Universitat Oberta de Catalunya
uoc.edu

A Product Data Matching System for an e-Commerce Aggregator

Teguayco Gutiérrez González
Master's Degree in Data Science
Data Mining & Machine Learning

Laia Subirats Maté (Consulting Professor)
Jordi Casas Roma (Responsible Professor)

June 2019

Copyright

SHEET OF WORK

Title:	A Product Data Matching System for an e-Commerce Aggregator
Author:	Teguayco Gutiérrez González
Consulting Professor:	Laia Subirats Maté
Responsible Professor:	Jordi Casas Roma
Delivery Date:	June 2019
Studies:	Master's Degree in Data Science
Area:	Data Mining and Machine Learning
Language:	English
Keywords:	Product Matching, e-Commerce, Machine Learning

Abstract

In this work, an approach to address the Product Matching problem is implemented. The Product Matching consists in identifying the referenced or target product in different incoming product offers, which is not a trivial task as the source product offers are often presented in unstructured and different formats. Furthermore, universal product identifiers (e.g. GTIN, EAN or UPC) are missing most of the times. Thus, this functionality provides a special point of quality when building services such as e-Commerce aggregators, allowing the retrieval of all the available offers for a given product.

This issue has been the subject of numerous previous studies where different approaches have been proposed, especially those related to the field of Machine Learning.

The obtained results may help to automate the Product Matching task for a company which has developed an e-Commerce aggregator and has also provided their crawled product data for the development of this work. As a result, several Machine Learning classifiers have been generated, some of them giving accuracy results above 95 percent, and a REST API has been made publicly accessible as a way of making these models available and ready to use.

Pon lo que significan estos acrónimos

Quizá incluiría aquí la URL de la API rest

Table of contents

Es raro que el pu

Quizá pondría methodology en vez de rationale of the ir

Introduciría también un índice de figuras y un índice de tablas

Acknowledgements

I would like to dedicate this work to all the people that made it possible. First of all, to my parents, those who taught me throughout the years the importance of hard work. Without them, not only this work would not be possible, but also all the achievements in my life.

Secondly, to the company which collaborated first, giving me interesting ideas to address in this final work and then, providing me a very rich dataset to accomplish the development. The final result constitutes a work to be very proud of and that goes beyond academic boundaries.

Este trabajo va especialmente dedicado a mis padres, aquellas personas que me han enseñado la importancia del trabajo duro a lo largo de los años. Sin ellos, no solo este trabajo habría sido posible, sino también cualquiera de los logros conseguidos en mi vida.

También agradecer a la empresa colaboradora en este trabajo no solo la proporción de los datos para el desarrollo sino también la ayuda a la hora de decidir una temática interesante que cubrir. Gracias a ello, el resultado final constituye un trabajo del que sentirse orgulloso y que va más allá de los límites académicos.

1. Introduction

Quizá pondría unas líneas de qué se compone esta sección.

1.1 Context and rationale for the project

Nowadays, a bunch of different e-Commerce aggregators can be found on the Internet, which are websites that collect data of products from different online shops such as *Amazon*, *MediaMarkt* or *ToysRUs* with the aim of showing their customers the best options to buy a certain product. *Google Shopping* or *Idealo* are two famous examples of e-Commerce aggregators.

However, the problem of identifying the same products collected across many sources, known as “Product Matching”, must be addressed in order to offer an optimal service. In most cases this is not a trivial task, as data is presented in different formats depending on the source they come from.

In this work, it is implemented a proposed using Machine Learning and Natural Language Processing (NLP) techniques to automatically detect matches between products whose data come from different sources. In particular, this product data matching system will be developed based on the data products collected by a real-world company that has developed an e-Commerce aggregator. This solution may help this company to automate the product matching task (which is currently done manually), thereby freeing human resources to attend to other tasks.

1.2 Objectives

The main objective of the work is to develop one or more than one Machine Learning classifier that can be able to detect matches, with considerable confidence levels, between products coming from different web shops and products known for the company contained in its master-data storage.

These products to be matched will be initially smartphones, as they are a predominant type of products the company operates with and it is also desired to start with a homogenous dataset given the selected approach to implement.

Furthermore, the following can be taken as secondary objectives:

- To explore alternatives to create machine learning classifiers or extend the existing ones to be able to work with more heterogeneous products, not only smartphones.

- To develop an app or final product to input data for a certain product to output the best match for it. This product may serve to make the generated models available and ready to use for the collaborator organization’s e-Commerce aggregator system.

Formato: Falta ju

1.3 Approach and methodology

Aquí pondría Machine Learning (ML)

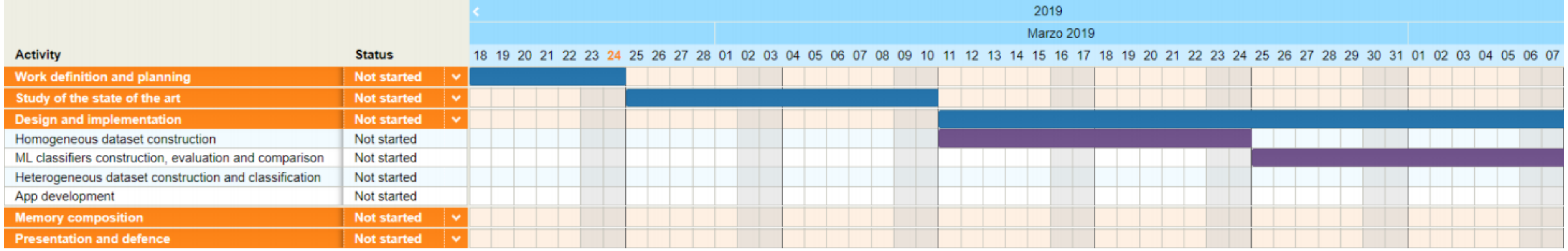
Given the large variety of products the company handles and the amount of Machine Learning classifiers that can be applied, a progressive approach will be used throughout the development of this work. So, to begin with, ML techniques will be implemented for a homogeneous set of products (in this case, smartphones). Thus, if time allows, more heterogeneous datasets will be generated and used to train different ML models each time.

Furthermore, as there are a lot of works addressing the “Product Matching” problem, the already-used approaches and techniques will be thoroughly analysed to then apply the subset of them which best fits for the specific product data the collaborator organization owns.

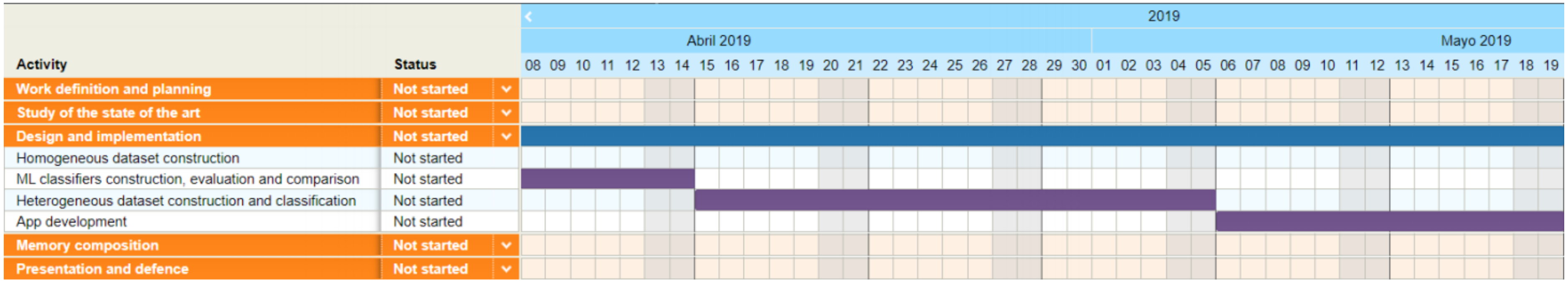
1.4 Planning

The following Gantt diagram illustrates how the tasks and milestones of the project are temporarily distributed.

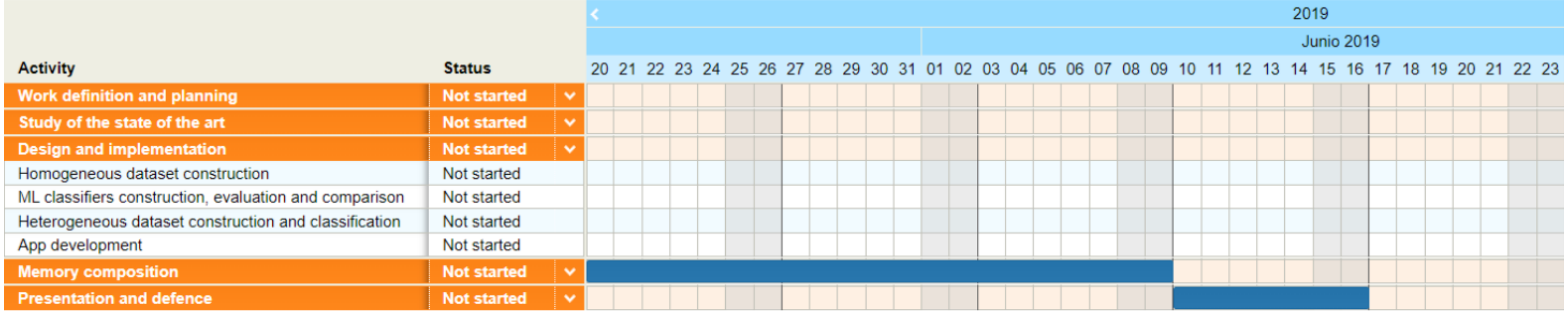
hablaría en pasado ya que el trabajo ya lo has acabado: have been distributed.



Activity	Status
Work definition and planning	Not started
Study of the state of the art	Not started
Design and implementation	Not started
Homogeneous dataset construction	Not started
ML classifiers construction, evaluation and comparison	Not started
Heterogeneous dataset construction and classification	Not started
App development	Not started
Memory composition	Not started
Presentation and defence	Not started



Activity	Status
Work definition and planning	Not started
Study of the state of the art	Not started
Design and implementation	Not started
Homogeneous dataset construction	Not started
ML classifiers construction, evaluation and comparison	Not started
Heterogeneous dataset construction and classification	Not started
App development	Not started
Memory composition	Not started
Presentation and defence	Not started



Activity	Status
Work definition and planning	Not started
Study of the state of the art	Not started
Design and implementation	Not started
Homogeneous dataset construction	Not started
ML classifiers construction, evaluation and comparison	Not started
Heterogeneous dataset construction and classification	Not started
App development	Not started
Memory composition	Not started
Presentation and defence	Not started

Numeraría esta tabla

1.5 Summary of obtained products

As a final product, the already-mentioned REST API developed to make the Machine Learning models predictions available is publicly accessible on Heroku.

Añadiría aquí la URL Heroku.

Furthermore, as intermediate products, attached to this document all the *Jupyter* notebooks used to generate the models and the remaining source code can be found.

Añadiría aquí la URL a tu Github donde hay los Jupyter notebooks.

