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

**Abstract**

“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 6](#_Toc512375718)

[1.1. Background 6](#_Toc512375719)

[1.1.1. Trump Tweets 6](#_Toc512375720)

[1.1.2. Machine Learning and Data Mining 6](#_Toc512375721)

[1.1.3. Similar Systems 6](#_Toc512375722)

[1.2. Analysis 7](#_Toc512375723)

[1.3. Process 7](#_Toc512375724)

[2. Design 8](#_Toc512375725)

[2.1. Overall Architecture 8](#_Toc512375726)

[2.1.1. Tools and Technologies 8](#_Toc512375727)

[2.1.2. Programming Language 8](#_Toc512375728)

[2.1.3. Libraries 8](#_Toc512375729)

[2.1.4. Data storage 9](#_Toc512375730)

[2.2. Detailed Design 9](#_Toc512375731)

[2.2.1. Markets package 9](#_Toc512375732)

[2.2.2. Webpage package 10](#_Toc512375733)

[2.2.3. Others 10](#_Toc512375734)

[2.3. Tools used to develop the project 11](#_Toc512375735)

[3. Implementation 12](#_Toc512375736)

[3.1. Data gathering 12](#_Toc512375737)

[3.1.1. Tweets scrapping 12](#_Toc512375738)

[3.1.2. Stocks data 12](#_Toc512375739)

[3.2. Sentiment Analysis and Phrases Extraction 12](#_Toc512375740)

[3.2.1. Data selection 12](#_Toc512375741)

[3.2.2. Features extraction 13](#_Toc512375742)

[3.3. Markets predicting 15](#_Toc512375743)

[3.4. Rules learning 19](#_Toc512375744)

[3.5. Flask webpage 20](#_Toc512375745)

[4. Testing 21](#_Toc512375746)

[4.1. Unit testing 21](#_Toc512375747)

[4.2. Integration/Acceptance Testing 22](#_Toc512375748)

[4.3. Usability testing 22](#_Toc512375749)

[5. Critical Evaluation 22](#_Toc512375750)

[Appendices 24](#_Toc512375751)

[A. Ethics Submission 26](#_Toc512375752)

[Annotated Bibliography 29](#_Toc512375753)

[1. Neil Mac Parthaláin, “MMP: Project descriptions”, 2018 (Online) Available at: https://teaching.dcs.aber.ac.uk/mmp Accessed April 2018. 29](#_Toc512375754)

# 1. 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.*

## Background

### Trump Tweets

Trump is known for his controversial, outragejous and sometimes very hateful tweets. He tweets about 10 times every day and his account is followed by over 50 million people. His account is used to attack his opponents and …

His tweets have got massive impact into markets and political scene. He admitted that without Tweeter he would not be a president.

(<https://www.independent.co.uk/news/world/americas/us-politics/donald-trump-tweets-twitter-social-media-facebook-instagram-fox-business-network-would-not-be-a8013491.html>)

His simple language makes them easy to analyze. He never uses a sarcasm or exquisite words. Due to the fact that his vocabulary is fairly small, it is easy to build a machine learning classifier to do **WHAT**?

His tweets are also very emotional, petulant and sometimes aggressive what makes it easy to determine their sentiment by a computer. They also follow some patterns, most of his positive tweets end up with “Make America Great Again” phrase.

Analysing this data seemed to be very interesting because **BLABLABLA**

### Machine Learning and Data Mining

Machine learning and Data Mining becoming very popular nowadays. The Internet is full of data that can be analyzed and BLABALBA

### Similar Systems

Tweets sentiment analysis, no clue what was used for

<https://dev.to/rodolfoferro/sentiment-analysis-on-trumpss-tweets-using-python->

Interesting was

<http://varianceexplained.org/r/trump-tweets/>

the analysis made by David Robinson (Chief Data Scientist at DataCamp)

comparing tweets content with the device they were sent from (part of tweets is sent from iPhone and some from Android). The results were interesting: most negative tweets attacking his rivals were sent from Android whereas iPhone was used more for benign announcements. The analysis concludes that tweets from these devices are written by different people. Almost all the tweets sent with a picture or hashtags come from iPhone and most of “emotionally charged” words were common for Android device. What we can learn notice from the analysis is that iPhone tweets are probably sent by people involved in planning his schedule because words like “join” or “tomorrow” come from iPhone. Fact that Android tweets are more objective may mean that these ones are sent by his public relations specialists or either by himself.

