



Predicting Product Category Using NLP and Machine Learning, a Case Study for e.fundamentals

Francesco Giuseppe Mascia

MSc Data Analytics

Academic Supervisor: Dr. Marc Roper

Industry Supervisor: Mihai Vaduva

University of Strathclyde

September 2019

Signed statement

Except where explicitly stated, all the work in this dissertation – including any appendices – is my own and was carried out by me during my MSc course. It has not been submitted for assessment in any other context.

Signed:

Summary

At e.fundamentals product classification plays an essential role in the system architecture, without which the company would not be able to provide its service to clients. Product classification is a popular task in machine learning projects, this thesis explores how Natural Language Processing can be used combined to Machine Learning to craft a product classification model for the e-commerce market. Different targets are taken into consideration and the most common algorithms are tested and compared using accuracy. A combination of Tf-idf and Random Forest showed to be the most accurate hence it chosen to develop the final model. Moreover, the model is implemented using Google Cloud Platform, advantages and disadvantages of this tool are outlined, caring to provide an alternative and a look at future approaches that could be furtherly used in the company.

Acknowledgments

The realization of this project would not have been possible without the help of others, I would like to take here the chance to thank all people that made this possible. Firstly I would like to thank The DataLab organisation which founded my Master and helped me find this internship opportunity and my company e.fundamentals which gave me chance to develop my own ideas and grow as an analyst. I would like to thank you my industry supervisors Mihai Vaduva and Antonio La Gamma whose technical knowledge came in help more than once through the project. A last thanks goes to my academic supervisor Dr Marc Roper for all the help and guidance for this dissertation.

Table of Contents

| | |
|--|----|
| Summary | 3 |
| Acknowledgements | 4 |
| Table of Contents | 5 |
| Project Context | 7 |
| 1. Company Background..... | 7 |
| 2. Project Background..... | 8 |
| 3. The Product..... | 9 |
| 4. Project Plan..... | 10 |
| References..... | 11 |
| Appendices..... | 11 |
| Client Report | |
| Executive Summary..... | 2 |
| Table of contents..... | 3 |
| 1. Introduction..... | 5 |
| 2. Problem Background..... | 6 |
| 3. Literature Review..... | 6 |
| 3.1 Classifying e-commerce Products by Category..... | 6 |
| 3.2 Natural Language Processing..... | 7 |
| 3.2.1Text Pre-processing..... | 8 |
| 3.2.2 Feature Engineering..... | 10 |
| 3.2.3 Vectorization and Tf-idf..... | 10 |
| 3.3. Machine Learning for Text Classification..... | 12 |
| 4. The Data..... | 14 |
| 4.1 Data Gathering..... | 14 |
| 4.2 Data Exploration..... | 16 |
| 5. Methodology..... | 17 |
| 5.1 Using Product Text to Predict Labels..... | 17 |
| 5.2 NLP-ML General Pipeline for Product Labelling..... | 17 |
| 6. The Models..... | 19 |
| 6.1 Model Structure..... | 19 |

| | |
|---------------------------------------|----|
| 6.2 The Models..... | 20 |
| 7. Validation..... | 23 |
| 8. Further Approaches..... | 25 |
| 8.1 The “Raw Text” Approach..... | 25 |
| 8.2 The “Multi Journey” Approach..... | 26 |
| 9. Implementation..... | 27 |
| 9.1 Google Cloud Platform..... | 28 |
| 9.2 Practical Use..... | 31 |
| 9.3 Further Developments..... | 32 |
| 10. Conclusion..... | 32 |
| References..... | 34 |
| Appendices.. | 36 |

Project Context

1. Company Background

E-Fundamentals is a British-built e-commerce analytics platform that provides e-commerce insights to brand owners on the effectiveness of their online retail presence, detailing key details on pricing, positioning, and promotion from real-time data. E.fundamentals reviews and gathers up to date data about products' performance online and then compiles it into the *The 8 Fundamentals* dashboard where the client can quickly action the insights that will deliver the biggest return (e.fundamentals.com). The company was born by the vision of professionals who have worked for some of the world's most renowned brands from working directly with clients to leading global, high powered teams. Their insight and expertise have led them to join forces and create an e-commerce solution that delivers a competitive advantage in today's digital landscape. Founded in London in 2014 it now has two main offices in the UK, London and Edinburgh where the product engineering office is.

The real strength of the company is its cutting-edge customised data gathering system. A Java application is used to gather a series of retailers' URL, which are predefined based on content relevance for either clients or the company itself. The process is scheduled every day, the software operates across different retailers (URLs) gathering info from the URL pages and storing screenshot, metadata and HTML for each one of them. The HTML offline page is used as a source of data that is gathered using a CSS selector which can identify different sections such as name, price, etc., these are distinguished and saved as a JSON file for each product from each single retailer. The JSON is then stored in an ElasticSearch (ES) NoSQL database that can be easily used to navigate through the raw data and acts as a basis for the final step.

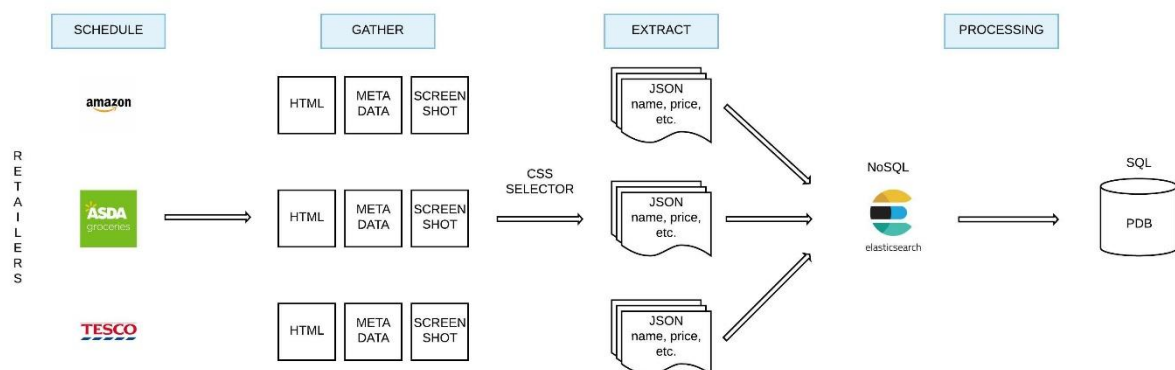


Figure 1 E.fundamentals data gathering architecture.

The ES data are raw and not yet ready to be used in the platform, each product has duplicates for all the retailers with possible inconsistencies amongst them, the data is hence parsed in the Product Database (PDB), each product undergoes a manual procedure of classification addressed to group the data into unique product units that can be univocally described, a visual representation of this architecture is given in **Figure 1**.

The company uses an internal tool to classify products in the PDB (**Appendix 1**), new products undergo a procedure of matching to existing products, those that are not matched or linked to anything in the dataset are flagged as missing products and put in the PDB review section, where an employee will manually take care of their classification.

2. The Product

The product offered by the company is an online platform. The platform organises client's data, providing an overview on major analytics and detailed categorisations, displaying insights by retailer brands, single products, etc. (**Figure 2**). The dashboard not only showcases the data but organises the actions that clients need to take, allowing them to make quick changes across the retailers and see immediate results.










| My Retailers Default ▾ | | | |
|---|----------------|---------|--|
| | RETAILER | ▲ SHARE | |
|  | Tesco | 38.1% | |
|  | ASDA | 20% | |
|  | Sainsbury's | 13.9% | |
|  | Ocado | 11.3% | |
|  | Iceland | 7.5% | |
|  | Morrisons | 6.9% | |
|  | Waitrose | 1.4% | |
|  | Amazon-Grocery | 0.8% | |
|  | Amazon-Fresh | 0% | |
| Total Revenue Share | | 100% | |

Figure 2 Total revenue by retailer (e.fundamentals.com).



Figure 3 The 8 Fundamentals (e.fundamentals.com).

This approach is defined through eight pivot points (**Figure 3**), *The 8 Fundamentals* listed below:

- *Fix the Basics*: simplifies and speeds up the work to ensure the brand standards are being met and that product listings are legal, guide the user straight to where the issues are that need to be fixed.

- *Sell the Benefits*: ensures that shoppers are presented with product benefits that provide a compelling reason to buy. If a shopper has clicked through to the product page then they are not convinced yet, the product content needs to be persuasive enough to close them.
- *Easy To Buy*: provides a shopper perspective on competitors presence in key searches for the category and considers a portfolio approach rather than just each individual product's rank.
- *Smart Pricing & Promotions*: tests promotions against strategy, provides pay for performance backup evidence and acts as an early warning indicator for potential in-store promotional issues. Breaking the pricing architecture online does not just impact the current shopping trip but also the AWOP of future baskets where the retailer has a favourites section.
- *Winning Campaigns*: measures the cumulative impact of promotions over time and assess this against your shopper objectives (e.g. trial, up-trading etc). This also allows to track all competitor promotional activity in the category and better understand promotional competitiveness.
- *Deliver Value that Counts*: assesses range against the criteria shoppers use to assess value for money. This enables brands to instantly review, compare and contrast each retailer's range to identify distribution opportunities.
- *Learn from the Shoppers Voice*: gathers learning from shopper feedback left for online retailers and crystallises key messages. It is important be on the pulse of any emerging negative trends and be able to summarise shopper feedback to support range reviews.
- *Execute your Category Vision*: the category drivers are built into a bespoke report and rolled up into a dashboard that provides an 'at a glance' assessment of each online retailer's support for the category strategy.

3. Project Background

As mentioned in the previous section the company operates a procedure of data manipulation to categorise JSON files into unique informative product data. This is done mostly through human work, a team of employees are in charge of manually compile the PDB filling the required fields, which are: *Name*, *Category*, *Sub-Category*, *Brand*, *Sub-Brand*, and *Manufacturer*. Here it lies the origin of this project, the goal is to somehow automatize, or at least speed up this procedure. Each employee is able to fill around 300 products a day, which is far not enough compared to the number of unlabelled products currently pending. In this sense the project is addressed to benefit both the company and the employees. The aim is not the one to entirely replace humans in this task, indeed it would be really difficult to reach a confidence level that allows to rely on the machine only. A certain number of employees

would still be needed to supervise the model prediction which will not be 100% accurate. Humans will correct machine's mistakes giving an additional contribute to the system.