1.6 Chapters in the memory

The contents covered in each of the chapters in this memory are the following:

- **Chapter 1: Introduction**. It specifies the project consistency, objectives and planning, amongst others.

- **Chapter 2: State of the art**. A comprehensive review of previous studies which tackle the product matching problem and the approaches they pose is done.

creo que más que work design pondría metodología.

- **Chapter 3: Problem introduction and work design**. It explains how the work is stated and how it will be addressed.

- **Chapter 4: Work implementation**. It explains in a more detailed and technical way how the work is implemented. It also covers some key theoretical concepts kept into account during this development.

- **Chapter 5: Source code**. It summarizes how the attached source code is structured.

- **Chapter 6: Conclusions**. It synthesizes the most important points that can be extracted from the obtained results.

- **Chapter 7: Future work**. It arises which can be worked in order to improve or extend the obtained results in this project.

- **Bibliography**. Enumeration of all the revised previous works, given by order of appearance throughout the document.

Additionally, a **glossary** of key concepts can be found at the end of the document.

2. State of the art

Aquí también pondría un par de frases indicando qué subsecciones tiene esta sección.

2.1 Introduction

Pondría qué significan estos acrónimos ya que es la primera vez que salen.

As explained in the introductory section, the problem of matching products coming from different sources brings with it the challenge of identifying identical products whose data may be missing (for instance, universal identifiers such as EAN or UPC codes could not be found) or presented in multiple formats. These difficulties make this matching task to be a non-trivial one and, as a result, a lot of research studies and tools have already arisen, most of them implementing Natural Language Processing (NLP) and Machine Learning techniques.

In this section, a comprehensive review of these previous works will be done based on the approaches they pose to analyse how they implement the solutions for this issue. It should also be noted that some of these works use more than one approaches.

2.2 Approaches

2.2.1 Text similarity

Falta poner el acronimo de TF-IDF. Pondría: Term frequency–inverse document frequency (TF-ID

The first approach could consist in comparing two product titles or descriptions to determine th [illegible] different web shops should have similar titles and descriptions. In NLP, a common solution is to use TF-IDF vectorization and calculate some measure, like the cosine similarity, to quantify the degree of similarity between two titles. TF-IDF vectorization could constitute a first valid approach, as it is a method which is able to identify the most important words in a given document.

However, in [1] they pose an adaptative string similarity measure that improves the matching results using TF-IDF and therefore demonstrate that the latter method has limitations when addressing the product matching problem. This proposal is based on giving higher weights to those terms in product titles that are more relevant to identify the product, as depending on the product, some terms will identify them better than others. For example, to identify a camcorder like the following:

Canon Vixia HF S10

The manufacturer code (HF S10) should have a high weight, as it is the most relevant term to identify the product. However, for the following cleaning toolkit for the former camcorder:

Canon Vixia HF S10 Cleaning Toolkit

The type of product (cleaning toolkit) should have the highest weight in order to avoid matching it with the former camcorder.

Word embeddings are another alternate approach to handle words as vector representations. Word2Vec [2] and GloVe [3] are possibly the best-known developed techniques to get such representations of words. Word2Vec has been already used in a work about product recommendations in e-Commerce [4], where the implemented language models are used to vectorize product titles and handle these vectorizations to make recommendations.

2.2.2 Attribute extraction

Most of the times, product titles include technical information which is crucial to identify them. However, in order to apply Machine Learning techniques, a more structured data about products should be managed. Therefore, the problem of obtaining these structured data about products from their textual information is arisen.

As an example, from the following product title:

Apple iPhone XS (64GB) – Gold

It would be desired to filter it to obtain the following structured data:

```
{
   “Brand”: “Apple”,
   “ProductName”: “iPhone”,
   “Model”: “XS”,
   “MemoryCapacity”: “64”,
   “MemoryCapacityUnit”: “GB”,
   “Color”: “Gold”
}
```

This problem can be seen as an instance of an NLP task known as **Named Entity Recognition** (NER). The goal in NER is to locate and classify the entities mentioned inside some given unstructured text.

2.2.2.1 Dictionaries and regular expressions

One straightforward solution may consist in using **regular expressions** and **dictionaries** of known values for the attributes to extract. However, this approach can have several disadvantages.

On one hand, the use of regular expressions could lead to an inaccurate identification of attributes. For example, given the product name “Apple iPhone 32GB”, a regular expression can be written to extract the phrase “32GB”. However, this method would not be able to correctly resolve if this value correspond to RAM memory or to hard-disk capacity, as both attributes are commonly given in GB. Furthermore, as the information may be presented in many different formats, a regular expression should be written to handle every instance. For example, the inches of the screen for a certain TV could be written like “50-inches” or ‘50”’, so two different regular expressions are needed in this case.

In order to bypass the problem of writing a lot of regular expression to handle every possible case, studies address the issue of “learning” regular expression [5, 6, 7]. Genetic programming is the favourite used approach when exploring this subject matter.

On the other hand, the use of dictionaries may be a good approach for attributes with a fixed list of possible values like the colour. However, it could fail with attributes with an open list of possible values such as the brand, as new brands come eventually out on the market. In this latter case, the dictionary used to perform brand extraction should be regularly updated.

2.2.2.2 Sequence labelling

Some studies [8, 9, 10] address this issue by implementing more sophisticated methods. They first train some NER model to extract products features from their titles to then train Machine Learning models based on the structured data obtained to make the matching.

Thus, ***Conditional Random Fields*** (CRF) can be seen as the favourite alternative, as it is used by the three of the above-cited works. It is a linear model for sequential labels. In fact, CRF can be considered as the sequential version of logistic regression [11].

Another implemented solution by [8] along with CRF is ***Structured Perceptron***, which is a supervised learning algorithm. By contrast, [9] goes beyond implementing CRF also with **text embeddings**.

The benefit of these sequence labelling algorithms is that they leverage the information given by the context of a given word in a certain product title. For example, in a product title like “Apple iPhone 4 (4GB RAM) - Black”, “4GB” could be recognized as RAM memory considering the next word is “RAM”. Using some of the above-mentioned simplest solutions, this information would not be exploited.

Nevertheless, sequence labelling algorithms need a significant sizable training set (usually labelled using BIO encoding) whose manual labelling would require a huge usage time. Some implemented alternatives to avoid this manual task consist in the use of “distant supervision”, in which a training dataset is automatically built based on heuristics and rules.

2.2.3 Image recognition

Matching products identifying these by performing image recognition on them could constitute the hardest approach of all. It is not only the complexity of models like ***Convolutional Neural Networks*** (CNN), which are a common solution for image recognition problems, but also some other challenges that have to be faced regarding products recognition from images: the same product can be found photographed from different perspectives, with different colours or different levels of brightness. Furthermore, CNNs could need a huge number of images to be trained, something that translates into managing and storing a lot of bytes of data.

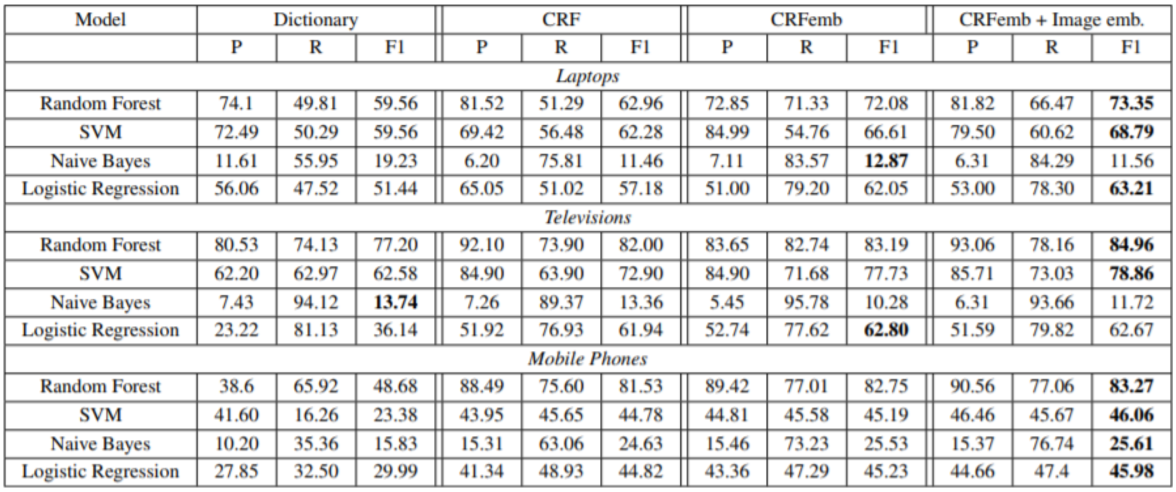
Indeed, [9] implements also a CNN on the premise that most of the web shops use the same image for identical products. Treating the product matching problem as a two-class classification problem (given a pair of products, decide if they match or not), the implemented CNN allows to obtain image embeddings for the candidate products pair. Then, cosine similarity between both vectors is calculated to determine if they are similar enough to match. However, this work consider similarity image embeddings as a weak indicator for product matching, as it performs better for the category prediction task.

2.3 Results

En el título añadiría Results of previous studies, ya que sino parece que son tus resultac

As a way of getting an idea on how good the results obtained so far in previous studies are, below the obtained results by [9] are shown.

The following figure summarizes the results showing the values obtained for the metrics *precision*, *recall* and *f1-score* when addressing the product matching problem using different approaches to detect features in the products and using several classifiers for the matching task (Random Forest, Support Vector Machines, Naïve Bayes and Logistic Regression). Also, three different types of product were used: laptops, televisions and mobile phones.



Model	Dictionary			CRF			CRFemb			CRFemb + Image emb.		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Laptops												
Random Forest	74.1	49.81	59.56	81.52	51.29	62.96	72.85	71.33	72.08	81.82	66.47	**73.35**
SVM	72.49	50.29	59.56	69.42	56.48	62.28	84.99	54.76	66.61	79.50	60.62	**68.79**
Naive Bayes	11.61	55.95	19.23	6.20	75.81	11.46	7.11	83.57	**12.87**	6.31	84.29	11.56
Logistic Regression	56.06	47.52	51.44	65.05	51.02	57.18	51.00	79.20	62.05	53.00	78.30	**63.21**
Televisions												
Random Forest	80.53	74.13	77.20	92.10	73.90	82.00	83.65	82.74	83.19	93.06	78.16	**84.96**
SVM	62.20	62.97	62.58	84.90	63.90	72.90	84.90	71.68	77.73	85.71	73.03	**78.86**
Naive Bayes	7.43	94.12	**13.74**	7.26	89.37	13.36	5.45	95.78	10.28	6.31	93.66	11.72
Logistic Regression	23.22	81.13	36.14	51.92	76.93	61.94	52.74	77.62	**62.80**	51.59	79.82	62.67
Mobile Phones												
Random Forest	38.6	65.92	48.68	88.49	75.60	81.53	89.42	77.01	82.75	90.56	77.06	**83.27**
SVM	41.60	16.26	23.38	43.95	45.65	44.78	44.81	45.58	45.19	46.46	45.67	**46.06**
Naive Bayes	10.20	35.36	15.83	15.31	63.06	24.63	15.46	73.23	25.53	15.37	76.74	**25.61**
Logistic Regression	27.85	32.50	29.99	41.34	48.93	44.82	43.36	47.29	45.23	44.66	47.4	**45.98**

In bold, they highlight they highest F1-score obtained for each classifier in each type of products.

