historical stock data provide only daily-interval stock prices changes due to the amounts of this data. All the websites XXX during the project XX data with smaller intervals (such as hourly changes) for money. In consequence, all markets changes analysed in the project are daily open-close price changes.

3.2 Sentiment Analysis and Phrases Extraction

3.2.1 Data selection

To begin with a sentiment analysis there was scrapped 120 tweets manually and their sentiment was marked manually. It was decided that half of them had to be positive and half should be negative to have balanced classes – ADD WHY?

Tweets were selected regardless of the date. The most important aspect was to find ones that are clearly positive or very negative to train the model as best as possible.

Tweets were found using Google by searching phrases such as “Most positive Trump tweets”, “The worst Trump tweets”. Very useful was the website: <http://www.trumptwitterarchive.com/> where we can see the most popular keyword in his tweets and search for them.

3.1.2 Building a model

When the dataset was selected, the next step was to build a classifier.

To train a model tweets had to be split into folds to do a cross-validation what prevents overfitting and gives more reliable results. Due to the fact that scikit-learns Kfold functions seemed to be complicated there was a decision to write a folding function manually. The code was splitting a corpus into K chunks with preservation of stratification (each chuck had half of the tweets positive and half negative).

Building a text classification system with Textblob is very trivial:

cl = NaiveBayesClassifier**(**train\_data**)**

cl.classify**(**"This is an amazing library!"**)**

The Classifier object just has to be fed with list of tuples, and each tuple has to consist of tweet text and marked sentiment.

To ensure that results are reliable the training process was run 40 times with 10-folds cross validation and the result accuracy is a mean from those 40 runs. The results seemed to be surprisingly good. Naïve Bayes had 82% of test accuracy. Unfortunately, after investigating the most informative features it turned out that the most decisive features were words such as “are”, “there” or even punctuation marks such as brackets. That meant that the model was overfitted and instead of making decisions based on words such as “good” or “bad” was using the most common words in the language. Even the sentence contained many negative words like for example: “Crocked Hillary Clinton ….” Was marked as positive because it just had a word “AND”.

3.1.3 Features extraction

Once it had been discovered that TextBlob does not extract phrases properly and it does not provide easy option to change it, it was necessary to move into NLTKs’ classifier.

Next step was to write a custom feature extracting function that splits tweets into words. To do so were used very simple NLTKs’ functions: sent\_tokenize and word\_tokenize that split text into sentences and then into words. All the extracted words then had been lowercased because “Then” and “then” is the same word. Unfortunately, even these functions are the part of so popular and XXX library, they had problems with splitting even simple sentences and words such as “doesn’t’ were separated into “does” and “’t”. Moreover, the model was still making decision basing on words such as “Did”, “of”, “And”. These senseless words are called stop words. These are words that are common in the language and do not tell anything about the meaning of the sentence.

Once it had been discovered that NLTK tokenizing functions cannot handle extracting words and phrases there was few alternatives checked out such as NLTKs ConllExtractor and FastExtractor. Most of them had problems with splitting the sentence properly. They either extract useless stop words or do not extract half of the important phrases. The only one that does it very well is a TextRazor – cloud service providing a deep learning analysis using their web API. Unfortunately, TextRazor is not free so the decision was made to write a custom phrase extractor.

After some research there was found easy to implement algorithm called RAKE(Rapid Automatic Keyword Extraction). There is many its implementations in the web but it was decided to write own extractor basing on LINK HERE. This one was selected because it is simple but it had to be modified because does not have enough configuration options and has too many redundant functions. TO DODALEM ALE USUNAELM Originally RAKE extracts also adjusted keywords (ones that include a stop word such as United States OF America) but this functionality was dropped due to the lack of these phrases in the corpus and risk of leading to unnecessary False Positives.

Feature extractor algorithm basically extracts candidate phrases by splitting text by stop words. A list of stop words used in the project was downloaded from the Internet. LINK HERE

Then candidate phrases are sifted by their length and number of occurrences. Phrases that are not accepted are split into words.

Accuracy of sentiment analyser while using a custom feature extractor increased to 84% and it is more reliable because the most informative words now are more sensible:

SCREEN HERE

The next step to improve the accuracy was to lemmatize words. It means that all inflected words were reduced to the root form (for example playing, plays, played into play). This time NLTKs lemmatize\_word function was sufficient. When subjected all the words to lemmatization, a test accuracy has increased to 86%.