For a while the company had this task pending and a few approaches had already been tried. Over the years the company recorded details for almost 100 thousand products, which have been carefully labelled and stored. This gold mine of data is a great source to experiment machine learning approaches to help the employees in the labelling process. The company initially used Google Auto ML (Google.com) to perform the task, this Google tool has the great advantage of being very easy and intuitive to use and can sometimes return good results as well. Product images were used to feed a Neural Network classifier, this gave a fairly good accuracy but soon proved to be expensive and most of all, to lack flexibility. Each Google Auto ML model has a daily cost which the company has to account for, moreover each training has a cost (more relevant), which makes a recurrent improvement very expansive, not to mention how time consuming an image-based model can become.

From this derived the decision to try developing a model that could be internally contained, adjustable and easy implementable in the current workflow of the company.

4. Project Plan

The first week of the project was addressed to get background knowledge about the client, their data and their needs. The focus was initially put on the discovery of patterns and correlations between sales, page rank and product attributes. The project proceed in an experimental way for the first two weeks and no clear plan was defined during this period of time. The initial work produced some minor results on page rank forecasting using product name and description, despite not being carried on it allowed to get confidence with text processing techniques, which has been fundamental for the final model development. The first two weeks served to build a solid knowledge of company complex data architecture and how it can be used it to produce the highest possible value using data analytics. From week 3 the project took a drift on product classification. The idea came from the product engineering team, the plan was to initially operate a series of tests to validate the intuition. Moreover, the project has not been conceived on a straight timeline but more as cycle of modelling and implementation for the remaining weeks, in order to get the most out of both aspects.

References

E.fundamentals.com. ef.uk.com/services/retail. [online] Available at: <https://www.ef.uk.com/services/retail> [Accessed 20 Aug. 2019].

Google.com. CloudGoogle/Vision/AutoML/Docs. [online] Available at: <https://cloud.google.com/vision/automl/docs/> [Accessed 20 Aug. 2019].

Ward, J. (2008). *Functions and Modelling. HELM Workbooks*. Loughborough University, UK. [online] Available at: https://ask.fxplus.ac.uk/tools/HELM/pages/helm_workbooks_jan2008.html [Accessed 20 Aug. 2019].

Appendices

Appendix 1

Product Database Internal tool

≡ [Engineering Hub](#) / PDB Automation

1
out of
65

Retailers

1 Journeys

Status: **AW...**

Name

PREV


NEXT

IGNORE

STORE

Name

Monster Energy The Doctor 500ml (Sugar levy applied)



Sainsbury's
7717424

Weight
500ml

Weight Value
0.5

Weight Unit: L

Department

Sub Department

Category

Sub Category

Gtin
5060335632777

Manufacturer

Brand

Sub Brand

Monster Energy

▲ COPY FROM SELECTED MATCH ▲

LINK SELECTED

● Duze opakowanie Royal Canin Club / Selection + zabawka Furry Monster gratis! - Club Pro Energy HE, 20 kg

● Monster Origin Energy Drink 4 X 500ml

● Monster Energy Ultra 500ml

● Monster Energy Ultra Sunrise 500ml

● Monster Energy The Doctor 4 X 500ml



Predicting Product Category Using NLP and Machine Learning, a Case Study for e.fundamentals

Client Report

Francesco Giuseppe Mascia

September 2019

The place of useful learning

The University of Strathclyde is a charitable body, registered in Scotland, number SC015263

THE LEADERSHIP & MANAGEMENT AWARDS 2017 | **Winner**
Workplace of the Year

THE AWARDS 2016 | **WINNER**
BUSINESS SCHOOL OF THE YEAR

THE UK Entrepreneurial University
of the Year 2013/14
UK University of the Year
2012/13

Executive Summary

The aim of this report is to provide the company with a Machine Learning model addressed to facilitate and semi-automatize product classification. Python has been used to combine Natural Language Processing and Machine Learning in order to produce a text-based classification model. Many options are tested to identify the best approach, which showed to be the mix of Tf-idf encoder and Random Forest classification algorithm. The whole project proceeded simultaneously with implementation, an aspect that guided the work to its final version. The model has been implemented using Google Cloud AI Platform, the many limitations of this beta Google tool have been an obstacle that led to consider other options, such as the use of a customised virtual machine.

Table of Contents

| | |
|--|----|
| Executive Summary | 2 |
| Table of Contents | 3 |
| 1. Introduction | 5 |
| 2. Problem Background | 6 |
| 3. Literature Review | 6 |
| 3.1 Classifying e-commerce Products by Category..... | 6 |
| 3.2 Natural Language Processing..... | 7 |
| 3.2.1 Text Pre-processing..... | 8 |
| 3.2.2 Feature Engineering..... | 10 |
| 3.2.3 Vectorization and Tf-idf..... | 10 |
| 3.3. Machine Learning for Text Classification..... | 12 |
| 4. The Data | 14 |
| 4.1 Data Gathering..... | 14 |
| 4.2 Data Exploration..... | 16 |
| 5. Methodology | 17 |
| 5.1 Using Product Text to Predict Labels..... | 17 |
| 5.2 NLP-ML General Pipeline for Product Labelling..... | 17 |
| 6. The Models | 19 |
| 6.1 Model Structure..... | 19 |
| 6.2 The Models..... | 20 |
| 7. Validation | 23 |
| 8. Further Approaches | 25 |
| 8.1 The “Raw Text” Approach..... | 25 |
| 8.2. The “Multi Journey” Approach..... | 26 |
| 9. Implementation | 27 |
| 9.1 Google Cloud Platform..... | 28 |
| 9.2 Practical Use..... | 31 |
| 9.3 Further Developments..... | 32 |
| 10. Conclusion | 32 |

| | |
|-------------------------|----|
| References | 34 |
| Appendices | 36 |

1. Introduction

Product categorization is an essential task for a company that operates in the e-commerce industry such as e.fundamentals. Every day the company manages a sizeable amount of data, the main goal of the project is to provide a useful support to the product categorization process, using tools such as Natural Language Processing (NLP) and Machine Learning (ML).

This report aims to provide a deep overview of the project, taking into account all the different angles that are necessary to provide a full understating, giving what can be considered as a written guide to its utilization. The first part of the report is intended to give an overview regarding the setting and a general project explanation, caring about the theoretical background of the relevant techniques and also about the implementation path, which will be essential for the future well-being and correct use of what it has been realised.

The project was defined starting from a lack that was present in the company. The many reasons that led to seeking an alternative approach to product categorization will be analysed, along with the pre-existing ideas on how to tackle this problem and if they affected or not the project.

A very important part of the project is the one related to the data infrastructure. What kind of data the company had to handle and how it was initially explored in order to define the modus operandi, keeping an eye on how it was gathered from the database. The methodology is later deepened, the strategy to tackle the task is defined. As already mentioned, the main purpose is to elaborate a model capable of performing an automatic product labelling. NLP and ML can be using to accomplish that, a theoretical background is provided regarding this, an essential component to consciously carry out any type of modelling, an angle of the analysis which cannot be split from the implementation.

The modelling phase itself will be then approached. First the usual pipeline for product labelling is outlined taking into account how to combine NLP and ML to deliver a model. Moreover, the general model structure is described, along with the validation and a comparison of different algorithm performances for the three targets classification. The practical implementation of the model is explored in the last bit of the report, looking at how the work done can possibly be used inside the company and what benefits it could bring to it.

2. Problem Background

Every day the company acquires a certain amount of data which gets stored in the ElasticSearch database. Over the years the company recorded details for almost 100 thousand unique products, which have been carefully labelled and stored. This gold mine of data is a great source to experiment machine learning approaches to help the employees in the labelling process. The initial approach experimented by the company was to use Google Auto ML (Google.com) to perform such a task, this Google tool has the great advantage of being very easy and intuitive to use and can sometimes return good results as well. Product images were used to feed a Neural Network classifier, this gave a fairly good accuracy but soon proved to be expensive and most of all, to lack flexibility. Each Google Auto ML model has a daily cost which the company has to account for, moreover each training has a cost (more relevant), which makes a recurrent improvement very expansive, not to mention how time consuming an image-based model can become.

From this derived the decision to try developing a model that could be internally contained, adjustable and easy implementable with the current procedures.

3. Literature Review

3.1 The Importance of Product Categorization in e-commerce

Product categorization is the task of correctly classifying a product in the related category. The exponential growth of e-commerce made this a more and more important task, becoming a must have requirement for every platform in the industry. Product classification helps marketers to focus their efforts using consumers' buying behaviour (Mack, 2019).

Any classification task can be put in practice using a variety of methods, these can be fundamentally divided in manual and automatic. In the first lie manual mapping and rule-based methods (Sundar, 2018), these approaches require a sizeable amount of human effort, they are usually quite time consuming, and almost impossible to act when dealing with Big Data. When the data at hand goes beyond what human labour can efficiently process, Machine Learning comes in help. It is indeed possible to use historic data to gather knowledge about products, categories and the connection that might or might not exist between them. This can then be used to build a model that allows to automatically classify a product in the right category (Surma, 2018).

The taxonomy of categories can follow different paths across different retailers and this is a component that majorly affects the classification strategy. Classes can be or not nested with different degrees of depth, this fact can be taken into account when tackling the problem, Vandic et al. (2018) differentiates between one-step or hierarchical approach. With a hierarchical classification two paths are available, the class can be predicted straight away or the same can be done gradually considering the hierarchy of sub-classes, each way has advantages and disadvantages that must be considered relatively to the case.

Each product has a diverse set of features, which can be used as the baseline to determine a connection between the product and the category. Product images are frequently used, as in Jain (2019), to build classification model using Neural Networks, which can in some cases achieve very good results. This approach has a major drawback in terms of manageability, not always images are available and even when they are, handling that kind of data for millions of products can be time consuming and computationally expensive. Despite it is common not to always have images available, it is very infrequent the case of not having any text related to a product, indeed a very common approach is to use product text to identify the category. A product is usually identified by a name and sometimes other textual features such as description, ingredients and size. Image-based classifiers often go through a process of labelling after which it usually follows the proper classification phase (Google.com), using the name straight away can be considered as a way to cut off this additional step. This type of data is usually more manageable than images, anyway it requires a certain amount of pre-processing to be fed to ML algorithms, this topic is known as Natural Language Processing and will be deepened in the next section.