3. Problem Introduction & Work Design

Lo mismo, indicaría que subsecciones incluyen esta sección.

3.1 Introduction

As pointed out in the introductory section, when building an e-Commerce aggregator, it is vital to address the **Product Matching** problem, as the capability of identifying the same product from different incoming product offers brings a high quality for a service such as this one. Thus, getting all the offers for a given product is possible, so that a client can compare among the existing offers in the market and then choose the one that best fits their needs. Furthermore, even a certain seller or supplier can use this service to study their competitors' offers.

The company which provides the data for the implementation of this work is currently tackling the Product Matching issue by manually assigning the new incoming product data offers to their corresponding base products, which are already existing in the company's database. The figure below illustrates how three offers for the same product coming from three different sellers point to the same base product in the company's storage:

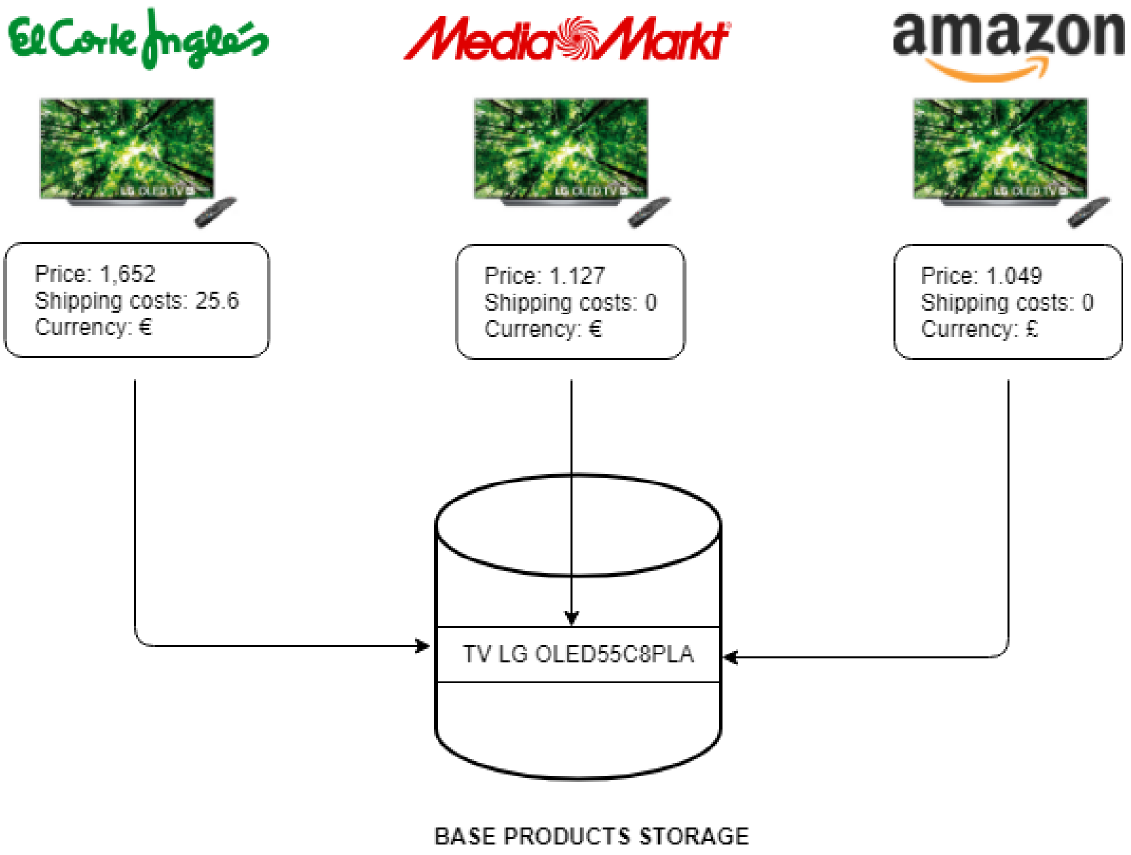


Figure 1: Product Matching for TV offer

The operators who perform this task usually navigate to the source URL of the crawled product, study their features, compare them to the most similar base products stored in the database and, if a good candidate is found, the matching or assignment is done. If none of the existing base products fits well enough with the incoming product offer, a new base product must be created in the database.

3.2 Problem statement and scope

The problem to solve can be seen as a supervised learning task. More specifically, it is desired to solve a **classification problem**: given a pair of products, predict whether they match or not (i.e. the labels or classes are *match* and *unmatch*).

Furthermore, as the product matching task must be performed by examining the features of the products and these features may be different depending on the types of product being handled, the task becomes more complex. For example, features such as “storage space” or “screen size” can be found in products like smartphones or tablets, whereas they do not exist for products like printers or household items. Because of this heterogeneity regarding product features, **this work focuses on the matching of smartphones**, which is a type of product the organization providing the data is highly keen on for its e-Commerce aggregator system. Thus, increasing the variety of products to be matched will constitute the main objective for the future work.

3.3 Provided real data

The collaborator organization has provided a CSV file for the development of this work (`smartphones_offers.csv`) containing 195,195 rows about crawled smartphone offers. For each crawled offer, there are attributes or fields giving information about the products themselves (brand, product title, …), the offer (prices, shipping costs, delivery information) and the matches performed manually by the data operators (through the attribute “MatchingID”).

Concretely, the following are the fields contained in the provided file:

- **ProductID**: internal identifier for the product.

- **CrawlerDateTime**: time the offer was crawled.

- **WebShopID**: internal identifier for the web shop the offer was crawled from.

Formato: Falta justific

- **WebShopName**: name of the web shop the offer was crawled from.

- **PSMID**: internal identifier for the *Price Search Engine* (*Preissuchemaschine* in German) the offer was crawled from.

- **PSMName**: name of the *Price Search Engine* the offer was crawled from.

- **IsPriceSearchEngine**: flag set to true if the web shop the offer was crawled from is a Price Search Engine.

- **TerritoryID**: identifier for the country of the source web shop.

- **LanguageID**: identifier for the language of the source web shop.

- **WebShopProductID**: identifier for the product given by the source web shop.

Falta justificar e

- **MatchingID**: identifier of the base product the offer's product matches with in the organization's base products storage. This field is the one used to generate the training dataset for the machine learning classifiers, as described further.

- **BrandID**: internal identifier for the brand of the product in the offer.

- **Brand**: name of the brand of the product in the offer.

- **Title**: crawled title of the product.

- **DeliveryTimeID**: internal identifier for the delivery time of the offer.

- **DeliveryTime**: normalized delivery time of the offer.

- **ShippingCosts**: shipping costs of the product in the crawled offer.

- **Price**: price of the offer.

- **Currency**: currency of the offer. The only values found are euros (EUR) and Swiss francs (CHF).

- **ProductLink**: link to the product detail page of the crawled offer.

- **FullNamePath**: category path where the product in the offer is located inside the source web shop. It is also known as "breadcrumbs".

- **Color**: color of the product crawled, if it exists. The name of the color can be found in German, English or Spanish.

The remaining fields contained in the provided CSV file are irrelevant, as they are only used for internal data maintenance purposes in the organization.

3.4 Rationale of the implemented approach

From all of the approaches that tackle the product matching problem examined in the state of the art section, it becomes clearly that a model which allows the extraction of product features from unstructured text such as product titles is needed: firstly, because the e-Commerce aggregator of the organization is fed from product data which is commonly unstructured (regarding the products, only titles are crawled; descriptions sometimes); secondly, this unstructured data is determined by the lack of structured data inside the web shops they information is crawled from. Because of this, it was decided that a ***Conditional Random Fields*** (CRF) model to extract the features of the products needed to perform the matching would be a good choice, given the numerous previous studies applying it [8, 9, 10] and the good results obtained.

Regarding the matching problem itself, it was firstly proposed as a classification task where the label to predict would be the *MatchingID* field in the provided dataset, using the features extracted from the titles by the CRF model as the attributes in the training dataset. However, despite the training of a kNN classifier obtained very good results, it was noted that there was **lack of generalization**; that is, the classifier could make really accurate prediction for products seen during the training set but failed miserably for unknown products.

Falta especificar

Due to this lack of generalization with the classification task proposal, another one had to be chosen. The final choice for the classification approach was to pose a **binary classification problem** which, given the attributes of a pair of products, decide whether they match (label *MATCH*) or not (label *UNMATCH*). This outlook allowed also the application of other classifiers, such as Random Forests, Logistic Regression and Support Vector Machines, apart from kNN. As a way of generating the training dataset for these ML models, the concept of ***distance vectors*** posed in [9] was chosen. Finally, it was proven that this approach generalized much better than the one chosen in first place. All of these models were generated using the well-known Python library *Sklearn*[1].

To conclude the work, a REST API was developed to make easier the use of the generated ML models with productive applications, as covered in section 5.4.

[1] https://scikit-learn.org/stable/

4. Work Implementation

Indicaría las subsecciones de esta sección

5.1 Feature extraction from product titles

The problem which consists in the extraction of features of products from their titles can be seen as a more general problem of the Natural Language Processing (NLP) field, known as ***Named-entity recognition (NER)***.

In the NER task, given a bunch of textual information, which is unstructured, it is desired to recognize and extract entities that can be contained in it. For example, given the following piece of text:

"*Jon Ziomek has written a riveting account of the errors that unfolded before the worst civil aviation disaster in history, when on March 27, 1977, two Boeing 747 passenger jets—KLM Flight 4805 and Pan Am Flight 1736—collided on a runway at Tenerife island, killing 583 people. Ziomek spoke with* Air & Space *senior associate editor Diane Tedeschi in April.*"

Entities such as the following can be identified in the former text:

- Jon Ziomek: **Person**
- March 27, 1977: **Date**
- Tenerife: **Place**
- Air & Space: **Organization**
- Diane Tedeschi: **Person**

Adapting this problem from NLP to the domain this work focuses on, from a product title such as the following:

Samsung Galaxy S10 Dual SIM 128GB 8GB RAM SM-G973F/DS White

The features (entities) found for the product in the title would be:

- Samsung: **brand**
- Galaxy: **model**
- S10: **model**
- 128: **storage**
- GB: **storage unit**
- 8: **RAM**
- GB: **RAM unit**
- SM-G973/DS: **manufacturer code**
- White: **color**

Given the complexity of this specific problem and the available provided data, the developed process of recognizing product features will be performed on smartphones to identify three types of entities: **brand, model and color**.