The overall process of building a vocabulary looks as follows:

SCREEN HERE

The process of training a classifier looks as follows:

* Building a phrases extractor vocabulary from tweets
* Extract features from each tweet
* Train a model using market features and their target (sentiment)

Then the process of analysing a particular tweet looks as follows:

* Pre-process tweet (lower case, split by punctuation, remove unnecessary characters)
* Find matching phrases and remove them from tweet
* Split the rest by stop words to get rid of them
* Lemmatize words
* Mark words matching with the vocabulary
* Pass the features vector to the model to classify it

3.4 Markets predicting

Once the sentiment analysis part was done, the next step was to build a market predicting model.

MOZE TU WRZUCIC O TWEETERZE?

Before training a classifier, the dataset was preprocessed:

- all the tweets containing only video/image and no text were removed

- tweets written in languages other than English were removed

- Unicode characters were removed such as ✔💜 ➡✅

- some Unicode characters were changed into proper words or characters: &amp -> and

- from the tweets were removed all the links

- manually removed all the tweets that could not have any impact such as “Happy birthday”

- manually removed all short and meaningless tweets such as “Jobs, Jobs, Jobs”

- removed all retweets

- merged tweets that were separated into few tweets because were too long (Twitter allows tweets to be max XXX chars length so when they are longer they are split into few others and they start or end with “…” ) EXAMPLE?

Then the dataset has decreased into 2026 tweets. All the tweets got the market change set up using the CSV file with stock prices. It was assumed that tweets affect the market within few hours so each tweet had got assigned a percent change of the index during the day it was published. It was also decided that all the tweets that are published after 10pm got the change value from the next day. MORE HERE

Market changes above 0 were marked as positive and the rest as negative what gave a binary target.

Then for each of the tweets sentiment was predicted and features extracted.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Tweet** | **Sentiment** | **Feature 1** | **Feat 2** | **Feat 3** | **…** | **Feat n** | **Change** |
| Tweet content | Pos | 1 | 0 | 0 | … | 1 | 0.05% |
| Tweet content | Neg | 0 | 1 | 0 | … | 0 | -0.12% |

That prepared dataset could be now used to train a model.

Experiments were performed using Weka J48 and SciKitLearns Naïve Bayes and Logistic. All of them was run with 10-fold cross validation. The base rate accuracy was 52.6%

**OPISAC JAK BUDOWALEM MODEL i ze pusiclem 30 run x 10 folds – 30 bo dawalo już tyle samo co 100**

|  |  |  |
| --- | --- | --- |
| Classifier | Test accuracy | Train accuracy |
| Naïve bayes in scikit-learn | 54.9% | 90.6% |
| Logistic in scikit-learn | 51.6% | 100% |
| J48 in Weka | 51.2% | 81.5% |

Feature extractor was extracting over 6000 of features form the dataset what was a lot and the model needed a lot of time to train. The model was also trained using features that occurred only once or twice what just lead to overfitting.

Therefore, feature selection was necessary. Running any feature selector on that big dataset would result on a very long time of processing so it was decided to remove features that occur only few times. After a bit of experimenting it was decided to remove features that occur less then 6/7 times because it decreased a number of features to more practicable size (1185). It also ZMNIEJSZYLO overfitting and gave a bit better test accuracy:

|  |  |  |
| --- | --- | --- |
| Classifier | Test accuracy | Train accuracy |
| Naïve bayes in scikit-learn | 56.6% | 70.3% |
| Logistic in scikit-learn | 52.0% | 84.0% |
| J48 in Weka | 51.8% | 76.7% TO JEST RANOM |

Once the number of features was dropped it was easier to try different feature selectors. The easiest way was to export the dataset to a CSV file and perform a selection in Weka. Many trials have been carried out with various selectors and finally the “Wrapper Subset Evaluator” proved to be the best.

It selected 116 best features what gave ILE TAM TERAZ? I ILE TWEETOW?

|  |  |  |
| --- | --- | --- |
| Classifier | Test accuracy | Train accuracy |
| Naïve bayes in scikit-learn | 67.8% | 70.0% |
| Logistic in scikit-learn | 66.8% | 69.2% |
| J48 in Weka | 62.0% | 69.4% TO JEST RANOM |

After that Weka was not used anymore to build a classifier due to the fact that uses only one thread, it is extremely slow and it is more convenient to process data and build classifier in python at one go without saving to file and bothering with opening it in Weka.

Target change

Since some of the tweets are completely neutral and have no influence into markets, there was add third target: No change…..