3.2 Natural Language Processing

Natural Language Processing (NLP) can be defined as the task of analysing vast amount of natural language data, this topic is widely studied both in terms of theoretical and practical applications in Sarkar (2019), which will be the main reference for this section. A natural language is a language developed by human beings rather than artificially crafted using computer programming languages. This distance between the two worlds represents the main challenge in this field, designing tools that enable the interaction between machines and a natural human creation such as language, is called Machine Translation (Joshi, 2019). In real world, NLP is usually combined with machine learning in order to tackle real and unstructured problem. The usual pipeline is to use NLP to reshape natural language into a more machine friendly format that can be used as an input for ML algorithms. This pipeline has various steps and many methods are available.

3.2.1 Text Pre-processing

Textual data are usually generated chaotically, since they are the expression of human communication. Text pre-processing (or text wrangling), is a series of steps addressed to clean and modify text in order to facilitate the use, commonly used techniques are known as tokenization, lowercasing, stopwords and special characters removal, and stemming (Ganesan, 2019). Special characters are commonly symbols, numbers or non-alphanumeric which add extra noise in the data. The reason is that the machine does not understand that @ is not a word itself. For the same reason it might be necessary to convert each word to the same case, upper-case and lower-case letter would otherwise be considered as different letters, *Apple* and *apple* would be treated as different words, which could create unnecessary confusion during the analysis. A few simple techniques are available to take care of it, either predefined libraries or customised strings can be chosen depending on the type of text. A line from the movie *Pulp Fiction* (1994) is followingly used to give an example of how text pre-processing usually takes place, the previously mentioned techniques are on it experimented.

“And I will strike down upon thee with great vengeance and furious anger those who attempt to poison and destroy my brothers. And you will know I am the Lord when I lay my vengeance upon you”

Original text

“and i will strike down upon thee with great vengeance and furious anger those who attempt to poison and destroy my brothers and you will know i am the lord when i lay my vengeance upon you”

Lower case and special characters removal

Once the words are all on the same form, the meaning has to be considered. Not each word is equally valuable for the analysis, some are more useful than others. Stopwords are words that have little to no relevance, they are usually removed in order to reduce the corpus only to those elements that will be furtherly useful. For example, words like *of* and *the* occur many times and could then seem as the most relevant to define the text, but they aren't. Taking them away speeds up the analysis and reduces the noise. Moreover, stopwords are not the only words that carry little or no significance, depending on the case it is adequate to take care of other types of word. A product description could contain a Facebook page link, which is most likely irrelevant in terms of text analysis. A very frequently used technique is to look at the most and least common words in the corpus, words occurring in every document would not be useful to perform a classification as they don't outline any difference between the documents, at the same time words that only appear once across all documents are equally

irrelevant (Jain, 2019), this little trick can remarkably reduce the size of data to handle. Let's apply this concept to the corpus.

“and i will strike down upon thee with great vengeance and furious anger those who attempt to poison and destroy my brothers and you will know i am the lord when i lay my vengeance upon you”

Before

“strike down upon thee great vengeance furious anger attempt poison destroy brothers know lord lay vengeance upon”

Stopwords removal

Twenty stopwords (in red) have been removed, reducing the text by almost 40%. The next step in the process is called stemming. A stem is the base form of a word, stemming is the action of taking the base form of a word. The words *jumped* and *jumping* could be seen as different but in terms of text analysis they carry the same meaning and it would be more efficient to treat them as equals. Stemming comes handy in that case, reducing both to the root form *jump*. The process of stemming is actually slightly more complicated than that, there is no one single way to do it and it can be completely different among different languages, the only English language have more than one library available to perform stemming and which one to choose can result in different final outputs. A big drawback of this method is that it can sometime struggle to handle vocabular exception such as name and brands, *Tesco* could be reduced to *Tesc*, losing the meaning that the presence of the brand can have. Hence, the use of each one of this simplification techniques should be carefully considered in relation to the type of text and analysis at hand.

“strike down upon thee great vengeance furious anger attempt poison destroy brothers know lord lay vengeance upon”

Before

“strike down upon thee great vengeance furious anger attempt poison destroy brother know lord lay vengeance upon”

Stemming

In this example Porter stemmer was used (Porter, 1980), the algorithm detected four words that needed to be stemmed, it can be seen that some make sense, such as *brothers* reduced to the root form *brother*, one the other hand, *vengeance* and *furious* were stemmed such in a way that does not really produce any improvement, respectively *vengeance* and *furiou*.

3.2.2 Feature Engineering

All machine learning algorithms are limited by not being capable of understanding non-numeric inputs (Bengford, 2018). This section is about how to reshape text data to make it easily understandable by machine and hence be used as an input. Feature engineering has always had a remarkable importance when working with machine learning (Klauke, 2019), it gets even more important when working with unstructured data such as text. Despite automatization of procedures, understanding the concept behind feature engineering still lies as a key factor to properly carry on text analysis, the more you understand the more you can act on it to improve your models.

A variety of methods is available to perform feature engineering on text data, from the simplest Bag-of-Words, to more sophisticated ones such as word-2-vec. Word-2-vec is essentially an unsupervised model that can take in massive textual corpora, create a vocabulary of possible words, and generate dense word embeddings for each word in the vector space representing that vocabulary (Sarkar, 2019). Moreover, Tf-idf is a very powerful method based on relative word frequency, a wide explanation of it is given in the following section.

3.2.3 Vectorization and Tf-idf

To understand how feature engineering is done on textual data, it is important to first go through the concept of vector space model. It can be defined as a mathematical representation of text documents, each document is represented as a word vector and each term is a vector dimension. Let's consider a document D in a vectoral space VS , the number of dimensions (columns) will be equal to the number of unique terms in the totality of document corpus. VS can be formalized as:

$$VS = \{w_1, w_2, \dots, w_n\}$$

where n is the vocabulary size. Moreover, we describe D as:

$$D = \{w_{D1}, w_{D2}, \dots, w_{Dn}\}$$

where w_{Dn} identifies the weight of word n in document D , it is a numeric value that can have different meanings depending on the type of encoding used.

The first vectorization approach is usually referred to as Bag-of-Words, indeed the words are grouped as a vector, using the form of sparse matrix, which gives back the term frequency over document D . Despite the advantage of simplicity this method can sometime cause some issues. The absolute frequency of terms can result misleading, words that occur in every

document could overshadow the importance of others, that being only rarely present, carry a specific meaning and high identity value.

In this regard, Tf-idf can be the solution, it stands for Term Frequency Inverse Document Frequency and is defined as the product of the two metrics tf and idf (Sammur and Webb, 2011):

$$tfidf = tf \times idf$$

Term Frequency is simply the raw frequency of a word in the document and can be expressed as:

$$f(w, D) = f_{wD}$$

Inverse document frequency denoted by idf is the inverse of the document frequency for each term, it is computed dividing the total number of documents in the corpus by the document frequency for each term and then applying logarithmic scaling to the result.

$$idf(w, D) = 1 + \log \frac{N}{1 + df(w)}$$

The final Tf-idf is the normalized product of tf and idf, using the formula:

$$tfidf = \frac{tfidf}{||tfidf||}$$

where the denominator is the Euclidean L2 norm of Tf-idf.

A good way to get a clear picture on how this method works in practice, an application example is following provided. A corpus of three product names is used:

- 1 - 'Original Gourmet Jelly Beans, 1.8kg' → 'original gourmet jelly beans kg'
- 2 - 'Lighter Frozen Home Chips Straight 900g' → 'lighter frozen home chips straight g'
- 3 - 'Bounce Breakfast High Fibre Protein Bar' → 'bounce breakfast high fibre protein bar'

The text in the corpus gets processed in advance, using the techniques showed in the previous section. Bag-of-Words method is then used to produce a frequency matrix which can be seen in **Table 1**.

| | bar | beans | bounce | breakfast | chips | fibre | frozen | gourmet | high | home | jelly | kg | lighter | original | protein | straight |
|---|-----|-------|--------|-----------|-------|-------|--------|---------|------|------|-------|----|---------|----------|---------|----------|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

Table 1 Word vector vocabulary representation.

The three names have been grouped into a single vocabulary of words (columns), the three rows are the single instances (the three products) represented through the absolute frequency. **Table 2** below shows the result after using Tf-idf. Each row is now a vector of numbers that are relative frequencies.

| | bar | beans | bounce | breakfast | chips | fibre | frozen | gourmet | high | home | jelly | kg | lighter | original | protein | straight |
|---|------|-------|--------|-----------|-------|-------|--------|---------|------|------|-------|------|---------|----------|---------|----------|
| 0 | 0.00 | 0.45 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.45 | 0.00 | 0.00 | 0.45 | 0.45 | 0.00 | 0.45 | 0.00 | 0.00 |
| 1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.45 | 0.00 | 0.45 | 0.00 | 0.00 | 0.45 | 0.00 | 0.00 | 0.45 | 0.00 | 0.00 | 0.45 |
| 2 | 0.41 | 0.00 | 0.41 | 0.41 | 0.00 | 0.41 | 0.00 | 0.00 | 0.41 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.41 | 0.00 |

Table 2 Tf-idf vector vocabulary representation.

3.3 Machine Learning for Text Classification

This section is a review on how to use machine learning for text classification purposes in a real-life context, the previously described vectorization approach will be integrated in the classification analysis workflow. A comprehensive view of the topic is provided in Bengfort *et al* (2018) which will be the main reference for the section.

Classification is one of the recurrent problems in the world of text analysis, it is used in a wide variety of real-life scenarios, such as recommendation systems, document topic, sentiment analysis and product categorization, the main topic of this thesis. This concept works on a very simple premise, given a target variable, such as category, the aim is to identify patterns that can eventually exist among instances, considering independent variables and their relationship with the target.