5.1.1 CRF model

Conditional Random Fields, as mentioned in the state of the art section, is one of the most common chosen alternatives when addressing the NER problem. It is a model that allows to leverage the information giving not only by the sequences, but also by the words themselves to label sequential data. For instance, for the already-seen smartphone title:

Samsung Galaxy S10 Dual SIM 128GB 8GB RAM SM-G973F/DS White

The model would learn that "S10" is part of the model because it comes after the word that constitutes the beginning of the model ("Galaxy") and contains alphanumerical characters.

Sklearn itself does not have an implementation of the CRF model, but **CRF-suite**[2], a fast implementation of this model written in C++, has a Python wrapper compatible with Sklearn called *sklearn-crfsuite*[3]. This latter implementation is the one used in this work.

5.1.2 Training dataset

Indicaría las subsecciones de esta sección para ayudar al lector.

5.1.2.1 BIO encoding

The training dataset for the CRF model consists of smartphones titles correctly labelled using **BIO encoding**[4] (which stands for *Beginning, Inside, Outside*). Using this tagging format, every word in the sequences (i.e. the product titles) that belong to an entity to be recognized are assigned a label with the prefix *B-* or *I-* depending on the position they occupy in the entity: *B-* if the word is the beginning of the entity and *I-* if the word is inside or at the end of the entity. The label *O* is used to mark non-relevant words. For instance, for the already-seen smartphone title, the corresponding BIO encoding would be the following:

Pondría un título a la tabla: Table 2: BIO encodings.

Title	**BIO Tags**
Samsung	B-Brand
Galaxy	B-Model
S10	I-Model
Dual	O
SIM	O
128GB	B-Storage
8GB	B-RAM
RAM	I-RAM
SM-G973F/DS	B-ManufacturerCode
White	B-Color

[2] http://www.chokkan.org/software/crfsuite/
[3] https://sklearn-crfsuite.readthedocs.io/en/latest/
[4] https://en.wikipedia.org/wiki/Inside%E2%80%93outside%E2%80%93beginning_(tagging)

5.1.2.2 Distant supervision

Labelling manually all the smartphone's titles provided by the collaborator organization to generate the needed training dataset for the CRF model would constitute a huge time-consuming task. Because of this, **distant supervision** was used to build this dataset, so the process could become faster.

Distant supervision refers to the process of generating a training dataset based on "the use of an external knowledge source" [14]. In this case, these external sources will be, as explained later in this section, a dataset found at Kaggle and scraped data. However, the final dataset built using distant supervision may content wrong labelling in some cases.

5.1.2.3 Dataset generation

The Python code in charge of building the training dataset for the CRF model is contained in the class `BIOTagger.py`. For every provided smartphone title, the `BIOTagger` tags every word contained in the title according the BIO encoding discussed earlier by the use of dictionaries for the features the model will be able to predict (as commented earlier, those features will be: brands, models and colors).

These used dictionaries refer to the set of known values for brands, models and colors and have been obtained from:

1. A dataset about smartphones found in *Kaggle*[5].
2. Applying web-scraping in the website *GMSArena*[6].
3. The dataset provided by the collaborator company.

The first two sources were used to get names of smartphone's brands and models, whereas the latter was used to get names for colors.

Kaggle dataset

The dataset found in Kaggle contains information about 8631 types of smartphones, covering from details such as the brands and models and more technical features (storage, operating system, screen inches, etc). However, the only attributes in this dataset that are important for the purpose of this work are the brand and the model.

Web-scraping on the website GMSArena

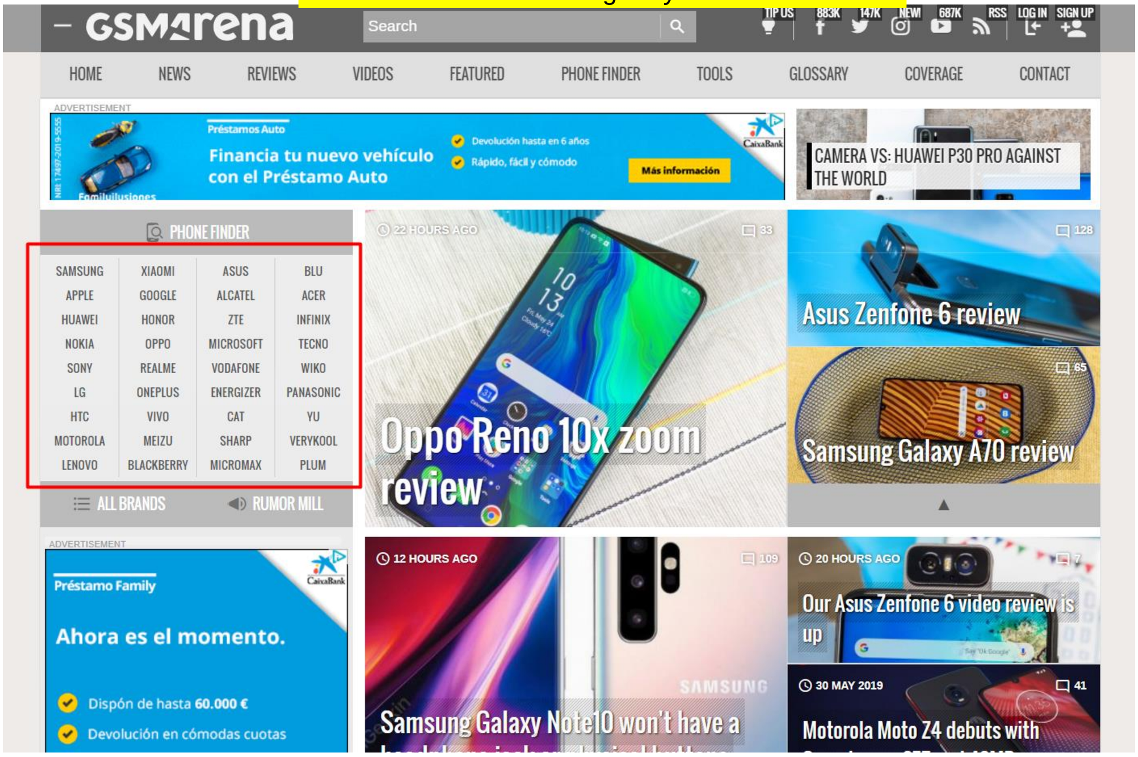
In order to enrich the dictionaries of brands and models obtained with the already mentioned dataset found in Kaggle, web-scraping processes were also applied on the website GMSArena.

[5] https://www.kaggle.com/arwinneil/gsmarena-phone-dataset

[6] https://www.gsmarena.com/

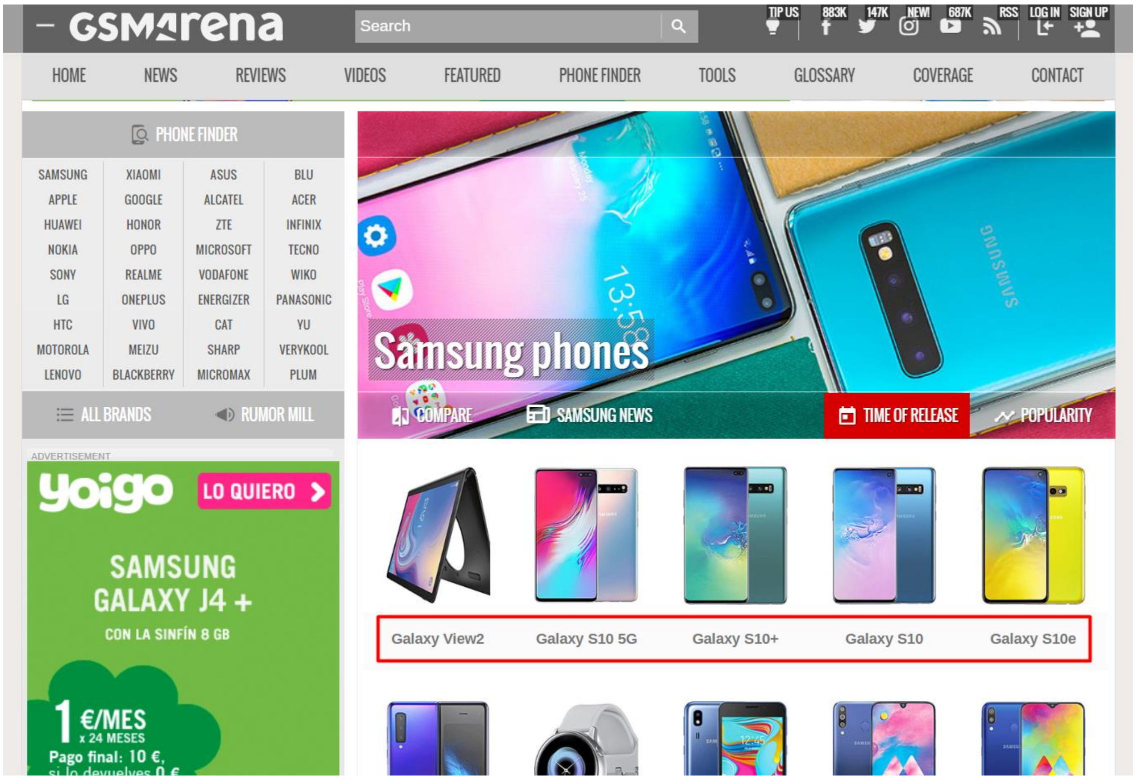
The Python code in charge to do is contained in `src/smartphones-db-crawler`. Firstly, the scraper visit the main URL of the site and extract the names and URLs of the brands located in the left side of the page:

Añadiría un título a esta figura y la numeraría.



GSMArena
Search
TIP US 883K 147K NEW! 687K RSS LOG IN SIGN UP
HOME NEWS REVIEWS VIDEOS FEATURED PHONE FINDER TOOLS GLOSSARY COVERAGE CONTACT
PHONE FINDER
SAMSUNG XIAOMI ASUS BLU
APPLE GOOGLE ALCATEL ACER
HUAWEI HONOR ZTE INFINIX
NOKIA OPPO MICROSOFT TECNO
SONY REALME VODAFONE WIKO
LG ONEPLUS ENERGIZER PANASONIC
HTC VIVO CAT YU
MOTOROLA MEIZU SHARP VERYKOOL
LENOVO BLACKBERRY MICROMAX PLUM
ALL BRANDS RUMOR MILL
CAMERA VS: HUAWEI P30 PRO AGAINST THE WORLD
Oppo Reno 10x zoom review
Asus Zenfone 6 review
Samsung Galaxy A70 review
Samsung Galaxy Note10 won't have a
Our Asus Zenfone 6 video review is up
Motorola Moto Z4 debuts with

Secondly, it visits all of the URLs collected in the former step and scrapes the names of the models which are in smartphones list. For example, in the case of the brand *Samsung*:



GSMArena
Search
HOME NEWS REVIEWS VIDEOS FEATURED PHONE FINDER TOOLS GLOSSARY COVERAGE CONTACT
PHONE FINDER
SAMSUNG XIAOMI ASUS BLU
APPLE GOOGLE ALCATEL ACER
HUAWEI HONOR ZTE INFINIX
NOKIA OPPO MICROSOFT TECNO
SONY REALME VODAFONE WIKO
LG ONEPLUS ENERGIZER PANASONIC
HTC VIVO CAT YU
MOTOROLA MEIZU SHARP VERYKOOL
LENOVO BLACKBERRY MICROMAX PLUM
ALL BRANDS RUMOR MILL
Samsung phones
COMPARE SAMSUNG NEWS TIME OF RELEASE POPULARITY
Galaxy View2
Galaxy S10 5G
Galaxy S10+
Galaxy S10
Galaxy S10e

Añadiría un título a esta figura y la numeraría.