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

## Analysis

Most of the Twitter sentiment analysis examples in the Internet use ready-made analyzers like the one built-in in the TextBlob. This approach is fast and easy but too general. It was assumed that it is better to build own classifier because peoples languages are different and one people words can have different emotions when spoken by someone else. For example, words like “Mexico”, XXX, XXX are neutral but in Trump's tweets, they usually have a negative attitude. Another example could be “Make America Great Again” which in all of the cases goes in positive tweets.

MUCH MORE THERE

## 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 to your needs.*

Deciding about the development process was simple. It was decided to use agile methodologies because ABOUT AGILE HERE

We are sure that if there was be too little time then we would still be able to deliver some working software.

With waterfall methodology, it would be very likely that there will not be enough time to do tests because implementing all of the features took too much time.

The agile methodology that fit the project specification the best is XP (Extreme Programming). Weekly meetings with a supervisor can be treated as client meetings during which a further direction is determined. Each week can be treated as a sprint within which there should be planned adding a small functionality to the project. Pair programming obviously could not be implemented due to the only one developer but the rest of XXX fits the project perfectly.

The project was split into sprints:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | W1 | W2 | W3 | W4 | W5 | W6 | W7 | W8 | W9 | W10 |
| Setting up Git, IDE, research |  |  |  |  |  |  |  |  |  |  |
| Data collection |  |  |  |  |  |  |  |  |  |  |
| Sentiment analysis |  |  |  |  |  |  |  |  |  |  |
| Phrases extraction |  |  |  |  |  |  |  |  |  |  |
| Markets model building |  |  |  |  |  |  |  |  |  |  |
| Feature selection |  |  |  |  |  |  |  |  |  |  |
| Creating a webpage |  |  |  |  |  |  |  |  |  |  |
| Adding new currencies |  |  |  |  |  |  |  |  |  |  |
| Integration tests |  |  |  |  |  |  |  |  |  |  |
| Writing a report |  |  |  |  |  |  |  |  |  |  |

Alternatives that were considered for the project were Scrum and XXX. Scrum would be rather more efficient in teamwork that in single person project.

# Design

## Overall Architecture

### Tools and Technologies

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

### Programming Language

When choosing a programming language, main criteria 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 choice. 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 forming data into DataFrames what is very handy in data analysis and provides many built-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 are many tutorials and snippets available on the Internet.

Other languages that were taken 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 frameworks such as Spring, JSF or Vaadin. XX also can use Weka (Waikato Environment for Knowledge Analysis) that is a very popular software suite.

MORE ABOUT LANGUAGES

Moreover, choosing Python is a good opportunity to develop skills that are used in the industry and meet the needs of the labour market.

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

### Libraries

To do a language processing firstly was used Textblob library which is built on top of NLTK but unfortunately, due to the poor documentation and lack of its capabilities, there was made a decision 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 are also many topics about it on sites such as StackOverflow.

Another 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 creating simple pages in the very easy way. I did not use Django because it is rather better for more complex web pages. Pyramid is also good for creating simple web pages but is much less popular, what makes it harder to find some solutions ETC on the Internet.

I also used Mlxtend library to sue Apriori algorithm, Nose to do unit tests 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.