This type of classification is called supervised machine learning, the target is known for each instance in the dataset and the model is trained via examples, formulating a structure that allows to minimize the error on the basis of the observed examples (Géron, 2017). The model aims to be a general picture that can be then used to predict unknown targets from new instances, using the mechanism identified during the training procedure.

A wide variety of methods is available to perform this kind of analysis, all of them are fundamentally based on the same workflow schema. The text corpus is first processed using the vector transformation technique, after that the numeric data are fed to a classification algorithm. Regarding the algorithm to use, the options are several, which one to choose is usually difficult to say in advance. The common procedure is to use validation in order to compare the different options. The data is split into train and test set, the train set is used to

train the model, which is then tested on the test set, giving back an accuracy score that can be used to compare the goodness of different approaches.

Shaikh (2017) and Li (2018) propose a comparison of different commonly used algorithm in this field. These listed below proved to get a good accuracy on text-based analysis, they are all very intuitive and don't require deep technical knowledge to be used. All of them are widely documented and implemented in most Machine Learning libraries (i.e. Sklearn).

- Logistic Regression: it is a linear model that makes predictions by computing a weighted sum of the input features, plus a constant bias term. Instead of outputting the result directly, logistic regression outputs the logistic of that result which addresses a result to a class (Hastie *et al*, 2009).
- Naïve Bayes Classifier: it is an algorithm that utilizes Bayes's Theorem, it is based on the assumption that the variables are conditionally independent. It provides a mechanism to compute the conditional probabilities of occurrence of two events based on the probabilities of occurrence of each individual event. Due to its computational efficiency it is often used in practice (Webb, 2011).
- Support Vector Machines: it works by finding a hyperplane that divides a space into two subspaces of data with as wide margin as possible. SVM is particularly well suited for classification of complex but small-medium sized data, it usually provides a good generalization accuracy and it is very popular for text classification tasks (Zhang, 2011).
- Random Forest: it is an ensemble learning technique that uses decision trees as the base classifier (Breiman, 2001). Many works have been done regarding its use for text classification, such as in Marshall (2018) where the algorithm was used to carry out an analysis of American presidents or in Xu *et al.* (2012) which provides an attempt of improving the algorithm specifically for text classification.

4. The Data

The company operates using two databases, a noSQL database based on ElasticSearch (ES) and an SQL one, which is called Product Database (PDB). The first step is to obtain a clean dataset that can be easily used to carry out the project, using it as an input for the classification model training. In this regard the communication with the engineering team was fundamental.

4.1 Data Gathering

It has been important to define a priori the type of information that was necessary for the project, more specifically the target information along with the one to be used to predict it. The project is addressed to develop a labelling system on multiple targets: sub-category, sub-brand and manufacturer. These three labels will be used to develop three different classification models. Regarding the predictor variables, the model will be text-based, given the little manageability of an image-based one. Textual data, such as *name* and *description* are the only ones (apart from the image) that can possibly carry some meaningful information to be used for a classification model.

Most of the times it is a good idea to do different tests and experiments in order to identify the best combination of techniques, for this reason the original dataset will not be strictly bounded to a predefined direction, many options are considered but not all of them will necessarily be used during the modelling phase.

The data is usually gathered by merging data from multiple sources, ElasticSearch and PDB. The PDB is the source where data is recorded by hand using the correct labels, these will be the target used to train the model. Anyway, the predictors must be a known information, hence they should be taken from the “original” dataset (ES), those are going to be the textual variables to be used to predict the target.

The two datasets have one variable in common, the *ProductID*, which can be used to perform a merge of the two. Products on ES can have a certain amount of uncertainty based on the presence of many different retailers, the same product can have slightly different names coming from different retailers, hence the same product would appear many times with slightly different names. This would be a problem in the modelling phase, product names have to be unique for a model to be reliable and non-biased. For this reason, the name has been taken from the PDB where products are univocally recorded. This will not affect the result, names in the PDB are almost the same in ES but they simply are univocally grouped in order to have unique products. The final output is the dataset that will be used to train the model, the

variables are as follows.

Variables from ElasticSearch:

- *description*: description of the product;
- *journey*: site journey for the product, comparable to department (Baking, Beer, etc.);
- *es_manufacturer*: manufacturer reported in ES;
- *es_brand*: brand reported in ES;
- *netQuantity*: size of the packaging.

Variables from Product Database:

- *name*: unique product name;
- *department*;
- *subDepartment*;
- *category*;
- *subCategory*, the three above are univocally related to subcategory, hence it is enough to predict this to also predict department, sub-department and category;
- *brand*;
- *subBrand*: as for sub-category, the sub-brand allows to automatically know the brand;
- *manufacturer*.

A snapshot of the raw data is given in **Table 3**.

| name | department | subDepartment | category | subCategory | manufacturer | brand | subBrand | journey | es_manufacturer | es_brand | netQuantity |
|---|-----------------|---------------|-------------------------|------------------------------------|---------------------|----------|----------|---------|-----------------|----------|-------------|
| Haribo Fried Eggs Bulk Bag 3kg | Ambient Food | NaN | Fruity Confectionery | Fruity Confectionery Sharing | Dunhills | Haribo | Haribo | Sweets | NaN | NaN | 3 |
| Nylabone PuppyBone - Regular | Petcare | NaN | Treats | Dental Chews | Nylabone | Nylabone | Nylabone | Pet | NaN | NaN | 0.3 |
| KONG Snacks Liver Large 312g | Petcare | NaN | Treats | Dog Treats and Biscuits | Kong Company Ltd | Kong | Kong | Pet | NaN | NaN | 0.3 |

Table 3 Raw data example.

Product name and the variables from ES will be used as predictors, while *subCategory*, *subBrand* and *manufacturer* from the PDB will be the targets to predict. Amongst all these, only a few will prove to be useful. A few different approaches are tested in section 8, the best one will be selected to craft the final model.

4.2 Data Exploration

Before any analysis it is needed to have an overview of the data structure and composition. The original dataframe has a length of 97650, there is the chance that some of them are non-unique, this could create bias in the model, hence it is necessary to check it, the major identifier of products can be considered the name, removing *name* duplicates the size is reduced to 96629, only 1021 duplicate products were found. Each variable on ES could contain a certain amount of null values, **Figure 1** displays it using a stacked bar chart (name was also included for clearance).

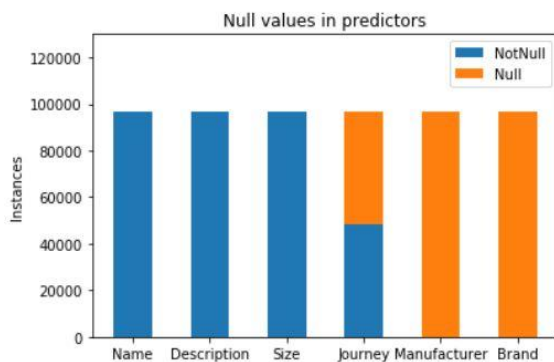


Figure 1 Stacked bar chart of null values in predictors.

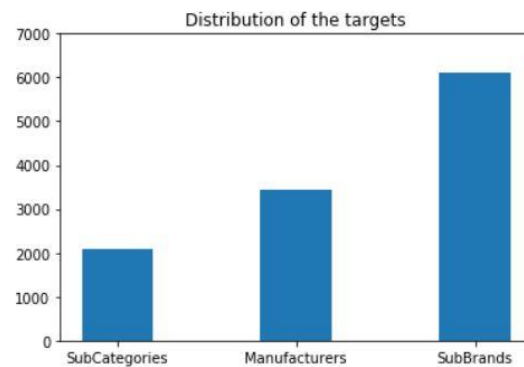


Figure 2 Number of classes in the three target variables.

It can be seen that the only variables with null values are *Journey*, *Manufacturer* and *Brand*. The product's *Journey* is not always recorded, it is only available for about a half of the data, which is still a pretty remarkable amount (almost 50 thousand products). As already known from the engineering team, manufacturer and brand are very rarely available in the ElasticSearch database and even then, they are not 100% reliable as they could slightly differ from those recorded in the PDB. As it emerges in **Figure 1** literally all of them are null values, further investigations showed that these two variables are only occasionally recorded for foreign retailers and never for English retailers, that is why none of them shows up in this dataset, these two columns will then be totally ignored from now on.

The targets have a structure of multiclass variables, the number of classes is different in each of them as it can be seen in **Figure 2**. The sub-categories are 2102, the manufacturers are 3431, and the sub-brands are a total of 6116, by far the most heterogeneous one.

One more notable piece of information regards the size of the different classes inside each target. Among the three there is a wide variety in terms of size, some classes like the *Pet* sub-category has over 20 thousand products, while *Chocolate Premium* only has 20. A rough idea about how size varies across the classes is given in **Table 4**. Figures are very similar for all three variables, sub-categories are slightly more homogeneous than the other two. It does not

really surprise that the majority of classes belongs to the type *Small* (100 products or less), only less than 1% of the total number of classes had more than 1000 products.

| | Small (0,100) | Medium (101,1000) | Large > 1000 | Total |
|--------------|------------------|----------------------|-----------------|-------|
| Sub-Category | 1954 | 134 | 14 | 2102 |
| Manufacturer | 3303 | 110 | 15 | 3431 |
| Sub-Brand | 6016 | 91 | 8 | 6116 |

Table 4 Classes size in the three target variables.

5. Methodology

5.1 Using Product Text to Predict Labels

Predicting product labels in an automatic way can enormously facilitate the task of categorization done at the company, improving speed and efficiency. As already said in section 2, this process can be done following different approaches, such as using image-based models, already tested with poor results, or manually, the currently used method. A common way to tackle this problem in product categorization is to consider textual data related to products. During this step, communication and confrontation with the Engineering team was essential. Taking into account all the previous information available, it has been assumed that a product text may contain information that is almost always enough to define its collocation in terms of the target variables considered (*subCategory*, *manufacturer*, and *subBrand*).

This one is indeed the approach followed in the project. The use of textual data hides many different challenges, hence it is particularly important to pay attention to the theoretical background provided in the section 3.