These smartphones list are considerably big, so they incorporate pagination buttons at the end to navigate to further pages. The developed scraper can also handle this pagination to reach all of the smartphones data stored in this website.

All the data scraped can be found in the file `src/_data/SmartphonesBrandsModels.csv`, and contains 6443 rows (corresponding to 6443 different smartphones found).

CRF training dataset format

Finally, the `BIOTagger` object, using the above-explained information, generates the file `src/_data/BIO_ENCODED_TITLES.csv`, which contains all the smartphone titles provided labelled with BIO encoding:

TitleNumber	Word	BIOTag
…	…	…
90682	xiaomi	B-BRAND
90682	mi	B-MODEL
90682	8	I-MODEL
90682	128	O
90682	gb	O
90682	weiß	B-COLOR
90682	dual	O
90682	sim	O
…	…	…

Pondría título y numeraría esta tabla.

5.1.3 Model training

The training of the CRF model can be found in `src/crf-model-training/CRF-Smartphones.ipynb`. Inside this Jupyter notebook, apart from the training process itself, the implementation of the function `title2features()` must be stressed. This function is used to tell the CRF model what features of the words found in the titles it must keep in mind during the learning process. For example, some of these features are if the word contains alphanumerical characters or what positions it occupies from the beginning or from the end. Finally, this function returns, for a given smartphone title, a Python list of dictionaries, each dictionary corresponding to the features found for each word contained in the provided title.

5.1.4 Model evaluation

In the model evaluation section in the above-mentioned Jupyter notebook, it can be observed that very good results (near 1 in most cases) are obtained for the precision, recall and f1-score metrics. The theoretical definition for each of these metrics will be covered in next chapters.

Along with the values obtained for the mentioned metrics, the learnt transitions by the model can also be inspected. For example, it can be seen that the model has learnt that a word which is inside a MODEL entity (I-MODEL) commonly follows a word which is at the beginning of that entity (B-MODEL). In the same way, the model also learnt what transitions are more unlikely to happen, such as the occurrence of a non-relevant word (O) followed by a word tagged with I-MODEL.

5.2 Product matching classification

To perform the classification task consisting in deciding whether two products are the same or not (i.e. they *MATCH* or *UNMATCH*) four of the most common Machine Learning models were trained using Sklearn: K Nearest Neighbors, Logistic Regression, Support Vector Machines and Random Forest.

For each of them, *Hyperparameter Tuning* using *K-Fold Cross-validation* with *k = 4* was performed in order to find the values for the most relevant parameters.

Both the hyperparameter tuning for the four trained classifiers and the training and model evaluation can be found in:

- `src/sklearn-ml-classifiers-smartphones/ML-Classifiers-Hyperparameter-Tuning.ipynb`

- `src/sklearn-ml-classifiers-smartphones/ML-Classifiers-Smartphones.ipynb`

5.2.1 Training dataset

The training dataset to solve this classification problem is generated in two steps:

1. Generation of matching and non-matching product pairs.
2. Calculation of distance vectors for each generated matching or non-matching product pair.

Both steps are performed using the following Python scripts:

1. `src/distance-vectors-matching-ds-generator/matching-products-pairs-generator.py`

2. `src/distance-vectors-matching-ds-generator/distance_vectors_generator.py`

The steps involved in the generation of this dataset are explained below.

5.2.1.1 Matching products pairs generation

From the file provided by the collaborator organization described in section 3.3 (`smartphones_offers.csv`) products pairs are generated. Every row in this file corresponds to a crawled product from the web which has a MatchingID, which is a numerical code that matches that crawled product with the base product known by the organization.

Product pairs are made using this MatchingID field, so that products with the same MatchingID will be matching pairs and products with a different MatchingID will form non-matching pairs. These product pairs have been exported to the file `src/_data/MATCHING_PRODUCTS_PAIRS.csv` and the following is a row contained in this file, corresponding to a **matching pair**:

Column	Value
ProductTitle1	*SAMSUNG Galaxy S6 Edge 32GB black*
ProductTitle2	*Samsung Galaxy S6 edge – SM*
ProductPrice1	497.0
ProductPrice2	359.7
ProductCurrency1	EUR
ProductCurrency2	EUR
Match	MATCH

Thus, the following is an example of a **non-matching pair**:

Column	Value
ProductTitle1	*Apple iPhone XR 256Gb Coral Libre*
ProductTitle2	*ShotX Smartphone isle blue*
ProductPrice1	1029.0
ProductPrice2	369.0
ProductCurrency1	EUR
ProductCurrency2	EUR
Match	UNMATCH

In total, 540668 matching and non-matching products pairs were generated in this step.

5.2.1.2 Distance vectors calculation

From the output generated in the former step (i.e. the file `src/_data/MATCHING_PRODUCTS_PAIRS.csv`) and the outputs given by the CRF model, the file `src/_data/MATCHING_DISTANCE_VECTORS.csv` was generated. This file itself constitutes the training set for the Machine Learning classifiers that will be fitted for the matching classification task.

In this step, for every product pair generated in the last step, a distance vector was calculated as follows:

1. Both *ProductTitle1* and *ProductTitle2* are passed to the CRF model, so that the prediction for the brand, model and colour of the smartphone are outputted.

2. For every smartphone feature found by the CRF model (i.e. brand, model and colour) and the attributes *ProductPrice1* and *ProductPrice2*, a distance value is calculated.

   2.1. If the values for the feature being compared are strings, the ***Levenshtein***[7] **distance** is calculated. Then a value between the range 0 and 1 is calculated from the former distance giving the **degree of similarity** between the two strings being compared. If the two strings are identical, this degree of similarity will be 1, whereas it will be 0 if the strings are completely different.

   2.2. If the values for the feature being compared are numerical (i.e. integers or float), then the **absolute value of its difference** is used.

3. The attributes *ProductCurrency1* and *ProductCurrency2* are only used to convert the prices into euros, so that prices in the same currency are used when calculating the distances.

4. Additionally, columns with the suffix "*FOUND*" are added indicating if the feature was found or not by the CRF model when performing feature extraction from product titles.

For the **matching product pair** example shown in the last section, the corresponding distance vector would be the following:

Column	**Value**
BRAND1	1
BRAND1FOUND	1
BRAND2	0
BRAND2FOUND	0
BRAND3	0
BRAND3FOUND	0
MODEL1	1
MODEL1FOUND	1

Numeraría y pondría título a esta tabla.

[7] **Levenshtein distance**: informally, it is defined as the minimum number of operations on a string (insertions, deletions or modifications of characters) needed to transform that one into another. For example, the Levenshtein distance between "Huawei" and "Hawaii" is 3, since the following set of operations is needed to transform one of them into another: on the word "Huawei", (1) deletion of *"u"*, (2) exchange of *"e"* by *"a"* and (3) addition of *"i"*.

MODEL2	1
MODEL2FOUND	1
MODEL3	1
MODEL3FOUND	1
MODEL4	0
MODEL4FOUND	0
COLOR	0
COLORFOUND	0
EUR_PRICE	137.3
EUR_PRICEFOUND	1
MATCH	MATCH

Thus, for the **non-matching product pair** example, the corresponding distance vector would be:

Column	**Value**
BRAND1	0
BRAND1FOUND	0
BRAND2	0
BRAND2FOUND	0
BRAND3	0
BRAND3FOUND	0
MODEL1	0
MODEL1FOUND	0
MODEL2	0
MODEL2FOUND	0
MODEL3	0
MODEL3FOUND	0
MODEL4	0

MODEL4FOUND	0
COLOR	0
COLORFOUND	1
EUR_PRICE	660.0
EUR_PRICEFOUND	1
MATCH	UNMATCH

The file containing these distance vectors has a few less rows than the file containing the matching and non-matching products pairs, as some pairs were discarded during calculating the vectors due to missing values. The final number of rows for this file is 536068.

5.2.2 ML classifiers

5.2.2.1 K Nearest Neighbors

This algorithm is one of the simplest [12]. For each new instance to be classified, the distances between this one and all of the examples contained in the training set are calculated. Then, the *k* closest examples based on the chosen distance metric are selected, so that the predominant class among these *k* examples will be the class for the instance to classify.

The distance metric used may vary according to the implementation of the algorithm. For example, *sklearn*[8], which only allows numeric values in the datasets, works by default with the Euclidean distance. This distance metric between two points *p* and *q* is defined as follows:

$$d(p,q) = d(q,p) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

Furthermore, in sklearn's implementation the *k* selected neighbor points can be given uniform (weight function "uniform" in Sklearn's implementation) or different weights (weight function "distance"), depending on their distance to the new example to be classified. If the second option is chosen, then the closest points receive higher weights than neighbours which are further away. This setting is configurable through the parameter "*weights*".

These two hyperparameters for the KNN model (the number of neighbours *k* and the weight function) were considered when performing the tuning. For this purpose, a Grid Search was used, consisting in generating several combinations of the values for the two mentioned parameters and choosing the pair that obtained the best mean accuracy when evaluating the model during the Cross-validation process. Thus, the selected values for the model training were the following:

- **Number of neighbours (*k*)**: 4
- **Weight function**: distance

Normalmente el K del KNN es impar. Yo repetiría los experimentos

K Nearest Neighbors is considered a *la*
model during a training phase and the
algorithm to calculate the distances to a
set. Because of this, if the training set increases over time, this constitutes a disadvantage regarding computational costs. However, Sklearn's implementation for the KNN algorithm provides a method `fit()`, making the model to be trained only once and erasing the need of "retrain" it for every new instance to classify. This is done using *kd-trees* or *ball trees*[9], which are data structures based on the training set that allow saving common operations for the new points to be classified.

[8] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
[9] https://stats.stackexchange.com/q/349863

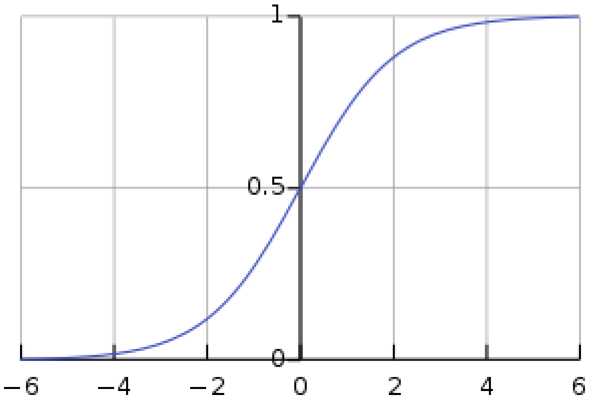
5.2.2.3 Logistic Regression

Despite its name, Logistic Regression is a method borrowed by Machine Learning from the field of statistics which is not used for regression problems, but for classification. The term “regression” is used because the idea behind this method is similar to the one used in Linear Regression, whereas “logistic” is taken from the ***Logit*** or ***Sigmoid*** function. In terms of computation, Logistic Regression is considered a very efficient algorithm.