In the USD Index dataset there is about 3% of days that the stock did not change at all. To get three classes more balanced there had to be some threshold set up to increase a set of “No change” objects. Following feedback from the project supervisor, it was decided that the threshold should be calculated using a standard deviation. To obtain about 1/3 of the targets as a no change the threshold is calculated by 1/3 of the standard deviation distance from the mean:

def calculate\_thresholds(stock\_prices):  
 mean = stock\_prices.mean()  
 sigma = stock\_prices.std(ddof=0)  
 lower\_threshold = (mean - (sigma / 3)).round(2)  
 higher\_threshold = (mean + (sigma / 3)).round(2)  
 return lower\_threshold, higher\_threshold

DODAC WYNIKI

NAPISAC ZE DODALEM MARKING AGAIN

WYNIKI

SENTYMENT Z POS/NEG NA 0.1-1

WYNIKI

AUTOMATING

TO AUTOMATE UZYLEM REFCV wa simplyfying the problem

Ze weka w processie, nowe currencies

RULES

3.4.3 Feature Selection

3.5 Rules learning

3.6 Flask webpage

Selection- a small class used to do feature selection using MultinomialNB and RFECV from Scikit-learn.

# Testing

Detailed descriptions of every test case are definitely not what is required here. What is important is to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.

Provide information in the body of your report and the appendix to explain the testing that has been performed. How does this testing address the requirements and design for the project?

How comprehensive is the testing within the constraints of the project? Are you testing the normal working behaviour? Are you testing the exceptional behaviour, e.g. error conditions? Are you testing security issues if they are relevant for your project?

Have you tested your system on “real users”? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

The following sections indicate some areas you might include. Other sections may be more appropriate to your project.

## Overall Approach to Testing

Initially all the experiments were performed just in one go. The data was loaded into a DataFrame and processing was performed in a pipeline. That approach was quick for experimenting but hard to test. Unittesty dla sentymentu I phrases

When the tweets processing and classifying part of the code became more complex, the program was split into more object oriented way. All the DataFrame code was wrapped into a DataSet class encapsulating all the DataFrame operations what ULATWILO testing because more units was possible to test without involving DataFrames and creating whole datasets just to test one simple function.

Unit tests and integration tests

Unit tests are good, prove quality, and making sure that smaller bits of code work we can do integration tests.

## Automated Testing

### Unit Tests

### User Interface Testing

### Stress Testing

### Other Types of Testing

## Integration Testing

## User Testing

# Critical Evaluation

Examiners expect to find in your dissertation a section addressing such questions as:

* Were the requirements correctly identified?
* Were the design decisions correct?
* Could a more suitable set of tools have been chosen?
* How well did the software meet the needs of those who were expecting to use it?
* How well were any other project aims achieved?
* If you were starting again, what would you do differently?

Other questions can be addressed as appropriate for a project.

Such material is regarded as an important part of the dissertation; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things and room for improvement with any project. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

In the latter stages of the module, we will discuss the evaluation. That will probably be around week 9, although that differs each year.

* If you were starting again, what would you do differently?
* Write data processing in more object-oriented way instead of using DataFrames. I would just make a DataSet class that would have a list of tweets and each tweet would store information about its features and sentiment. I think that I would make processing slower but it would help to achieve more modularity and lower cohesion. Therefore, testing would be easier.
* I think that if code was written more modular then it would be easier to test and mocking would not be so much need in some places.

# Appendices

SHOULD I INCLUDE ANY STORIES?

If you have taken an agile approach to developing the project, then you may be less likely to have developed a full requirements specification. Perhaps you use stories to keep track of the functionality and the ’future conversations’. It might not be relevant to include all of those in the body of your report. Instead, you might include those in an appendix.

1. Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. If third party code or libraries are used, your work will build on that to produce notable new work. The key requirement is that we understand what is your original work and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

As an example, you might include a definition such as:

**Apache POI library** – The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the client’s existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [5]. The library is released using the Apache License [6]. This library was used without modification.

TextBlob – a free library for processing textual data.( FROM http://textblob.readthedocs.io/en/dev/index.html) Was dropped and scikit learn was used in lieu

SciKit-Learn – Python machine learning library. It was used to do cross-validation and build classifiers It is free and open source. This library is released using BSD license.