5.2 NLP-ML General Pipeline for Product Labelling

This section is intended to explain in a broad way how to proceed, giving a procedural baseline for the single models that will follow later in this section. The whole analysis is carried out using Python language with Spyder and Jupyter Notebooks, the main libraries used are Nltk (Looper and Bird, 2002) for text processing and Sklearn (Pedregosa *et al.*, 2011) for machine learning.

The first step is to consider possible data manipulation operations to be done before the modelling. Looking at classes' distribution in **Table 3**, it can be noticed how they are heavily unbalanced in size. A considerable amount of them have a very small number of products, which would not let the algorithm able to distinguish relevant features, it has then been decided to drop all classes with a population smaller than twenty products. This way the algorithm will have to work on a lower number of classes and it is less likely to get confused among small and sparse classes.

Furthermore, it is needed to perform a text cleaning to reduce noise in data and optimize its usage. As per section 3, this phase is built of many steps. The first step is usually tokenization, each name is split into tokens, each token is a single word that gets considered as a separate entity from others. After tokenization the function `lower()` can be used to standardise the case among all words, so that upper cases are not confused for different words. Now it is time to remove those tokens that do not carry any useful information for the categorization, these are special characters, punctuation and stopwords (see 3.2), English vocabulary is used in this regard. The last important operation is stemming, `PorterStemmer` library is used. This procedure can be all wrapped up in a single customised function `clean_text()` (see 6) that has the raw text as in input and give the processed text as output.

Once the text is cleansed to its simplest version, it is time to encode it, the Tf-idf method is used. The text encoding process can be considered as a prior modelling phase, it is addressed to produce an input for the classification model. At this point it feels appropriate to point out a very important detail that can often be overlooked. If the model is going to be used in the future to classify new data, it is essential that this new data undergoes the exact same Tf-idf encoding used for the training. When performing the encoding on the training data, it is necessary to save the vocabulary of the vector generated by the `CountVectorizer()` function, which represents the vector structure that is needed to fit new data in the future. This way the new data will not be re-encoded from scratch, it will be embedded in the same vector structure produced by the training data. The first approach was to save on disk two different models .joblib, the text vector vocabulary and the machine learning model, this structure has been slightly modified further on in the project for implementation related reasons, which are better explained in the following section.

After the input is ready for the algorithm, it's time to consider which method to use for the classification. This usually cannot be known a priori, it is necessary to test different algorithms using some type of validation techniques to compare their results. In order to perform the validation, the dataset is split in two subsets, train data and test data, the algorithm is trained on train data and later tested on test data. So, the model is being used to predict "unknown"

data, anyway the real class of these instances is known, so the information can be used to measure the accuracy of the prediction, and then compare the performance of the algorithms based on their prediction accuracy.

The most complete way to evaluate an algorithm performance is by looking at the classification report produced. An example is provided in **Table 5**, the three metrics used are precision, recall and f1: precision is the fraction of items classified in a right class, recall is the fraction of items that are in the class the classifier detects, f1 is a metric that combines the two.

| | precision | recall | f1-score | support |
|------------------|-----------|--------|----------|---------|
| Ale | 0.77 | 0.93 | 0.84 | 60 |
| Belgian Beer | 0.33 | 0.33 | 0.33 | 3 |
| Cider | 0.80 | 0.88 | 0.84 | 32 |
| Craft Beer | 0.75 | 0.55 | 0.63 | 33 |
| Fruit Cider | 0.73 | 0.53 | 0.62 | 15 |
| Lager | 0.82 | 0.94 | 0.88 | 64 |
| No Alcohol Lager | 1.00 | 0.12 | 0.22 | 8 |
| Stout | 1.00 | 0.50 | 0.67 | 10 |

Table 5 Classification report example.

It has to be taken into account that unlike the example, the target classes in this model are thousands, it would result difficult to visualize and analyse such a sizeable matrix. The metric used will simply be an accuracy score averaged for all classes, more precisely it is called Jaccard index and is defined as the size of the intersection divided by the size of the union of two label sets (Scikit-learn.org, 2019).

6. The Model

6.1 Model Structure

This section looks at the importance of keeping an eye on the implementation from the very beginning of the modelling phase. The aim of the model is to be practically utilized by the company for automatic product classification. The mode of how this will take place has to be considered since the very beginning of the project and it has to influence the way the model is conceived, with special attention on the technical aspect (i.e. software, format, size), which has to match what will be required during the implementation phase.

The idea of implementation heavily affected the modelling phase in this case. Google Cloud Platform is used as a base for implementation, this Google tool is the most common way to keep ML models in cloud storage and connect them to external environments.

This platform presents a few limitations that guided the model into a particular layout which could be compatible with this implementation method. A first limitation is that only one single model can be used to make predictions. The initial approach was to have two sub-models, text encoder and product classifier, this proved not to be possible as in contrast to the future eventuality of an implementation on Google Cloud Platform. The solution was to ideate a pipeline that incorporates all the steps into one single model. The pipeline is the model itself, it is saved on disk and when using it to predict new files the raw text is given as input and the model takes care of each step, from text cleaning to algorithm fit, section 5.2 explains it into more detail.

The most troubling limitation is probably the one related to the model size. The platform can only work with models of 250Mb or smaller, many methods have been tried to get around this boundary. This showed to be a particularly strict limit for a machine learning model and it has been the main obstacle during the implementation phase (section 7).

6.2 The Models

The sub-category classification represented the true incipit of this project, to predict sub-category was the first task coming from the management. The first test on sub-category classification returned good results, something that gave enough confidence to expand the model to new targets, sub-brand and manufacturer.

Three different versions are developed of the model using Python: Sub-category, Sub-brand, and Manufacturer. One single script is created to predict these three targets based on product name, the code is exactly the same for the three version, the only thing that changes is target, which gets defined outside the code's main corpus. The model is structured using functions, the concept of parsimony was followed during the design, cutting off the non-essential details to keep the architecture lean and simple, *"among competing models, all of which provide an adequate fit for a set of data, the one with the fewest parameters is to be preferred"* (Everitt and Skronidal, 2010).

The code script aims to be as general as possible, the model refers to predictor and target, not specifying what they are. The three version are produced running the code each time defining the different target. The predictor is the same (*name*) for all of them, anyway it has been helpful to follow the same procedure to test initial approaches that included product size and/or

description, these were soon discarded as they slowed down the procedure giving no visible improvements in accuracy. This structure keeps an eye on future adjustments, which can be facilitated by externally defined parameters. If new useful data will be gathered in the future, they could be very easily tested as a predictor in the model.

```
def get_data(df):
    df['class_size'] = df.groupby(target)[target].transform('count')
    df = df.loc[df['class_size'] >= 20]
    return df
```

The `get_data()` function is first used to import the data and operate the first manipulation, the classes with less than 20 products are dropped in order to reduce noise and help the classification to perform the best. Two customised functions are defined before the model pipeline, displayed below, `clean_text()` is a text cleaning function that groups all the steps treated in section 4 into one job (Stackoverflow.com, 2019).

```
def clean_text(text, ):

    def tokenize_text(text):
        return [w for s in sent_tokenize(text) for w in word_tokenize(s)]

    def remove_special_characters(text, characters=string.punctuation.replace('-', '')):
        tokens = tokenize_text(text)
        pattern = re.compile('[{}]'.format(re.escape(characters)))
        return ' '.join(filter(None, [pattern.sub('', t) for t in tokens]))

    def remove_stopwords(text, stop_words=default_stopwords):
        tokens = [w for w in tokenize_text(text) if w not in stop_words]
        return ' '.join(tokens)

    def stem_text(text, stemmer=default_stemmer):
        tokens = tokenize_text(text)
        return ' '.join([stemmer.stem(t) for t in tokens])

    text = text.lower() # lowercase
    text = remove_special_characters(text) # remove punctuation and symbols
    text = remove_stopwords(text) # remove stopwords
    text = stem_text(text) # stemming
```

This code reduces to a single customised function the many steps of text cleaning. The problem is now that customised functions cannot be used into a scikit-learn's pipeline. In a scikit-learn's pipeline every step has to look like a sklearn transformer. Everything that goes into a pipeline has to implement both `fit()` and `transform()` methods. The built-in `FunctionTransformer` does this handily, the solution is to wrap the custom functions in a function that creates a list comprehension that applies the custom function to the series passed in, then wraps that in a `FunctionTransformer` (Cranfil, 2019).


```
def pipelineize(function, active=True):
    def list_comprehend_a_function(list_or_series, active=True):
        if active:
            return [function(i) for i in list_or_series]
        else:
            return list_or_series
    return FunctionTransformer(list_comprehend_a_function, validate=False, kw_args={'active':active})
```

The model is developed using two pipelines, the first one called `text_pipe()` takes care of text processing. Here the `pipelineize()` function is used, in order to include the `clean_text()` customised function, in addition to that, the text is vectorized and transformed using Tf-idf, this step is ideated to input raw text and output a machine friendly Tf-idf vector. The `text_pipe()` pipeline is used inside a second sklearn pipeline, this can be considered as the main model and it works in two steps a first step of encoding (`text_pipe()`) and the sklearn machine learning algorithm in the second step.

As already mentioned, the pipeline is very generic, the algorithm is not yet defined, it is identified externally as well as target and predictor, this way the same code can be easily used to test different algorithm during the validation phase. In next section the same piece of code is used for all three models, comparing the results of different types of algorithm.

```
def model(df):
    vectorizer = CountVectorizer(decode_error="replace")
    tfidf_transformer = TfidfTransformer(use_idf=True)
    text_pipe = Pipeline([("clean", pipelineize(clean_text)), ("encoder", vectorizer), ("transform", tfidf_transformer)
    ])
    train_predictor, test_predictor, train_target, test_target = train_test_split(df, df[target], test_size = 0.2, random_state = 42)
    mod = Pipeline([("encoder", text_pipe),
    ("model", model),
    ])
    mod_fit = mod.fit(train_predictor[predictor], train_target)
    y_pred = mod_fit.predict(test_predictor[predictor])
    return accuracy_score(y_pred, test_target)

def process():
    data = get_data(df)
    mymodel = model(data)
    return mymodel

def main():
    final = process()
    print(final)

main()
```

The code here explained is addressed to test different paths, indeed it includes a step of data split, the 20% of data is taken out in order to perform validation. Once the best path is identified the final training can be done, this time using the full dataset. The two functions `get_data()` and `model()` are wrapped in `process()` and `main()` for functionality.