Linear Regression used in classification problems may lead to a model which is very sensitive when there are **outliers** in the training data, so that is the reason why Logistic Regression comes in. In order to deal with outliers, Logistic Regression uses the Sigmoid function, which takes any real value between zero and one and it is defined as follows:

$$g(z) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}}$$

As can be observed in the graphic representation of the Sigmoid function, it is equal to 1 when *z* tends to positive infinity and equal to 0 when *z* tends to negative infinity.



Now, what Logistic Regression does is to consider as *z* the same linear function used by Linear Regression:

$$z = \theta_0 + \theta_1 x$$

So that the equation to be optimized in Logistic Regression becomes the following:

$$g(z) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}}$$

To estimate the best values for the coefficients of the former logistic function, several optimization algorithms can be used, such as *Gradient Descent* or *Stochastic Gradient Descent*.

When fitting the Logistic Regression model using Sklearn’s implementation[10], the hyperparameters to be tuned were:

- **C**: inverse of regularization strength; where small values imply strong regularization.

Falta justificar

- **Penalty**: penalty term used when applying regularization.

Using a Grid Search with 4-Fold Cross-validation, the best values for these two hyperparameter were:

- *C = 59.95*
- *penalty = “l1”*

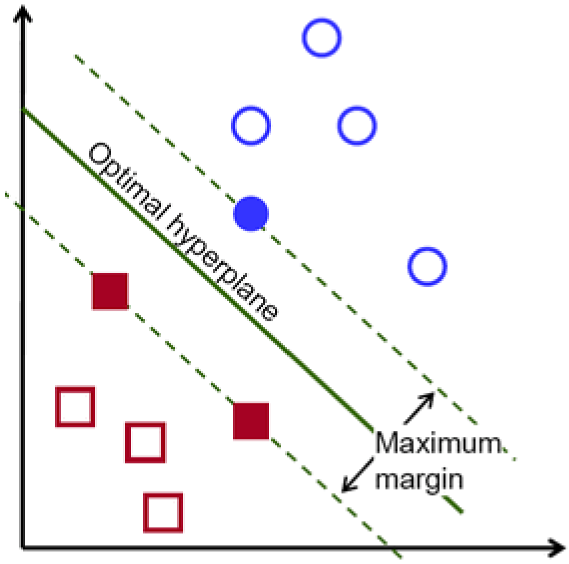
[10] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

5.2.2.4 Support Vector Machines

Support Vector Machines is a supervised learning model highly used for both classification and regression problems which was proposed by Vladimir Vapnik and Corinna Cortes in 1995 [13].

pondría que significan las siglas y entre paréntesis las siglas.

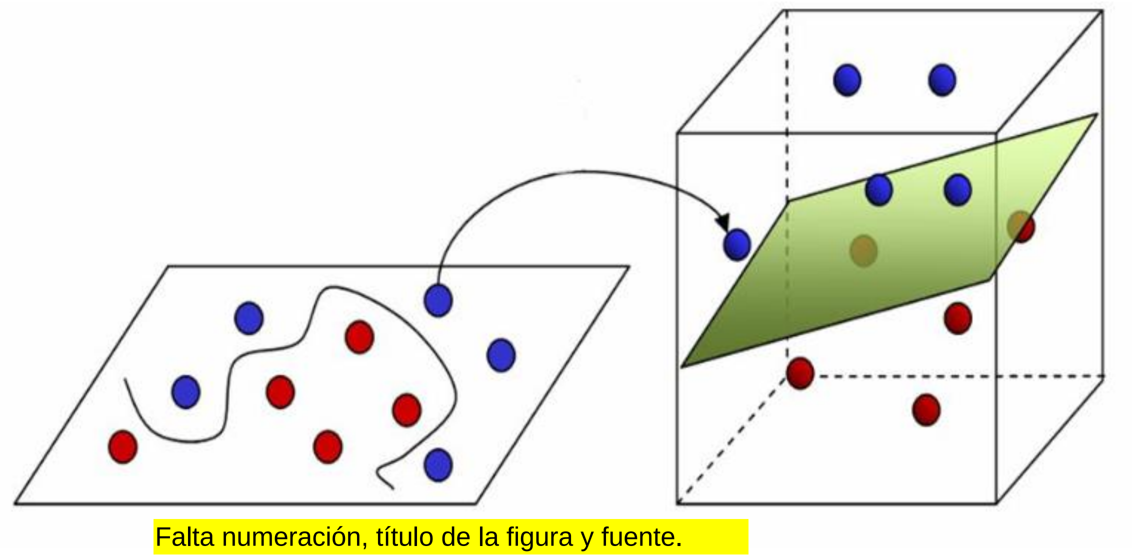
The idea behind SVM is to draw a decision boundary (formally called *hyperplane*) that best separate the points in a dataset based on the classes they belong. In a 2-dimensional case as depicted in the picture below, the hyperplane would be the straight line that best separate the classes represented by blue circles and red squares. In a 3-dimensional case, the hyperplane would be a plane.



Falta numerar, poner el título de esta

Here, the “support vectors” would be the points that define the margin for the optimal hyperplane, corresponding to the squares and circles in bold. This implies that actually only the support vectors in the training dataset are the points taken into account when defining the hyperplane, making the remaining points ignorable.

Commonly, there are cases when the classes are not linearly separable (i.e. there is no a line to separate classes in a 2-dimensional case nor a plane in a 3-dimensional case). For these cases, a function is used to transform the data into a higher dimensional space, so that they become now linearly separable. For example, the following picture shows how data in a 2-dimensional space which is not separable by a line is transformed into a 3-dimensional space, making them separable by a plane:



Falta numeración, título de la figura y fuente.

The main problem regarding these transformations is that they are computationally expensive most of the times.

Sklearn implements Support Vector Machines in the package `svm`[11], and the following two parameters were tuned when hyperparameter tuning was performed:

- **C**: penalty parameter of the error term. In other words, large values for this parameter makes to choose a small margin when defining the hyperplane, whereas small values will make the model choose a large margin.

- **Gamma**: it is the parameter for the Gaussian kernel. It is used when applying transformation to reach higher dimensional spaces when the data is not linearly separable.

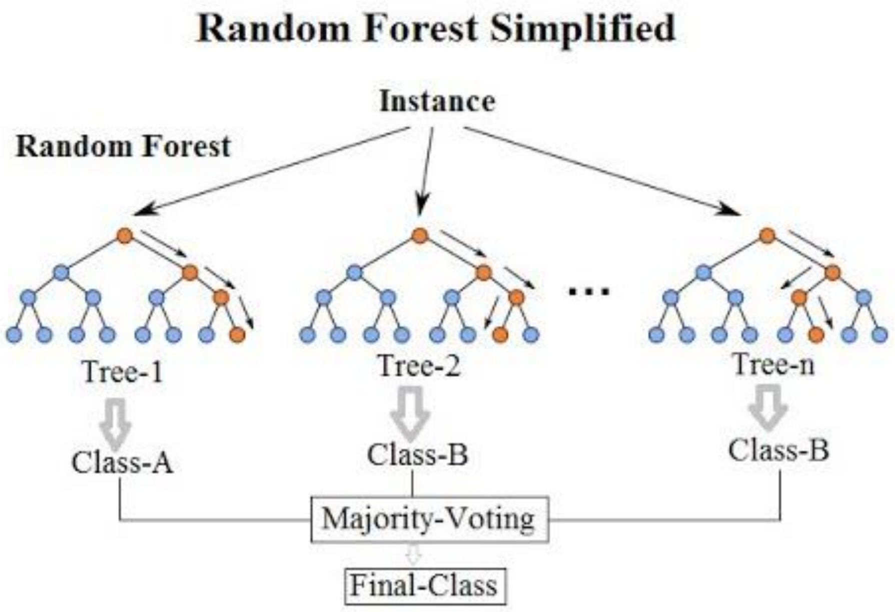
Using a Random Search with 4-Fold Cross-validation, the best values for these two hyperparameter were:

- *C = 450.08*
- *Gamma = 9.71e-06*

[11] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

5.2.2.5 Random Forest

A Random Forest, which is basically a collection of several decision trees, is one the most typical examples of ensemble methods using majority voting. Every decision tree that composes the Random Forest outputs a prediction, so that the mode of all the predicted classes by the decision trees (i.e. the most common prediction) is the final prediction of the Random Forest classifier.



For the Random Forest[12] classifier, the hyperparameter `n_estimators` in Sklearn's implementation, which is the number of decision trees that compose the forest, has been tuned. Again, using a Grid Search with 4-Folds Cross-validation, *n_estimators = 23* obtained the best results.

[12] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

5.2.2.6 Model evaluation

For each of the above-mentioned Machine Learning classifiers, a confusion matrix and four metrics have been calculated in order to evaluate the performance.

Specifically, for the problem being handled, the **confusion matrix** for each classifier will look as follows:

		PREDICTED CLASS	
		MATCH	UNMATCH
ACTUAL CLASS	MATCH	**True Positives** (TP): number of instances of class "MATCH" which were correctly labelled as "MATCH".	**False Positives** (FP): number of instances of class "MATCH" incorrectly labelled as "UNMATCH".
	UNMATCH	**False Negatives** (FN): number of instances of class "UNMATCH" incorrectly labelled as "MATCH".	**True Negatives** (TN): number of instances of class "UNMATCH" correctly labelled as "UNMATCH".

Regarding the results shown by the confusion matrix, it is important to emphasize that there is a type of error **much more critical** than the another one: if the classifier incorrectly predicts an UN*MATCH* when the actual label is *MATCH* (i.e. a False Positive), then the example could be revised by some data operator to verify it is actually a non-matching example; however, if the model incorrectly predicts a *MATCH* when the actual label is an *UNMATCH*, then the clients would be receiving wrong matching data, which constitutes a **very negative impact for their business**.

Thus, the four calculated metrics based on the obtained confusion metrics are defined as follows:

- **Accuracy**: it is the fraction of instances correctly classified.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision**: it is the fraction of positive predicted instances (predicted: *MATCH*) that are actually positive (actual: *MATCH*).

$$Precision = \frac{TP}{TP + FP}$$

- **Recall** or **Sensitivity**: it is the fraction of positive predicted instances (*MATCH*) correctly classified.