### Data storage

Write about CSV and SQLALCHEMY

## Detailed Design

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

* Markets package
* Webpage package
* Tests package

### Markets package

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

Write that python is not oop and that’s why there are almost no classes

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

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

**Phrases Extraction** – contains a PhrasesExtractor class which builds a vocabulary of phrases and words found in the set of texts and then extracts those features from particular 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. All the functionality was wrapped in a class because it is more convenient to load and save the Analyser and perform any tests.

**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 information uses SentimentAnalyser and PhrasesExtractor instances.

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

**Currency Analysis** – the main module that connects all the functionalities. There is a CurrencyAnalyser class that is used to analyze a CSV file of stock prices and provides results of this analysis such as association 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 them with tweets datasets.

**Market** **Predicting**– contains all the code that is responsible for training a classifier that predicts stock changes. Contains 3 classes: Classifier representing a classifier model (MultinomialNB by default) and wrapping all its functionality, AnalysResult that represents 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” directories. “data” folder stores all data used to do analysis such as a list of stop words, scrapped tweets and CSV files generated by the application. “pickled\_models” stores saved sentiment analysis and market predicting models.

### Webpage package

Web page 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 sent 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 the database.
* \_\_init\_\_.py – Initializes the application, sets up its configuration and database.
* Currencies.db – stores Currencies models that can be loaded when the application runs.

### Others

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

* -manage.py – a script used to init/drop the database, fill the 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 Industrial 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.

Keeping the project both on the GitHub server and two machines prevents the loss of the project in the event of a problem with a git or even accidental deletion of repository.

# Implementation

## Data gathering

### Tweets scrapping

The first step was to gather Donald Trump tweets. For this purpose, there was used TweePy library that allows fetching 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 fetching particular tweets data from the API by the ID of the tweet. The data such as tweet id, creation date, and text were 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 getting 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 created between 01.2017 and 03.2018.

### Stocks data

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

## Sentiment Analysis and Phrases Extraction

### 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.

**3.2.2 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 the 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”.

### Features extraction

Once it had been discovered that TextBlob does not extract phrases properly and it does not provide an 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 the text into sentences and then into words. Afterwards, all the extracted words had been lowercased because “Then” and “then” is the same word. Unfortunately, even that these functions are the part of so popular and reputable 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 were 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 an algorithm called RAKE(Rapid Automatic Keyword Extraction). There are many its implementations on 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. Originally RAKE extracts also adjusted keywords (ones that include a stop word such as United States OF America). This functionality was also added but was dropped later on 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 the candidate phrases are sifted by their length and number of occurrences. Phrases that are not accepted are split into words.

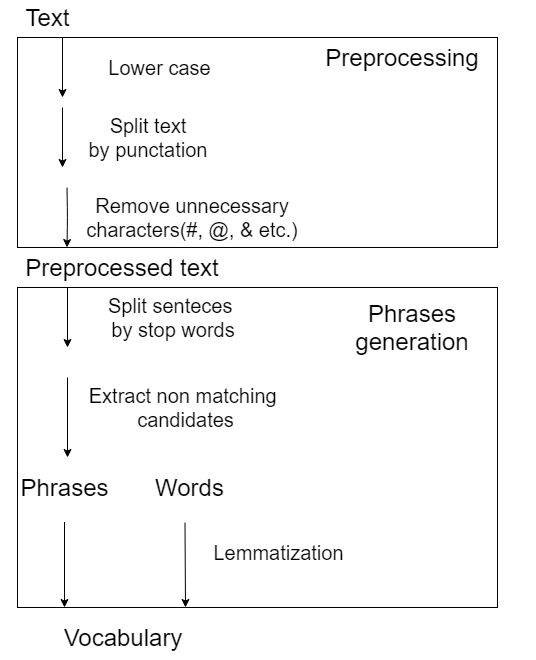
The accuracy of sentiment analyzer while using a custom feature extractor increased to 84% and it is more reliable because the most informative words now are more sensible:

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 analyzing a particular tweet looks as follows:

* Pre-process tweet (lower case, split by punctuation, remove unnecessary characters)
* Find matching phrases and remove them from a 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

## Markets predicting

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

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 140 chars length so when they exceed the limit they are split into few separate ones that 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 10 pm 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.