7. Validation

In this section the different methods are tested to compare the results and define which path is the best choice to further develop the final model. The first thing considered is the choice of predictor variables. It is already consciously assumed that a product text can usually be informative about the three targets at hand. The product text available includes name, description and size.

This phase started with the preconceived idea that the name alone can be enough to predict the target, and that it would most likely be the best choice in terms of running time and efficiency, as including too much text in the model could end up in an unnecessarily bloating. Anyway, this is a relatively unknown territory, so a test is necessary to confirm that assumption. A sample of data was taken to try predicting the sub-category using a simple linear regression. The name alone was used first, resulting in a pretty good accuracy, the description was later added, instantly proving not to be a good path. On average, product name has a length of about 8 words, product description is most of the times above 40 words and usually loaded with negligible information. In addition to a crazy long running time, this approach did not lead to any relevant improvement in accuracy, hence it was promptly passed over. The size variable was initially considered in relation to sub-category prediction as some of them are indeed related to product size (i.e. *Chocolate Premium Pack XXL*), anyway this did not bring any increase in accuracy. The initial assumption was confirmed and name will be the only data used as a predictor.

At this point, a few additional experiments were done with text, a very frequently used technique is to look at the most and least common words in the corpus, words occurring in every document would not be useful to perform a classification as they don't outline any difference between the documents, at the same time words that only appear once across all documents are equally irrelevant (Jain, 2019). Once again, this additional step was not successful, despite bringing a minor advantage in terms of running time, it would be rather difficult to include inside a sklearn pipeline, hence the simplest alternative was preferred.

Five different algorithms were chosen to be validated, this choice was partially dictated by the usual procedure in this type of analysis as in Shaikh (2017), these five are rather simple but yet effective algorithms, moreover all of them are "implementation friendly" as all included in Sklearn framework, a factor that will turn out to be important as explained in next section. The five alternatives are listed below along with a very brief explanation for each one:

- Logistic Regression: it is a linear model that makes predictions by computing a weighted sum of the input features, plus a constant bias term (Hastie *et al*, 2009).
- Naïve Bayes Classifier: a classification algorithm that utilizes Bayes's Theorem (Webb,

2011).

- Support Vector Machines (SVM): it works by finding a hyperplane that divides a space into two subspaces of data with as wide margin as possible (Zhang, 2011).
- Random Forest: it is an ensemble learning technique that uses decision trees as the base classifier (Breiman, 2001), this algorithm is tested in two different versions, 10 and 100 estimators.

As mentioned in section 5.2 the accuracy score is used to compare the performance. In **Table 6** the results from the validation are displayed for all the three models and all five algorithms. In all of them SVM is the one that performed the worst with an accuracy around only 40% and very long running times. Naïve Bayes performed quite poorly as well, on the opposite hand Linear Regression and Random Forest achieved always great results. Accuracy is very similar among these last three alternatives, what really makes a difference in this case is running time, the simplest Random Forest (10 estimators) is the one with by far the tiniest running time, around a half compared to Linear Regression.

| | Linear Regression | Multinomial NB | SVM Classifier | Random Forest 10 | Random Forest 100 |
|---------------|----------------------|---------------------|----------------------|---------------------|----------------------|
| Sub- Category | 77.32% (> 10mins) | 54.95% (< 5mins) | 35.66% (> 15mins) | 76.61% (< 5mins) | 80.91% (> 10mins) |
| Sub-Brand | 91.29% (> 10mins) | 36.49% (< 5mins) | 39.94% (> 15mins) | 92.63% (< 5mins) | 94.55% (> 10mins) |
| Manufacturer | 95.13% (5mins) | 51.69% (< 5mins) | 47.34% (> 15mins) | 95.76% (< 5mins) | 97.16% (> 10mins) |

Table 6 Algorithm performance comparison on validation set.

Another important factor worth mentioning is model size, as highlighted many times this model was thought with a view to implementation, despite the more complex Random Forest (100 estimators) has a slightly better accuracy, it also has ten times more estimators, which means a remarkably bigger size, longer saving time, loading time and prediction time, even a few seconds per prediction could be relevant when dealing with thousands of products, hence a few percentage points in accuracy are traded for a considerably higher manageability.

8. Further Approaches

8.1 The “Raw Text” Approach

The model has been developed using the standard procedure for text processing, which can be generally defined as the correct way to approach any type of text analysis. Anyway the action must also consider the nature of the data at hand in order to accordingly adjust the standard procedure.

In the case at hand the text can thought be considered slightly unusual, the product name in use for the train is not chaotic text taken from retailers’ websites, as mentioned in section 3.1 they are unique names inputted manually in the PDB. For this reason it can be assumed they only contain information useful to the cause. This may impact the choice of performing text cleaning, which based on this assumption might not be necessary. Brands and names play a big role in carrying product information and it is known that lowercasing and stemming are not ideal in that case. A brand such as *Box* would be lowercased to *box* and be then confused for a normal word, moreover, the brand *m&m* would be affected by special characters removal and be reduced to “*m m*” losing its true meaning.

For this reason it may be worth to try on testing the model not including text cleaning at the beginning. The same model from 6 (RF10 algorithm) was tested simply removing `text_clean()` from the pipeline, the accuracy results are shown in **Table 5** below.

| | Sub-Category | Sub-Brand | Manufacturer |
|----------|--------------|-----------|--------------|
| Accuracy | 76.88% | 96.07% | 92.80% |

Table 7 “Raw Text” model validation accuracy.

The validation surprisingly confirmed the assumption, the accuracy is slightly higher for all the three models, the raw product names can hence be considered more informative in this case. In addition to improving accuracy this new approach carries the advantage of simplicity, the less steps a model includes the faster it is and less bugs could occur along the way. Moreover, the use of a custom function for text cleaning could result in troubles in the implementation, which is usually easier when using pre-packaged function only.

8.2 The “Multi Journey” Approach

An alternative original approach was attempted to pursue two aims, improve accuracy and facilitate implementation. Apart from product name and the other data initially considered, the ES database has available one attribute that is strictly related to product classification. This attribute is called *journey*, it is an information about the taxonomy of website from which products are taken, it can be somehow compared to a very general category. What brought to consider this path was the extremely high number of classes in the target variables, despite this the initial model achieved a surprisingly good accuracy, managing to distinguish more than six thousand different sub-brands. Anyhow, dividing the data in order to produce smaller and more specific models could possibly help to improve the classification even more.

The *journey* variable is only available for about half the products, precisely 48396. The variable contains 70 classes of very heterogeneous size, from a maximum of 15292 to a minimum of 8 products per journey, only those with a population above 50 are considered, this led to drop 22 journeys, only 48 are used.

The same structure from the previous section is used to test this approach, the only difference lies in `get_data()`, this function is now given `product_type` as an entry parameter, where `product_type=Journey`. This way the code runs for each subset of data grouped on *journey*. The accuracy gets calculated for each subset using the algorithm chosen in section 6 (Random Forest with 10 estimators). There is a considerable variety of performance across different journeys, a quick glance is given in **Table 8**.

| Journey | Size | Sub-Category | Sub-Brand | Manufacturer |
|------------------------------|-------|--------------|-----------|--------------|
| Pet | 15292 | 0.90389 | 0.926773 | 0.972213 |
| Frozen-Meat & Poultry | 436 | 0.568182 | 0.727273 | 0.909091 |
| Beer | 1199 | 0.725 | 0.666667 | 0.641667 |
| Chilled Ready To Bake Pastry | 52 | 0.909091 | 0.818182 | 1 |
| Fresh - Confectionary | 116 | 0.333333 | 0.416667 | 0.583333 |

Table 8 “Multi Journey” model validation accuracy.

Pet got the highest accuracy, it does not really surprise considering it is also the journey with the most data, on the other hand the least accurate model appeared to be *Fresh-Confectionary* (only 116 products available). However, a journey like *Chilled Ready To Bake Pastry* got a great accuracy despite being the least populated one with only 52 instances to train the model, quite better than *Beer* which has twenty times more data. The pattern is not really clear in that sense, further investigation could be required. Moreover, some journeys showed homogeneity in the prediction of the three different variables, i.e. *Pet* is above 90% for all three, while others

performed better in a field rather than another, such as *Frozen-Meat*, 90% for manufacturer but only 56% for sub-category.

The accuracies from the 48 models are averaged keeping into account the population of each subset, the complete list of accuracies for each journey subset is available in **Appendix 1**.

| | Sub-Category | Sub-Brand | Manufacturer |
|----------|--------------|-----------|--------------|
| Accuracy | 74.76% | 75.46% | 87.84% |

Table 9 “Multi Journey” average model validation accuracy.

The table above shows the weighted averages of accuracy for the three models, all of them performed quite well but still worse than the base model. Only Sub-Category model performed very close to the previous (-2%), the other two got a quite lower accuracy, 5% less for Manufacturer and a remarkable -20% for the Sub-Brand model. That is probably because journeys divide data in a way that is closely related to sub-categories but not so much to sub-brand or manufacture. Moreover, it has to be accounted that this model has only half the data available compared to the base model, which is a big disadvantage in machine learning.

9. Implementation

In a project like this one it is always essential for the model to be designed with a view to action. This model is addressed to help and act (David, 1999), it can be defined as an operational model, oriented to perform some kind of operation, in this case a classification task.

The aim was made clear from the beginning. The model development followed a path defined on the base of a future use in the company. In this regards it was once again very important the collaboration with the software engineering team.

As in section 6.2, the models correspond to a Python file that automatically operates train and test. When speaking about the use of models, it is advisable not to divide the dataset into train and test set, this way the train can be done using the whole available dataset, the more data it is used to train the better model will hopefully come out. An additional important step is model saving, it has to be saved on memory in order to be reused without having to retrain the system every time, which can be extremely inefficient, in terms of time and computational power.