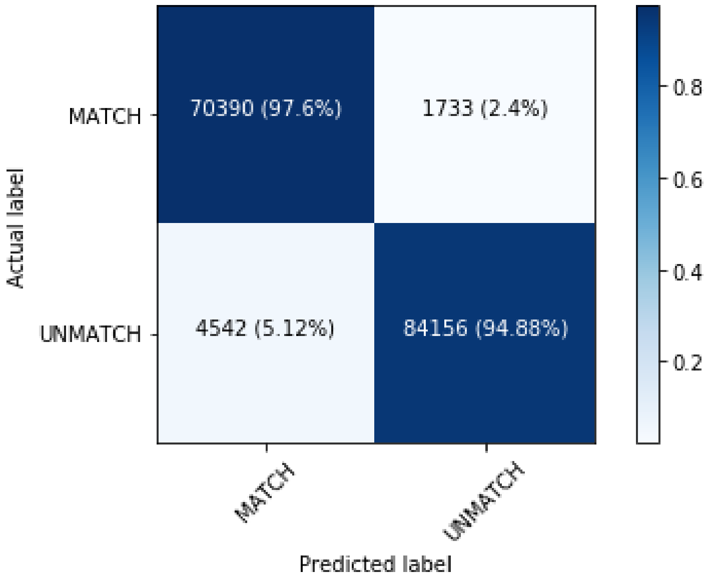
$$Recall = \frac{TP}{TP + FN}$$

- **F1 score**: it is the “harmonic mean” of the precision and recall.

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

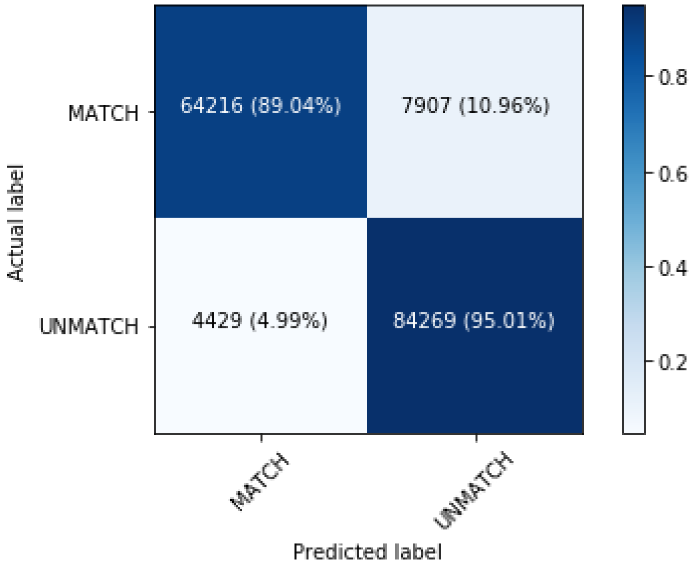
Below the obtained confusion matrix and metrics are shown for each classifier: kNN, Logistic Regression, Support Vector Machines and Random Forest. For all of these models, the training dataset was composed by the 70% of the generated dataset explained in 5.2.1 section, whereas the test set was composed by the remaining 30%.

K Nearest Neighbors



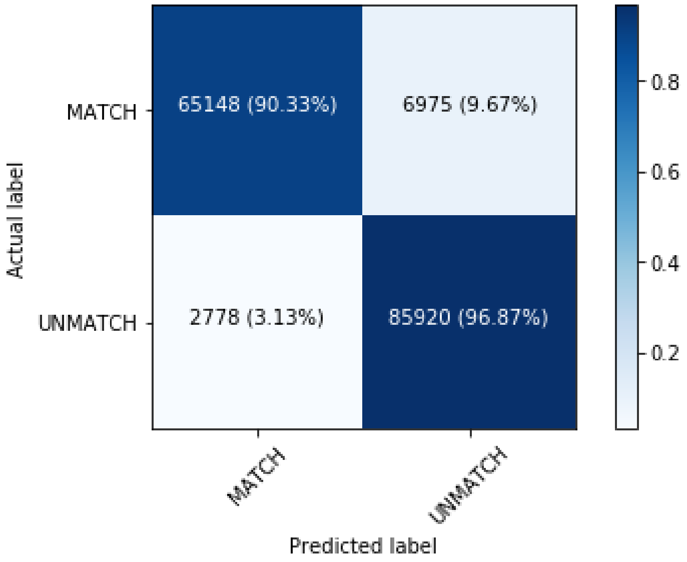
Accuracy: 96.1%
Precision: 93.94%
Recall: 97.6%
F1 score: 95.73%

Logistic Regression



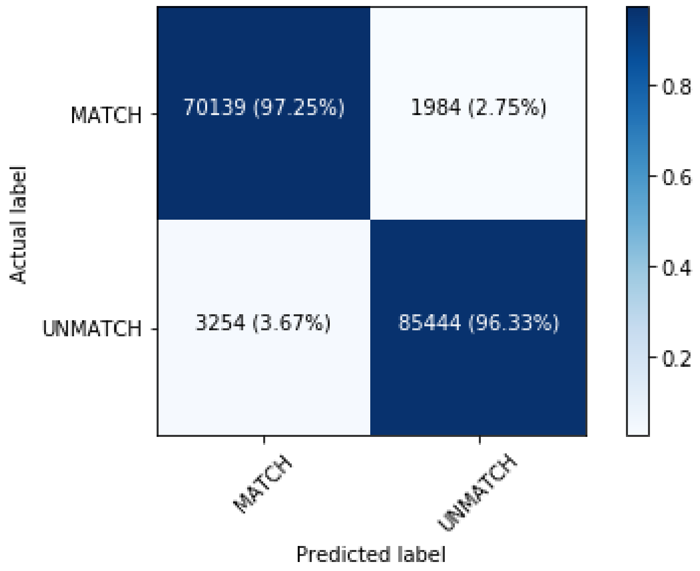
Accuracy: 92.33%
Precision: 93.55%
Recall: 89.04%
F1 score: 91.24%

Support Vector Machines



Accuracy: 93.94%
Precision: 95.91%
Recall: 90.33%
F1 score: 93.04%

Random Forest



Accuracy: 96.74%
Precision: 95.57%
Recall: 97.25%
F1 score: 96.4%

Results interpretation

From the performance results obtained by the four classifiers, the **Random Forest gets the best results** for all of the calculated metrics. It is followed by the **kNN** classifier, which performs almost as good as the Random Forest demonstrating that even a simple model gets very good results. Furthermore, attending to the type of error which is more critical in this problem domain (i.e. predicting a MATCH when the actual result is an UNMATCH), both models do it well enough.

Regarding the **Logistic Regression** and **Support Vector Machines models**, it is proven that neither of the two classifiers are the most suitable for this problem, not only for the obtained accuracy results but for the long time it took to fit them in comparison with the Random Forest and kNN models.

5.3 Model persistence

Model persistence refers to the capability of saving the status of a trained Machine Learning model, so that it can be later recovered and used without the need of being “retrained” again. This feature is very useful especially when, as in the case of this work, the generated ML models must be somehow integrated within productive applications (see next section: REST API).

Sklearn[13] enables this functionality through the use of two libraries: *pickle* and *joblib*. Here, *joblib* is used, as it is recommended to be used when the objects to be exported (in this case, the models) carry *numpy* arrays internally.

As the official Sklearn’s documentation suggests, all of the Machine Learning models generated throughout this work have been exported to files as a way of persisting them for later use. The following lines of Python codes correspond to the step of persisting the model:

```
import joblib
joblib.dump(clf, "filename.joblib")
```

Whereas the following lines are used to recover the persisted model:

```
import joblib
joblib.load("filename.joblib")
```

Thus, all the generated classifiers in this work can be found as joblib files inside the folder `src/_models`:

- **CRF**: *crf_smartphones.joblib*
- **kNN**: *knn_smartphones.joblib*
- **Logistic Regression**: *lr_smartphones.joblib*
- **Support Vector Machines**: *svm_smartphones.joblib*
- **Random Forest**: *rf_smartphones.joblib*

[13] https://scikit-learn.org/stable/modules/model_persistence.html

5.4 REST API

Once the Machine Learning models have been trained, they must be found available and ready to use somehow for the organization. One of the most common solutions to make it possible is the development of a REST API, so that a productive application can make requests to it and handle the received predictions in a convenient manner.

One of the main advantages of this solution is that it maintains a high level of independence between the code for the training of the Machine Learning models and the applications which are going to use them, even if the programming languages used at both sides are different. In this case, the Machine Learning models have been generated using *Sklearn*'s implementation, which is a library written in Python, whereas most of the already-existing productive applications used in the organization are written in C#. Using a REST API there is no need to use a single programming language for both parts.

The two concepts which a REST API consists of are the following: (1) an **API** (*Application Programming Interface*) is an interface with a set of methods that allows the communication with some part that acts as a server (for example, with a computer program or web site); whereas **REST** defines the communication protocol used, which is HTTP.

Pondría, lo que significa el acrónimo REST, creo que es r

A REST API works using URIs (*Uniform Resource Identifiers*) to access the resources a server exposes. As an example, the well-known software development platform GitHub has a REST API available to make requests. It can be used, for instance, to retrieve information about a specific user:

```
https://api.github.com/users/tteguayco
```

In this case, the response is returned as a JSON object:

qué significa el acrónimo JSON?

```
{
  "login": "tteguayco",
  "id": 20015750,
  "node_id": "MDQ6VXNlcjIwMDE1NzUw",
  "url": "https://api.github.com/users/tteguayco",
  "html_url": "https://github.com/tteguayco",
  "type": "User",
  "site_admin": false,
  "name": null,
  "company": "Universitat Oberta de Catalunya",
  "blog": "",
  "location": "San Cristóbal de La Laguna",
  "hireable": true,
  "bio": "Software Developer & Data Science student",
  "public_repos": 23,
  "followers": 15,
  "following": 29,
  "created_at": "2016-06-18T15:56:28Z",
  "updated_at": "2019-04-06T15:34:20Z",
  ...
}
```

5.4.1 Implementation

The implementation of the REST API to deploy the generated Machine Learning models has been done with *Flask*[14], a "microframework" for Python that allows the creation of web services easily. The source code containing this implementation can be found on the folder `ml_api_rest`.

There are two resources to make requests to:

- `/predictFeaturesFromTitle`
- `/predictProductsMatching`

On one hand, the resource `/predictFeaturesFromTitle` expects an argument `title`, which contains a product title for the CRF trained model to predict its features. The JSON response contains two fields: `title`, a string with the original provided product title in the API call; and `features`, a list containing a structure with the BIO encoding and confidence levels assigned to the tokens founds by the CRF model.

On the other hand, the resource `/predictProductsMatching` receives some arguments containing data about two products to decide whether they match or not. These arguments are:

- `product_title1`: title of the first product.
- `product_title2`: title of the second product.
- `product_price1`: price of the first product.
- `product_price2`: title of the second product.
- `product_currency1`: currency of the first product. Only admits "EUR" and "CRF" as values.
- `product_currency2`: currency of the second product. Only admits "EUR" and "CRF" as values.

Internally, the app loads the pre-trained CRF model to make predictions in both resources. In the case of the second resource, the predictions of the Random Forest are used, as it was the classifier that best prediction results obtained.