To build a model there was used a Scikit-learn MultinomialNB and LogisticRegressionCV. First one is a Naïve Bayes classifier implantation that is suitable for classifying discrete values such as “Down”, “No change”, “Up. Logistic regression classifier used is also known as logit, MaxEnt (<http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html>)

It uses a cross-validation to find the best hyperparameters to the data, what is very handy.

The data that is fed to the model has to be in proper format. Training data has to be split into a 2D array of features and their marks for each instance and a 1D array of results for each instance.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2D array of 2 instances with 5 features:   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1 | 0 | 1 | 0 | 0 | | 0 | 0 | 0 | 1 | 0 | | 1 | 1 | 0 | 0 | 0 | | 1D array of targets for 3 instances:   |  |  |  | | --- | --- | --- | | Up | Down | Up | |

Using a DataFrame to store data was useful there because it was easy to format this data in such a way.

Then the model was built 30 times to take an average of different randomized runs. 30 gave the same results as 100 so there was no point to run it more times.

Each run trained a model 10 times (10-fold CV) with stratified folds.

The model was build using XXX objects and over 6000 features and the accuracy that was obtained looks as follows:

|  |  |  |
| --- | --- | --- |
| **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% |

Experiments were performed using a model written in Python and also by exporting a data to the file and using J48 in Weka.

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 in a very long time of processing so it was decided to firstly remove features that occur only a few times. After a bit of experimenting it was decided to remove features that occur less than 7 times because it decreased the number of features to more practicable size (1185). It also diminished 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% |

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.

Napisac ze do PL:IKU BYŁO ITP

It selected 116 best features what gave … add how many features and tweets

|  |  |  |
| --- | --- | --- |
| **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% |

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 on markets, there was add the third target: No change…. WRITE ABOUT 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

The base rate accuracy was changed to 41% due to the change into a 3-class problem.

|  |  |  |
| --- | --- | --- |
| **Classifier** | **Test accuracy** | **Train accuracy** |
| Naïve Bayes | 52.7% | 59.7% |
| Logistic regression | 49.1% | 54.0% |

Maybe some comment about it

After some investigation, it was discovered that if a lot of features is removed then there are many objects left without any feature marked. For example, if tweet “Happy Birthday” had marked “happy” and “birthday” features marked and they were deleted what makes the instance useless. Those instances have got only one feature that is a sentiment, what could just spoil training a model.

After removing tweets that do not have any feature the dataset length decreased into XX objects.

The model trained on that dataset gave better accuracy:

|  |  |  |
| --- | --- | --- |
| **Classifier** | **Test accuracy** | **Train accuracy** |
| Naïve Bayes |  |  |
| Logistic regression |  |  |

XXXXXXXXXX

The next step was to mark features again. Due to the fact that the feature extractor extract features in more greedy?? way words are not extracted from market phrases so if “Crooked Hillary Clinton” is found then words “Crooked”, “Hillary” and “Clinton” will not be marked. ADD WHY IS IT BETTER

(Marking everything that was found was dropping the accuracy for about 2%)

If some phrases are found the text then they are removed and words that this phrase consists of are not marked.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Text** | **Crooked** | **Mexico** | **Reform** | **Proud** | **Crooked Hillary Clinton** |
| Crooked Hillary Clinton is the worst loser of all time. | 0 | 0 | 0 | 0 | 1 |

After deleting “Proud” and “Crooked Hillary Clinton” it is needed to mark features again to mark words that may have been in removed phrases.

|  |  |  |  |
| --- | --- | --- | --- |
| **Text** | **Crooked** | **Mexico** | **Reform** |
| Crooked Hillary Clinton is the worst loser of all time. | 1 | 0 | 0 |