Weka – is a suite of [machine learning](https://en.wikipedia.org/wiki/Machine_learning) software written in [Java](https://en.wikipedia.org/wiki/Java_(programming_language)). It is [free software](https://en.wikipedia.org/wiki/Free_software) licensed under the [GNU General Public License](https://en.wikipedia.org/wiki/GNU_General_Public_License). (<https://en.wikipedia.org/wiki/Weka_(machine_learning)>)

It was used to for experimenting with the data and to do a feature selection. It is run by the program by a subprocess XXX command. Weka Jar file is included into the project directory. Version 3.8.2 was used.

Mlxtend – The project used this library to do association rule learning with Apriori algorithm. It is released under BSD licence. Version used 0.11. (https://github.com/rasbt/mlxtend)

Pandas – library used to do data manipulation and analysis. Library is released using BSD license. Version used 0.22.0

TweePy – Python library used to access the Twitter API. It is released using MIT license. Version used 3.6.0 (https://github.com/tweepy/tweepy/blob/master/LICENSE)

All those libraries were used without modification.

1. Ethics Submission

This appendix includes a copy of the ethics submission for the project. After you have completed your Ethics submission, you will receive a PDF with a summary of the comments. That document should be embedded in this report, either as images, an embedded PDF or as copied text. The content should also include the Ethics Application Number that you receive.

1. Code Samples

This is an example appendix. Include as many appendices as you need. The appendices do not count towards the overall word count for the report.

For some projects, it might be relevant to include some code extracts in an appendix. You are not expected to put all of your code here - the correct place for all of your code is in the technical submission that is made in addition to the Final Report. However, if there are some notable aspects of the code that you discuss, including that in an appendix might be useful to make it easier for your readers to access.

As a general guide, if you are discussing short extracts of code then you are advised to include such code in the body of the report. If there is a longer extract that is relevant, then you might include it as shown in the following section.

Only include code in the appendix if that code is discussed and referred to in the body of the report.

Random Number Generator

The Bayes Durham Shuffle ensures that the pseudo random numbers used in the simulation are further shuffled, ensuring minimal correlation between subsequent random outputs.

// Some example code here…

# Annotated Bibliography

This final section should list all relevant resources that you have consulted in researching your project. Each reference should also include a brief annotation.

1. Neil Mac Parthaláin, “MMP: Project descriptions”, 2018 (Online) Available at: https://teaching.dcs.aber.ac.uk/mmp Accessed April 2018.

2. <https://twitter.com/realdonaldtrump/status/901802524981817344>, 27th Febuary 2017. Accessed April 2018.

This is of Donald Trumps’ Tweets.

3. “What is Jython?”, Jython webpage <http://www.jython.org/archive/21/docs/whatis.html> Accessed April 2018POCZ

4. https://pypi.python.org/pypi/javabridge/1.0.17

1. Sylvia Duckworth. A picture of a kitten at Hellifield Peel. <http://www.geograph.org.uk/photo/640959>, 2007. Copyright Sylvia Duckworth and licensed for reuse under a Creative Commons Attribution-Share Alike 2.0 Generic Licence. Accessed August 2011.  
     
   This is my annotation. I should add in a description here.
2. Mark Neal, Jan Feyereisl, Rosario Rascunà, and Xiaolei Wang. Don’t touch me, I’m fine: Robot autonomy using an artificial innate immune system. In *Proceedings of the 5th International Conference on Artificial Immune Systems*, pages 349–361. Springer, 2006.   
     
   This paper…
3. W.H. Press et al. *Numerical recipes in C*. Cambridge University Press Cambridge, 1992.  
     
   This is my annotation. I can add in comments that are in **bold** and *italics*and then further content.
4. Various. Fail blog. <http://www.failblog.org/>, August 2011. Accessed August 2011.  
     
   This is my annotation. I should add in a description here.
5. Apache Software Foundation (2014) “*Apache POI - the Java API for Microsoft Documents*” (Online) Available at: [http://poi.apache.org](http://poi.apache.org/) Accessed: 14th March 2014.
6. Apache Software Foundation (2004) “Apache License, Version 2.0” (Online) Available at: <http://www.apache.org/licenses/LICENSE-2.0> Accessed: 14th March 2014.
7. Neil Taylor, “MMP: Final Report and Technical Work”, 2017 (Online) Available at: <http://blackboard.aber.ac.uk/> Accessed 26th April 2017.

A document that outlines information about the marking guide for the Final Report and Technical Work. This document was referred to as Structure of the Final Report before academic year 2016-2017. This is published in the Assignments folder. If you are logged in to Blackboard, you can access the folder using <http://jump.aber.ac.uk/?sxxpt>.