5.3.2 Deployment

The developed REST API is deployed on *Heroku*[15]. This cloud platform as a service (*PaaS*) offers free plans for developers to deploy their apps or use on-

[14] http://flask.pocoo.org/

[15] https://www.heroku.com/

cloud services, such as databases. The deployed API can be found through the following URI:

```
https://product-matching-tfm-api-rest.herokuapp.com/predictProductsMatching
```

5.3.3 Examples

Requests to the REST API deployed on Heroku can be made in many ways. For example, using **cURL**[16] or from a Python script. cURL is a well-known CLI (*Command Line Interface*) tool used to transfer data using URLs.

5.3.3.1 Features from title prediction

The following example illustrates how to use cURL to make a request to the resource `/predictFeaturesFromTitle` deployed in the API, in charge of calling the CRF model to make a prediction of smartphone features on the provided title:

```
$ curl -X GET https://product-matching-tfm-api-
rest.herokuapp.com/predictFeaturesFromTitle -d title="Apple
iPhone 4S Plus 6GB RAM Black"
```

The JSON object returned as a response is the following (truncated for a better visualization):

```
{
   "title":"Apple iPhone 4S Plus 6GB RAM Black",
   "features":[
      [
         "Apple",
         "B-BRAND",
         {
            "confidence":{
               "B-BRAND":100.0,
               "B-MODEL":0.0,
               ...
            }
         }
      ],
      [
         "iPhone",
         "B-MODEL",
         {
            "confidence":{
               "B-BRAND":0.0,
               "B-MODEL":100.0,
               ...
            }
```

[16] https://curl.haxx.se/

```
         }
      ],
      [
         "4S",
         "I-MODEL",
         ...
      ],
      [
         "Plus",
         "I-MODEL",
         ...
      ],
      [
         "6GB",
         "B-RAM",
         ...
      ],
      [
         "RAM",
         "O",
         ...
      ],
      [
         "Black",
         "B-COLOR",
         ...
      ]
   ]
}
```

As can be observed, for each word contained in the sent title, the label with the highest confidence level is assigned to it. Furthermore, for each label known for the CRF model, the corresponding confidence is also returned.

5.3.3.2 Matching prediction

For the resource `/predictsProductMatching`, data of two products is sent in order to decide if they match or not. The following Python script makes the request resource passing the needed parameters containing the information about the products to decide whether they match or not.

```python
import requests
import json
import pprint

url = "https://product-matching-tfm-api-rest.herokuapp.com/predictProductsMatching"

```

```
matching_pair = {
    "product_title1": "ZTE Blade V8 Lite Negro",
    "product_title2": "ZTE Blade V8 Lite Smartphone de 5\" 2 GB RAM Oro",
       "product_price1": 139.0,
       "product_price2": 107.0,
       "product_currency1": "EUR",
       "product_currency2": "EUR",
   }

   response = requests.get(url, matching_pair)
   pprint.pprint(response.json())
```

When executing the script, the following JSON object is received:

```
{
   'confidence': 1.0,
   'prediction': 'MATCH',
   'product1': '{"brand1": "zte", "brand2": "", "brand3": "", "color": "negro",
             "eur_price": 139.0, "gb_ram": -Infinity, "model1": "blade",
             "model2": "v8", "model3": "lite", "model4": ""}',
   'product2': '{"brand1": "zte", "brand2": "", "brand3": "", "color": "oro",
             "eur_price": 107.0, "gb_ram": -Infinity, "model1": "blade", '
             '"model2": "v8", "model3": "lite", "model4": ""}'
}
```

5. Source code

Inside the attached folder `src`, the following structure of subfolders can be found:

- **`_data/`**: contains both the dataset provided by the collaborator company (`smartphones_offers.csv`) and all the data generated during among the steps of the development of the work, detailed throughout this document.

- **`_models/`**: contains all the persisted models generated as *joblib* files.

- **`crf-titles-bio-tagger/`**: *BIOTagger* codification, object that generated the training dataset for the CRF model.

- **`smartphones-db-crawler/`**: code for the scraper used to crawl the website *GMSArena*.

- **`crf-model-training`**: contains the *Jupyter* notebook used to train the CRF model.

- **`distance-vectors-matching-ds-generator/`**: contains the scripts corresponding to the steps for generating the training dataset for the machine learning classifiers explained in section 5.2.1.

- **`sklearn-ml-classifiers-smartphones/`**: *Jupyter* notebooks containing the hyperparameter tuning and training of the machine learning classifiers.

- **`ml_api_rest/`**: REST API development written with *Flask*.

- **`common/`**: code used in more than one script.

- Files in the root folder: the files `Procfile`, `requirements.txt` and `runtime.txt` are used to deploy the API to Heroku.

All of the developed scripts have been written using the ***Python*** language.

6. Conclusions

The complex and real-world problem of product matching is addressed for the case of smartphones and the obtained results may be considered good enough to use the generated ML models in a real environment. In the case of the e-Commerce aggregator developed by the organization which contributes to this work by providing the products data, integrating the use of the REST API to automate the matching of smartphones may lead to save huge consumptions of time and human efforts.

Nevertheless, since the complexity of the problem is given mainly due to the high feature's heterogeneity present in the different types of products the e-Commerce aggregator handles (i.e. amongst others, home supplies, computers, tablets, kitchenware products, love toys, movies, etc), time constraints have made this work only focuses on smartphones (which are still one of the most important type of products for the company). Thus, the idea of covering more types of products included in the planning is not met but will obviously have to be the main priority for future work.

Concerning the development of the work itself, the generation of the CRF model must be emphasized. Its use is vital to extract the smartphone's features from unstructured text for later use when making matching predictions, since these features are rarely found as structure data in the sources the products are crawled from. Furthermore, this type of model has represented additional efforts of research because it has not been covered by the contents of these Master's studies.

With regard to the matching classifiers, Random Forest and kNN gave the best results against Logistic Regression and Support Vector Machines, where these latter models also took longer during the training phase. It was also stressed that during the matching prediction task between a pair of products, the error of predicting a match when the actual result is an unmatch leads to a higher negative impact for the business, since the service provided by the e-Commerce aggregator losses an important point of quality.

Finally, as an added value for this work, a REST API was developed with the aim of facilitating the embodiment of the machine learning models in the organization's productive applications. This API can receive requests from applications that can be even written in languages different from Python and has been set publicly available and ready to use on the on-cloud service provided by Heroku.

7. Future work

As noted in the conclusions above, one of the main tasks to focus on as a future work is the capability to handle more different types of products when performing the matching predictions. This necessarily induces to be able to extract more features than the covered in this work for the smartphone's case (i.e. brand, model and color), since depending on the type of product, other features may be more important. For example, for a computer, important features may be the brand and the model as well as smartphones but, in the case of a music album, the name of the album and the author would be the most relevant features to identity in order to perform the matching.

In addition, the results obtained in this work can be refined, as there are some points that need wider development and research. On one hand, the colors presented in the provided dataset can be found in three different languages: German, English and Spanish. Somehow, every name of a color should be translated into the same language so that, when calculating the distance vectors seen in section 5.2.1, two colors that are the same but written in different languages should be considered as equals and, consequently, the generated degree of similarity for them should be 1. These translations can be done by the use of `googletrans`[17], a Python library which implements the Google Translate API.

On the other hand, an analysis study on the probabilistic estimators thrown by the ML classifiers should be done in order to check their plausibility. If these estimators are credible enough, they could serve to avoid the system make wrong predictions (especially those with higher impact on the business) by discarding the predictions with low probabilities at a certain threshold. It will be important here to keep in mind that the definition of higher thresholds to discard predictions will lead the system make less mistakes but at the expense of saving less human effort.

[17] https://pypi.org/project/googletrans/

8. Glossary

Cross-validation: model validation technique which consists in splitting the training data in different partitions, so that each partition is used as a “validation” set in each iteration of the process. The number of partitions or folds is commonly denoted by *k*.

e-Commerce aggregator: system that brings together product offers from different third-party sellers in order to facilitate price comparison. *Google Shopping* or *Idealo* are some of the most famous existing e-Commerce aggregators.

European Article Number (EAN): 13-digit product identifier in Europe. It is a standard compatible with UPC, such that adding a leading zero will result in the corresponding UPC code. Japan has its own version of EAN codes called JAN (*Japan Article Number*), which uses different digits making them globally unique.

Global Trade Item Number (GTIN): global product identifier which consists of 14 digits. It is compatible with both EAN and UPC codes, so that adding one and two zeros at the beginning of any EAN or UPC codes respectively will lead to the corresponding GTIN.

Hyperparameter Tuning: in Machine Learning, it is the process of finding the optical set of values for the hyperparameters of a certain model.

Price Search Engine: very similar concept to an e-Commerce aggregator but with the slightly difference that a Price Search Engine allows purchases. Amazon is well-known PSE.

Universal Product Code (UPC): global product identifier which is commonly used in the USA, Canada, United Kingdom and Australia. It is composed of 12 digits.

Usas muchos acrónimos en tu memoria, haría el Glosario más largo

9. Bibliography

1. Andreas Thor. (2010). Toward an adaptive string similarity measure for matching product offers. In GI Jahrestagung (1), pages 702–710.

2. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J. (2013). Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems. pp. 3111–3119.

3. Pennington, J., Socher, R., Manning, C.D. (2014). Glove: Global vectors for word representation. EMNLP. vol. 14, pp. 1532–1543.

4. Grbovic, M., Radosavljevic, V., Djuric, N., Bhamidipati, N., Savla, J., Bhagwan, V., Sharp, D. (2015) E-commerce in your inbox: Product recommendations at scale. In: Proceedings of the 21th ACM SIGKDD.

5. Petrovski, P., Bryl, V., Bizer, C. (2014). Learning regular expressions for the extraction of product attributes from e-commerce microdata.

6. Svingen, B. (1998) Learning regular languages using genetic programming. In Genetic Programming 1998: Proceedings of the Third Annual Conference.

7. Morgan Kaufmann, M. 1998. Alberto Bartoli, Giorgio Davanzo, Andrea De Lorenzo, Marco Mauri, Eric Medvet, and Enrico Sorio. Automatic generation of regular expressions from examples with genetic programming. In Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO ’12, pages 1477–1478, 2012.

8. Ajinkya More. (2016). Attribute Extraction from Product Titles in eCommerce. WalmartLabs, Sunnyvale CA.

9. Petar Ristoskia, Petar Petrovskia, Peter Mikab, and Heiko Paulheima. (2017). A machine learning approach for product matching and categorization. Semantic web.

10. Mikhail Sidorov. (2018). Attribute extraction from eCommerce product descriptions. CS229.

11. Charles Sutton, Andrew McCallum. (2012). An introduction to Conditional Random Fields. Fundations and Trends in Machine Learning. Volume 4 Issue 4, April 2012. Pages 267-373.

12. Gironés, J., Casas J., Minguillón, J., Caihuelas R. (2017). Minería de datos: modelos y algoritmos. Editorial UOC.

13. Vapnik, V. Cortes, C. (1995). Support-Vector Networks. Kluwer Academic Publishers.

14. Roller R, Stevenson M (2015) Making the most of limited training data using distant supervision. In: 2015 workshop on biomedical natural language processing (BioNLP 2015), Beijing, pp 12–20.