Marking features again resulted in better accuracy and X instances less to delete

|  |  |  |
| --- | --- | --- |
| **Classifier** | **Test accuracy** | **Train accuracy** |
| Naïve Bayes | 53.8% | 60.3% |
| Logistic regression |  |  |

The next change that was performed was making a sentiment to be a continuous value. Before the sentiment was marked as 0/1 (positive/negative). Some tweets could be more neutral and some evidently emotional. Therefore, the sentiment values have been changed into continuous values from 0 to 1. Although it did not change the results for Naïve Bayes at all and Logistic regression train accuracy changed only by 0.01%. This is probably due to a large number of features and the fact that Naive Bayes treats features independently so changing one feature, and moreover, in a small extent, had no influence on the result.

Adding more stocks

The main purpose of next sprint was to implement functionality to build a model using another CSV files with stock prices. There was downloaded Euro Index that is a ratio of four major currencies (USD, British Pound, Japanese Yen and Swiss Franc) against the Euro. Due to the fact that a lot of Trump tweets were about the wall on the border with Mexico and hurting their Mexico peso with taxes (<https://www.ft.com/content/31967e6b-f227-33c4-9c8d-8952ba83b081>)

Another currency that could have some effect was Peso. Unfortunately, currency indices are CARRIED OUT mostly for XXX currencies and markets. IN consequence, there was downloaded historical data of S&P/BMV what is a ratio of an American stock market index based on 500 the largest companies to the biggest Mexican stock exchange index.

Adding new currencies involved a lot of refactoring. All the process of building model was changed to work in one go and data files are not hardcoded anymore. For each currency, the program retrieves a CSV file that has to be called “currency + \_Index.csv”, then the file is read and merged with tweets. Features are selected individually for each stock what in consequence selects most efficient features for each data. That is because targets for each stock are different so different instances (and their features) have an impact on the result.

The most problematic was to do a feature selection. Until that moment there were filtered features that had been selected in advance in Weka. To automate whole process features had to be selected another way. Three possible ways were identified:

-feature selection using Scikit learns selectors

-feature selection using other third-party libraries

-feature selection in Weka

The simplest way seemed to be using Scikit-learn. EVEN that it is the most popular framework among machine learning community, it includes only a few selectors.

* “VarianceThreshold” - that removes features with low variance, what is useless because all the infrequent features are already dropped. (all that occurred less than 7 times)
* Univariate selectors such as KBest with chi2XXXX. These gave very bad results because they rank features and select K best from the top of the ranking. That approach is ineffective because features sets are chosen against their helpfulness alone, not when there are together. For example, some features can significantly improve selection when there are used together but they may be useless while being alone. In case of the ranking, they would be near to the bottom of the list and would not be selected. Besides, many of the features from the top of the ranking may be correlated (both would duplicate the information they provide so one of them would be sufficient). Moreover, correlated features make training slower and may change model outputs.

Some algorithms such as Naïve Bayes can actually benefit or loss due to the redundant features depending on their correctness. (NB treats features independently so if many correlated features are correct then they can improve the result, but on the other hand, they can spoil it when features are misinformative). That is why the best solution is to use Wrapper method for feature selector because it removes features that do not contribute directly to the performance. (<https://datascience.stackexchange.com/questions/24452/in-supervised-learning-why-is-it-bad-to-have-correlated-features>)

Recursive feature elimination. Eliminator requires **being** given an external estimator that WEIGHTS features. Then they are selected by recursively considering smaller and smaller sets of features.