The joblib library is commonly used for this task, the model is trained and saved in a Python object (i.e. `model`), after that the function `joblib.dump()` can be used to save that object on disk.

9.1 Google Cloud Platform

Implementation of the model means to elaborate a method that allows to interface it to an external environment. A common and easy way to do it is by using Google Cloud Platform, the same platform previously used for the image-based model. Apart from the predefined AI tools, Google Cloud Platform lets users able to implement their customised code in the cloud using Storage buckets. The first step is to move the training code in the cloud, this way the `.py` file performs the training online in the Google Cloud environment saving the models straight on that with no need to use disk memory. The model gets saved as a `.joblib` file in the Google Cloud Storage, which is then connected to the AI framework.

This step revealed to be the toughest part of the whole project. Google puts many limits on the use of customised models on the platform. The first is related to the framework, it only has available three: Tensorflow, XGboost and Sklearn. Sklearn Random Forest is used in the model, hence it has not been a limit on that, anyway the whole pipeline also includes the additional step of text processing, which uses Nltk library, here the first obstacle. A customised Predictor dependency can be used to get around it, this method allows to include additional frameworks such as Nltk.

In section 6.1 it has been mentioned how the double-model approach had to be reviewed to match Google requirements. The prediction framework in the platform can only work under the strict requirement of only using one single model named `model.joblib`. Because of this fact a backup plan was ideated, creating a pipeline with all steps internally included, despite a few troubles along the way this solution proved to be successful.

The most limiting obstacle encountered regarded the model size. Google has a limit of 250Mb, it cannot handle models bigger than that. This is for sure a big limitation for a text-based machine learning model, the three files on which the models are stored are all above 2Gb (almost ten times bigger), this does not surprise when thinking about the structure of the model and all the work of text-processing internally contained in the pipeline. A few solutions have been initially tested to reduce the size of the model, such as dropping superfluous words, using different models, less estimators and less data, none of them led to relevant improvements without drastically decreasing accuracy. The `joblib.dump()` function gives the user an option to compress the model file by simply including the parameter `compress=True`.

This solution does only make sense if there is then the chance to decompress the model once it gets uploaded, this would not normally be possible on Google Cloud, but it is since the customised Predictor dependency is being used, it manually loads the model and hence allows to perform some kind of operation on it, decompression in this case. Unfortunately this did not bring to a positive result, it only moved the problem further on in the procedure, the compression allows the model to be loaded, but at the moment of decompression the system incurs in a memory error once again, since the platform cannot handle the decompress model in any way, being above the memory limit.

A further approach is the one proposed in section 8.2, which is basically a way to split the full model in 48 sub-models of smaller size, which can then be stored and used in the platform. This approach works, but it presents a few major drawbacks in terms of manageability. Moreover, this is a solution that can only be considered in the short term, once again because of a platform limit, indeed there is a maximum number of 250 models allowed in the storage. The three models at hand would be stored in 48 different versions each, for a total of 143, which is below the limit for now, anyway this number will most likely increase over time. Every week the company onboards new clients and starts getting data on new market sectors, the number of journeys will soon increase above 250 and this approach will become unusable.

| Name | Default version | Region |
|-------------|-----------------|-----------|
| subCategory | Alcohol_Premix | eu-west-1 |

| VERSIONS | | EVALUATION | BETA |
|-----------------------------|---------------------------|---------------------------|------------|
| Name | Create time | Last used | Evaluation |
| Alcohol_Premix (default) | Aug 19, 2019, 11:53:15 AM | Aug 19, 2019, 11:57:33 AM | N/A |
| Asian_Cooking | Aug 19, 2019, 11:58:30 AM | | N/A |
| Baby_Food | Aug 19, 2019, 12:00:46 PM | | N/A |
| Baking | Aug 19, 2019, 12:03:19 PM | | N/A |
| Baking_Buttercream_Frosting | Aug 19, 2019, 12:02:10 PM | | N/A |
| Baking_Cake_Mixes | Aug 19, 2019, 12:02:46 PM | | N/A |
| Beer | Aug 19, 2019, 12:04:01 PM | | N/A |
| Biscuits | Aug 19, 2019, 12:04:31 PM | | N/A |
| Breakfast_Cereal | Aug 19, 2019, 12:05:16 PM | | N/A |
| Butter_Spreads_Margarine | Aug 19, 2019, 12:05:53 PM | | N/A |

Figure 3 Models on Google Cloud AI Platform.

Despite the few issues just outlined, it seems appropriate to give a demonstration of how the model can be used through Google Cloud. In **Figure 3** above a screenshot of Google AI Platform is provided, a Sub-Category model is created to contain the different versions of the

model, each one of them is created based on the joblib files present in Google Storage. Each model can be tested and used directly on the platform.

In **Figure 4** *Beer* model is used to test Google Cloud AI, the product names are given using json syntax, this tool simply uses the model to automatically give a prediction. It seems the five beers were correctly classified into Ale, Lager and Cider.

The screenshot shows the 'TEST & USE' tab of a Google Cloud AI interface. It features a section titled 'Test your model with sample input data' with a sub-instruction: 'Request an online prediction by sending your input data instances as a JSON object.' Below this is a link 'Learn how to format input data'. A text area contains a JSON object with five beer names under the 'instances' key. A 'TEST' button is positioned below the input area. The output area shows a JSON object with five predictions: three 'Ale', one 'Lager', and one 'Cider'.

PERFORMANCE EVALUATION **BETA** TEST & USE

Test your model with sample input data

Request an online prediction by sending your input data instances as a JSON object.
[Learn how to format input data](#)

```
{
  "instances": [
    "Marston's Old Empire India Pale Ale 500ml",
    "Badger Hopping Hare Ale 500Ml Bottle",
    "Bishops Finger Strong Ale 500Ml",
    "Budweiser Budvar Lager Bottle 500ml",
    "Thatchers Katy Cider 500Ml Bottle"
  ]
}
```

TEST

```
{
  "predictions": [
    "Ale",
    "Ale",
    "Ale",
    "Lager",
    "Cider"
  ]
}
```

Figure 4 Model test on Google Cloud AI.

9.2 Practical Use

The model has a great potential if successfully implemented and it could revolutionize the way the company classifies products in the Product Database. The model would take care of those instances classified as missing product, the new product's name would be used as an input to predict sub-category, sub-brand and manufacturer. This way all the field could be filled automatically, the confidence level achieved is not yet enough to trust the prediction on its own. The product would be saved in the review section in order to be checked by an employee, who will decide if it can be confirmed or if it needs to be adjusted first.

This kind of help would consistently improve and speed up the way products are classified. Assuming the validation accuracy is reliable, the employee in charge of reviewing would find the products correctly classified four out of five times, and the job would be reduced to eyeball the fields and push the *Store* button, just a few seconds compare to the current procedure of going through dozens of categories and brand, which can take up to two minutes.

The screenshot displays a product classification interface. At the top, there's a progress indicator '1 out of 65' and navigation buttons: 'PREV', 'NEXT', 'IGNORE', and 'STORE'. Below this, a search bar contains 'Monster Energy The Doctor 500ml (Sugar levy applied)'. The main area shows product details and classification results:

- Product Image:** A can of Monster Energy The Doctor 500ml.
- Weight:** 500ml
- Weight Value:** 0.5
- Weight Unit:** L
- Department:** Drinks (Correct)
- Sub Department:** Sports & Energy Drinks (Correct)
- Category:** Energy & Stimulant Drinks (Correct)
- Sub Category:** Energy & Stimulant Drinks (Correct)
- Gtin:** 5060335632777
- Manufacturer:** Monster Beverage Corp... (Correct)
- Brand:** The Doctors (Incorrect, marked with a red X)
- Sub Brand:** The Doctors (Incorrect, marked with a red X)

At the bottom, there are buttons for 'Search by Name', 'COPY FROM SELECTED MATCH', and 'LINK SELECTED'. A message at the bottom states 'No data available.'

Figure 5 Model test on Google Cloud AI.

In **Figure 5** a simulation of how the process would work is given. The model has been used to predict the fields, Sub-Category was correctly classified as *Energy & Stimulant Drinks* and consequently Category, Sub-Department and Department. The Manufacturer was identified as well but Sub-Brand was misclassified as *The Doctors* when it should actually be *Monster*. It would be here sufficient to correct the two wrong fields before storing the final version.

9.3 Future Development

The Google Cloud AI Platform provides a valid way to implement machine learning models, anyway it is only a beta version for now, so this method is not always straight forward and includes many limitations. The base model cannot currently be implemented with this method and the backup plan of using multiple smaller model cannot last for long, given the constant flow of new data in the company data warehouse. It is necessary to find a way to get around these limitations, this can be done setting up a virtual machine to perform the prediction job. As before, python script can be stored in Google Cloud (or Github) to perform automatic training and saving of the models on Google Cloud itself.

The virtual machine would use Google Kubernetes Engine to run a job that uses Github predict.py script and the Google Cloud models previously saved. This is only giving a general overview to code a way to perform a job that wraps all the steps included in Google AI Platform model but without a memory limit.

10. Conclusion

During the whole project the aim was to work in a way that could be useful to the company. The approach of using product text for classification as proposed by the management proved to be winning, having advantages in terms of manageability and flexibility. This option has been detailly explored by the usual techniques of NLP and ML, some theoretical background appeared to be important in that regard.

Sklearn library in Python was chosen to develop the model, being the most adequate framework for this type of task as it is implementation friendly (supported in Google Cloud Platform) and it includes all the most commonly used algorithm for this type of analysis.

Some initials tests confirmed the assumption of product name being the best predictor for the three targets, sub-category, sub-brand and manufacturer. Product name was hence used as the only predictor variable. Random Forest showed to be the best performing algorithm, the ten trees ensemble scored an accuracy between 76% and 97%, depending on the target.

Implementation revealed to be the most troubling step. It affected the layout of the model which has been adjusted on the basis of that. Google Cloud Platform was chosen as an initial approach to model implementation, the model was split into smaller parts in order to fit the memory limit of the platform, a short-term solution that will not be sustainable in future.

Moreover, a long-term solution is proposed, Google Kubernetes can be used to set a virtual machine and be able to act with less strict memory limits.