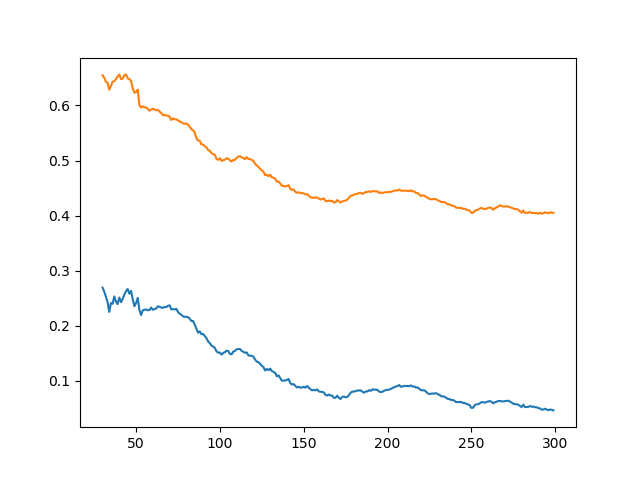
(<http://scikit-learn.org/stable/modules/feature_selection.html>)

After training on the whole set of features, coefficient marks are taken into account and the less important features are pruned. The process of pruning is repeated until the user-defined number of features is reached.

It was decided to try out RFECV (Recursive feature elimination with cross-validation) which tunes the number of features to prune with cross-validation. (same link here)

The taken approach was to set a wide range of features number (20-300) to select by RFE and then choose the best set of features.

RFE was returning results for each number of features to prune and the results were much better than when using Wekas’ wrapper.



The orange line is test accuracy of the model with X number of features. The blue line represents a difference of test accuracy and base accuracy (changing a number of features was altering the size of the majority class).

Unfortunately, what was discovered is that selecting lower and lower number of features was simplifying the problem because tweets that did not include any of the selected features were dropped. It resulted in a smaller number of tweets that were taken into account. Therefore, this way was incorrect because a lower number of features and tweets is extremely simplifying a problem and the model would be able to predict the target only for tweets that have got any of the small set of features.

WEKA IN PYTHON

Due to the lack of possibilities to select features using Scikit-learn, other options have been considered. There were found Mlxtend library – A library of extension and helper modules for Python's data analysis and machine learning libraries. (https://github.com/rasbt/mlxtend)

There were tested “ExhaustiveFeatureSelector” and “SequentialFeatureSelector” from such a module but both required a lot of time to process and gave very poor results.

Thus it was decided to use SPRAWDZONEGO Wrapper Evaluator from Weka.

Weka is a Java library and there two options considered to integrate it with the project.

One option was to run Weka using wrappers that allow Python interact with JVM such as JavaBridge, Py4j, Jpypy.

Another way was to run Weka as a system process and parse the output. This way seemed to be the simplest and was implemented.

There was added a Weka jar file into a project directory and the package is run as system command using “subprocess” module:

def run\_weka\_with\_file(temp\_filename):  
 command = ['java', '-classpath', WEKA\_JAR\_FILE,  
 'weka.attributeSelection.WrapperSubsetEval',  
 '-T', '0.5',  
 '-B', 'weka.classifiers.bayes.NaiveBayesMultinomial',  
 '-s', 'weka.attributeSelection.BestFirst',  
 '-i', temp\_filename]  
 stdoutdata = subprocess.getoutput(command)

Then the features have to be parsed from the output. Obviously, this way is not perfect because has got drawbacks such as fact that the exceptions would have to be caught and parsed in the output. It is also harder to debug and test.

The model is then build using a dataset that is sifted with selected features.

This way it was chosen 103, 99, 103 features and 1173, 1121, 1081 tweets for USD, EUR and MEX respectively, what are reasonable values.

Results of running the program with new currencies and feature selection done with Weka looks as follows:

(Base rate accuracies are 41%, 36%, 39% for USD, EUR and MEX respectively.)

|  |  |  |  |
| --- | --- | --- | --- |
| **Currency** | **Classifier** | **Test accuracy** | **Train accuracy** |
| USD | Naïve Bayes | 53.8% | 60.3% |
| Logistic regression | 52.0% | 57.4% |
| EUR | Naïve Bayes | 52.6% | 58.8% |
| Logistic regression | 41.7% | 50.3% |
| MEX | Naïve Bayes | 51.2% | 59% |
| Logistic regression | 44.1% | 51.9% |

Features selected by Weka library are then saved to the file in “data” folder so when the model is built next time, selected are features that are stored in a file.

Two models to predict changes

At this stage when the user would like to analyse a tweet that would not include any of the 99-103 selected features the model would predict the value based on a sentiment. Following feedback from the project supervisor, it was decided that there should be built another model without filtering any features that could be used when tweets features do not match with selected ones. It was decided to take the following strategy:

-if the tweet has got any feature from sifted model then use this model

-if has not got any selected features but has any of the features that is the second model built on/from then use this one

-if has not got any features matching to the ones that were extracted from the corpus then use both models and return average of results.

Deciding what to do in the case when the tweet does not have any features was hard. The only sensible solution was to just prediction basing on sentiment. In this case, it was decided to use both modules and return the average value.

## Rules learning

Once the model was working properly next thing that was done was association rules learning. It is a "rule-based machine learning method for discovering interesting relations between variables in large databases” (wiki).

The algorithm that has been used is Apriori. It is the most popular association rule learning algorithm and it was also available in one machine learning modules: Mlxtend.

The outcome of the learning process is a set of rules. Each rule consists of antecedents and consequents. Antecedents are words whose appearance is accompanied by consequents. Basically, rules tell us which words or phrases occur frequently together in a dataset.

For example, this rule tells us that if in any tweet in the dataset has got words “Dossier, Hillary, Trump campaign” it also has word “Clinton”:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Antecedents** | **Consequents** | **antecedent support** | **consequent support** | **Support** | **Confidence** | **LIft** |
| dossier, hillary, trump campaign | clinton | 0.00185014 | 0.03607771 | 0.00185014 | 1.0 | 27.7 |

Support is an indication of how frequently the words set appears in the dataset.(wiki)

If there were 1080 tweets and this words set occurred twice the support is 2/1080 =

0.00185014.

Antecedent support is TODO

Consequent support is TODO

Confidence tells us how often the rule has been found to be true(wiki). In this case, Every time those antecedents occurred in a tweet, word “Clinton” occurred as well.

Lift is a ratio of the confidence of the rule and the support of the consequents (<https://www.ibm.com/support/knowledgecenter/en/SSEPGG_9.5.0/com.ibm.im.model.doc/c_lift_in_an_association_rule.html>)

Greater lift values indicate stronger associations what can be interpreted as an importance of a rule. (source lift from wiki)

Rules are different for each currency because dataset used to build models are sifted differently. Rules presented on the webpage are filtered by a number of occurrences (at least twice). It was decided that there is no point to present a rule that occurred only once.

Rules presented on the webpage are grouped into words set. For example:

{daca, drug} => {military} and {drug, military} => {daca} are grouped into one set of words. Both of them have got the same confidence and support. That way of presenting results is more readable. By clicking on the words set, rules containing these words are presented.

****

**TODO ADD COEFFICIENT FEATURES**

## Flask webpage

TODO

# Testing

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 behavior? Are you testing the exceptional behavior, e.g. error conditions? Are you testing security issues if they are relevant to your project?

## Unit testing

Testing was carried out throughout the entire development process.

The chosen strategy was TDD (Test driven development) so while adding new functionalities, tests were written first and then was added code to fulfil those tests. Writing unit tests was a basic premise of the project because they prove the quality of the code and that it works at all. Having a set of tests is extremely helpful when modifying the code because we know that while adding one feature we do not break another. To write unit test there was used built-in python “unittest” module and “nose” to run them. Nose finds all the tests in the module and runs them in a more user-friendly way, errors are more readable. There were also necessary “mock” and “parametrized” modules to mock out and monkey patch some parts of the code and run parametrized tests to avoid code duplications.