In conclusion, machine learning proved to be a very useful tool that can concretely help the company in the product classification procedure. If correctly implemented in the company work flow the model could save the company up to 80% of time during the classification procedure.

References

Bengfort, B., Bilbro, R. and Ojeda, T. (2018). *Applied Text Analysis with Python*. 1st ed. Sebastopol, US. O'Reilly Media, Inc.

Breiman, L. (2001). *Random Forest*. [online] Available at: <https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf> [Accessed 20 Aug. 2019].

Cranfil, R. (2016). *Building a Sentiment Analysis Pipeline in scikit-learn Part 3: Adding a Custom Function for Preprocessing Text*. Ryan-Cranfield.github.io. [online] Available at: <https://ryan-cranfill.github.io/sentiment-pipeline-sklearn-3/> [Accessed 20 Aug. 2019].

David, A. (1999). Model implementation: A state of the art. *European Journal of Operational Research*, 134(2001), pp.459-480.

E.fundamentals.com. ef.uk.com/services/retail. [online] Available at: <https://www.ef.uk.com/services/retail> [Accessed 20 Aug. 2019].

Everitt, B. and Skrondal, A. (2010). *The Cambridge Dictionary of Statistics*. 4th ed. Cambridge, UK. Cambridge University Press. pp.317.

Ganesan, K. (2019). *All you need to know about text pre-processing for NLP and Machine Learning*. Kdnuggets.com. [online] Available at: <https://www.kdnuggets.com/2019/04/text-preprocessing-nlp-machine-learning.html> [Accessed 20 Aug. 2019].

Géron A. (2017). *Hands-On Machine Learning with Scikit-Learn & Tensorflow*. 5th ed. Sebastopol, US. O'Reilly Media, Inc.

Google.com. CloudGoogle/Vision/AutoML/Docs. [online] Available at: <https://cloud.google.com/vision/automl/docs/> [Accessed 20 Aug. 2019].

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning*. 2nd ed. New York, US. Springer.

Jain, P. (2019). *Image Classification for E-Commerce - Part I*. TowardsDataScience.com. [online] Available at: <https://towardsdatascience.com/product-image-classification-with-deep-learning-part-i-5bc4e8dccc41> [Accessed 20 Aug. 2019].

Joshi, P. (2019). *A Must-Read NLP Tutorial on Neural Machine Translation - The Technique Powering Google Translate*. Analyticsvidhya.com. [online] Available at: <https://www.analyticsvidhya.com/blog/2019/01/neural-machine-translation-keras/> [Accessed 20 Aug. 2019].

Klauke, P. (2019). *Importance of Feature Engineering methods*. TowardsDataScience.com. [online] Available at: <https://towardsdatascience.com/importance-of-feature-engineering-methods-73e4c41ae5a3> [Accessed 20 Aug. 2019].

Li, S. (2018). *Multi-Class Text Classification Model Comparison and Selection*. TowardsDataScience.com. [online] Available at: <https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568> [Accessed 20 Aug. 2019].

- Looper, E., Bird, S. (2002). NLTK: the Natural Language Toolkit. *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics*, 1, pp.63-70.
- Mack, S. (2019). *The Classification of Products in Marketing*. Chron.com. [online] Available at: <https://smallbusiness.chron.com/classification-products-marketing-65394.html> [Accessed 20 Aug. 2019].
- Marshall, C. (2018). *Random forest text classification in R: Trump v. Obama*. TowardsDataScience.com. [online] Available at: <https://towardsdatascience.com/random-forest-text-classification-trump-v-obama-c09f947173dc> [Accessed 20 Aug. 2019].
- Pedregosa *et al.* (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning*, 12(10), pp.2825-2830.
- Porter, M. (1980). An algorithm for suffix stripping. *Program*, 14(3) pp.130-137. [online] Available at: <http://people.scs.carleton.ca/~armyunis/projects/KAPI/porter.pdf> [Accessed 20 Aug. 2019].
- Pulp Fiction. 1994. [Film]. Quentin Tarantino. dir. USA: Miramax.
- Sammut C. and Webb G. (2011). TF-IDF. *Encyclopedia of Machine Learning*. Boston, US.
- Sarkar, D. (2019). *Text Analytics with Python*. 2th ed. Bangalore, India. Apress Media.
- Scikit-learn.org. (2019). Sklearn.metrics.accuracy_score. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html [Accessed 20 Aug. 2019].
- Shaikh, J. (2017). *Machine Learning, NLP: Text Classification using scikit-learn, python and NLTK*. TowardsDataScience.com. [online] Available at: <https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a> [Accessed 20 Aug. 2019].
- Sundar, A. (2018). ML-Powered Product categorization for smart shopping options. TowardsDataScience.com. [online] Available at: <https://towardsdatascience.com/ml-powered-product-categorization-for-smart-shopping-options-8f10d78e3294> [Accessed 20 Aug. 2019].
- Surma, M. (2018). *Product classification – the ideal task for machine learning*. AltkomSoftware.pl. [online] Available at: <https://altkomsoftware.pl/en/blog/machine-learning/> [Accessed 20 Aug. 2019].
- Vandic, R., Frasincar, F. and Kaymak, U. (2018). A Framework for Product Description Classification in e-Commerce. *Journal of Web Engineering*, 17(1).
- Webb G. (2011). Naïve Bayes. *Encyclopedia of Machine Learning*. Boston, US. Springer.
- Xu, B., Guo, X., Ye, Y. and Cheng, J. (2012). An Improved Random Forest Classifier for Text Categorization. *Journal of Computers*, 7(12).
- Zhang, X. (2011). *Support Vector Machines*. *Encyclopedia of Machine Learning*. Boston, US. Springer.

Appendices

Appendix 1

Multi-Journey Approach accuracy table

| Journey | Size | Manufacturer | SubBrand | SubCategory |
|---------------------------------|-------|--------------|----------|-------------|
| Sweets | 865 | 0.872832 | 0.768786 | 0.635838 |
| Pet | 15292 | 0.972213 | 0.926773 | 0.90389 |
| Biscuits | 1319 | 0.863636 | 0.784091 | 0.727273 |
| Coffee | 1537 | 0.935065 | 0.857143 | 0.941558 |
| Tea | 1238 | 0.907258 | 0.866935 | 0.75 |
| Shaving | 855 | 0.953216 | 0.707602 | 0.666667 |
| Skincare | 3293 | 0.919575 | 0.758725 | 0.772382 |
| Frozen-Chips & Potatoes | 237 | 0.979167 | 0.645833 | 0.645833 |
| Tinned Vegetables | 282 | 0.929825 | 0.877193 | 0.929825 |
| Suncare | 1099 | 0.9 | 0.722727 | 0.795455 |
| Frozen-Meat & Poultry | 436 | 0.909091 | 0.727273 | 0.568182 |
| Frozen Pizza | 230 | 0.956522 | 0.630435 | 0.913043 |
| Cheese | 1396 | 0.896429 | 0.810714 | 0.935714 |
| Frozen | 177 | 0.944444 | 0.638889 | 0.305556 |
| Breakfast Cereal | 1753 | 0.900285 | 0.763533 | 0.732194 |
| Baking Buttercream & Frosting | 91 | 0.947368 | 0.684211 | 0.578947 |
| Frozen-Vegetarian | 254 | 0.882353 | 0.705882 | 0.411765 |
| Mexican | 352 | 0.901408 | 0.732394 | 0.859155 |
| Chilled-Vegetarian | 239 | 0.833333 | 0.666667 | 0.479167 |
| Yoghurt | 694 | 0.848921 | 0.769784 | 0.791367 |
| Baking Cake Mixes | 105 | 0.714286 | 0.619048 | 0.714286 |
| Cereal Bars | 648 | 0.776923 | 0.692308 | 0.753846 |
| Beer | 1199 | 0.641667 | 0.666667 | 0.725 |
| Ice Cream | 744 | 0.852349 | 0.812081 | 0.697987 |
| Fish | 406 | 0.890244 | 0.756098 | 0.670732 |
| Chilled-Ready Meals | 1123 | 0.888889 | 0.688889 | 0.626667 |
| Chocolate | 2605 | 0.911708 | 0.802303 | 0.677543 |
| Chilled-Meat & Poultry | 1865 | 0.922252 | 0.898123 | 0.903485 |
| Chilled Ready To Bake Pastry | 52 | 1 | 0.818182 | 0.909091 |
| Frozen-Vegetables | 314 | 0.984127 | 0.936508 | 0.698413 |
| Protein Bars | 155 | 0.83871 | 0.83871 | 0.83871 |
| Baby Food | 1136 | 0.973684 | 0.75 | 0.903509 |
| Fresh - Confectionary | 116 | 0.583333 | 0.416667 | 0.333333 |
| Frozen-Ready Meals | 499 | 0.788889 | 0.644444 | 0.588889 |
| Frozen-Desserts, Fruit & Pastry | 188 | 0.868421 | 0.815789 | 0.631579 |
| Fizzy Drinks | 927 | 0.88172 | 0.83871 | 0.817204 |
| Pasta Sauces | 565 | 0.911504 | 0.849558 | 0.840708 |
| Frozen-Pies & Slices | 72 | 0.733333 | 0.8 | 1 |

| | | | | |
|-----------------------------|------|----------|----------|----------|
| Baking | 382 | 0.961039 | 0.844156 | 0.636364 |
| Coffee Machines | 219 | 0.818182 | 0.704545 | 0.795455 |
| Table Sauces & Marinades | 1262 | 0.889328 | 0.794466 | 0.758893 |
| Grocery Breakfast Bars | 187 | 0.815789 | 0.657895 | 0.921053 |
| Asian Cooking | 1033 | 0.869565 | 0.772947 | 0.78744 |
| Medicine Sweets | 53 | 0.909091 | 0.818182 | 0.909091 |
| Dishwashers | 356 | 0.930556 | 0.694444 | 0.930556 |
| Butter, Spreads & Margarine | 73 | 0.666667 | 0.466667 | 0.933333 |
| Soft Drinks | 182 | 0.945946 | 0.864865 | 0.648649 |
| Alcohol-Premix | 58 | 0.75 | 0.916667 | 0.833333 |