There was a couple of exceptions in unit tests.

Scrapping code was not tested at all because the script was used just to download tweets and there was no point to test it. The code worked properly and tweets were scrapped as assumed. If the program was developed more, there was added a feature to choose any politicians’ tweets and scrapping was be done automatically, then tests would be obviously required.

Another code that was not unit tested was initially the code that processes DataFrames. In the beginning, 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. It would require some mocking, patching and writing many sample DataFrames, which structure also was changed many times at the begin.

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

Tests do not check too many exceptions that could occur during input files and data frames analysis. The program assumes that they are in a correct format. If the project was developed a bit more and allow users to add custom stock files then there would be much more tests to write and more corner cases to investigate?

Unit tests cover XX % of the code. Some of the functions were not tested because they were too simple like for example saving to file or using other libraries.

The worst THING with unittests was that they had been changed while implementing every new functionality. The structure of the program was changing and the code was refactored what lead to the TEGO ZE TESTY WYGLADLY ZUPELNEI INCZAEJ

Most of the functionalities that are not tested separately in unit tests are tested in integration tests.

## Integration/Acceptance Testing

When it was certain that smaller bits of code work then next step was to write integration tests. This is also a very important suite of tests that prove that all the bits of code will work together and the program They test the functionality of every feature.

To write integration tests was used a Behave module. It is an equivalent of Cucumber – popular testing framework among Ruby on Rails’ community. All the features are tested by writing scenarios. All the scenarios are written in Gherkin – a simple human-readable language for automated tests. Write how it works

It keeps high-level concerns separated from the code and allows non-technical people to write tests. MORE

**Scenario:** Extract words  
 **Given** we have a vocabulary of: cucumber, apple, banana, tomato  
 **When** we extract phrases from the sentence: I like to eat apples and bananas  
 **Then** we get apple, banana extracted

Functionalities that were tested are:

-Phrases extraction

-Sentiment analysis

-Stock prices prediction

Models use Scikit learn classifiers what would make testing hard and unrepeatable, therefore it was decided to mock out the models for tests. Anyway, the tests should verify the correctness of links among the bits of the project and should not tests third-party code. Mocking out with auto-speccing (mock module makes sure that the patched code has got the same interface) should be sufficient in this case.

## ****Usability testing****

Usability testing is a technique used to evaluate how easy is the interface to use. The webpage in this project is only used to present the results and only functionalities are changing currencies pages and analyzing tweets what was quickly tested by the student and supervisor.

# Critical Evaluation

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?

The aim of this project was to determine if there is any connection between Trump tweets and the markets. It was also decided to analyze the tweets dataset and find out what we can learn and what information we can gather.

The goal has been achieved, there was found a relationship between tweets and the markets. The classifier is able to predict the currency change much better than if it was doing randomly (53% accuracy to XX base accuracy).

The most coefficient features while training the model turned out to be quite sensible:

FEATURES HERE

Python was a great choice for this project. It had all the functionality required for the task, all the needed libraries were available, up to date and well-functioning. Programming in this language was very quick and allowed to do quick experiments.

If you were starting again, what would you do differently?

* Write data processing in the 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 the code was written more modular then it would be easier to test and mocking would not be so much need in some places.

Some of the tasks that were specified as additional, to do in the spare time were not accomplished because required more work to do. The program is ready to implement an option to add custom stock prices or even select from them on the webpage to be scrapped from the internet but it would require to verify the input, check a lot of corner cases and handle all of the possible exceptions. It was decided to do not even start doing this because it would require a lot of time to do it properly.

Appendices

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.

# A. Third-Party Code and Libraries

TextBlob – a free library for processing textual data.( FROM http://textblob.readthedocs.io/en/dev/index.html) Was dropped and sci-kit 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>)

Behave

Flask, ChartJS bootstrap

All those libraries were used without modification.

# B. Ethics Submission





# 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>.