Thesis
Relation Extraction Methods for Learning Models and Product Matching in on-line Shops

Michael Voutas

October 15, 2017
## Contents

1 Introduction ........................................ 2

2 Data set ............................................... 3
   2.1 Data description .................................. 3
   2.2 Data selection .................................... 6
   2.3 Data preprocessing .................................. 9

3 My Approach ......................................... 11
   3.1 First model ....................................... 12
      3.1.1 Product Categories Classifier ............... 13
      3.1.2 Radius Nearest Neighbors ..................... 16
      3.1.3 Evaluate product classification ............... 18
   3.2 Second model ..................................... 21
      3.2.1 Detect first model classification issues .... 21
      3.2.2 Normalize quantity dissimilarities .......... 25
      3.2.3 Revise brand names misquotations .......... 27
      3.2.4 Evaluate product classification ............... 31

4 Conclusions .......................................... 34

5 Future Work .......................................... 36

6 Technical Appendix .................................. 38
Chapter 1

Introduction

This project is a capstone project for the Master of Data Science at Athens University of Economics and Business. My aim was to apply machine learning techniques in the field of product matching. Moreover, I cooperated with the eRetail team of Convert Group company to develop my solutions as they have already applied their algorithm for product matching and they want to improve its performance.

More specifically, I created a model that classifies the product’s sales from on-line shops, using text mining. My experimental implementation included a combination of Radius Nearest Neighbors algorithm and Logistic Regression classifier as well as auto correction techniques with regular expressions and string functions.

The current company’s algorithm takes as input the e-shops sales and extracts the products unit using multiple features as product name, SKU (stock keeping unit) and price. Then it classifies each input product to their database products or puts it to an unsupported dataset if the accuracy is under a defined boundary which had set at 90%. So a new metric arises, the “matching rate” that represents the percentage of input products that can be classified over the intended accuracy. Their matching rate is at 75% and my aim is to increase it, without the accuracy falls under the level of 90%.

My model combines different techniques and shoots up the matching rate to 86% with the accuracy remains at 90% or up to 96% if the total accuracy is limited to 88%.

To develop my system I used the Python 3 programming language with pandas[1], numpy[2], matplotlib[3] and scikit-learn[4] libraries to manage the data and apply machine learning algorithms.
Chapter 2

Data set

The data sets that company granted to me, consist of information that distributed at 8 separated tables. The format of them is csv and each table has fields that are connected or referred to other tables of data set. Furthermore, the company has split the data to training and test sets in order to check the over-fitting of my model and be a fair comparison to our measures.

2.1 Data description

The relation graph below, pictures the tables and the connections between them. Each table has features which describe their records. At my experiment applications, I joined some of them on Product table in order to manage the available informations.

Barcoded Products:
This table contains as records 4,122 unique products that have been inserted to the database. Each record has the following attributes:

- **id**: A unique key number. There are 4,122 unique values.
- **name**: A custom text that describes the product and created by Convert Group. It is a summarization of product names that came from matched products of the specific barcoded product. There are 4,059 unique values.
- **barcode**: A text that used as product barcode from convert group. There are 3,636 unique values and could be null.
• category id: A number that refers to id key of table Categories. There are 60 unique values.

• brand id: A number that refers to id key of table brands. There are 278 unique values.

• system name: A text that describe the product and created by convert group. There are 3,695 unique values and could be null.

• product line: A number that refers to id key of product lines table. There are 1,014 unique values.

Categories:  
This table contains as records 60 unique product’s categories. Each record has the following attributes:

• id: A unique key number. There are 60 unique values.

• name: A text that represents the category. There are 60 unique values.

Brands:  
This table contains as records 277 unique product’s brands. Each record has the following attributes:

• id: A unique key number. There are 277 unique values.

• name: A text that represents the brand. There are 277 unique values.

product lines:  
This table contains as records 1,014 unique product lines. Each record has the following attributes:
• **id**: A unique key number. There are 1,014 unique values.

• **name**: A text that represents the product line. There are 1,011 unique values and could be null.

**Products(matched):**
This table contains as records 38,404 products that have been inserted to the database. Each product is matched only to one barcoded product and a barcoded product can be matched to multiple product records. Their attributes are:

• **id**: A unique key number. There are 38,104 unique values.

• **name**: A text that represents the site name and created by each e-shop. There are 36,012 unique values.

• **title**: A text that represents the name of e-shop. There are 49 unique values.

• **barcoded product id**: A number that refers to id key of table Barcoded Products. There are 3,841 unique values.

• **sku**: A number that refers to stock keeping unit number and has been field by the corresponding e-shop. There are 3,841 unique values and could be null.

**Filters:**
This table contains as records 155 product’s filters. These are some features that developed by Convert Group team and each barcoded product could be described with multiple filters. Each record has the following attributes:

• **id**: A unique key number. There are 155 unique values.

• **name**: A text that represents the filter. There are 146 unique values.

• **filter group id**: A number that refers to id key of table Filter Groups. There are 19 unique values.

**Filter Groups:**
This table contains as records, 19 unique product lines. Each record has the following variables:

• **id**: A unique key number. There are 19 unique values.

• **name**: A text that describes the filter group. There are 19 unique values.
Barcoded Products Filters:
This relational table connects the Filters with the Barcoded Products. Each barcoded product can be matched with many filters. The record has the following variables:

- **filter id**: A number that refers to id key of table Filter. There are 155 unique values.
- **Barcoded Product id**: A number that refers to id key of table Barcoded Products. There are 2,622 unique values.

### 2.2 Data selection

There is no doubt that there are plenty of data that I could combine. Also it is very important that there are 2 different type of sources that describe the products. The first is the e-shop sites that have posted their products and have given features as Products.name, Products.title, Products.sku. The second is the Convert Group’s eRetail team that has analyzed the data, classify the products to Barcoded_Products and recognize the Brands, the Categories, the Product Lines and the Filters per barcoded product.

My aim was to select the suitable data attributes, combine them and create an efficient algorithm with highly accuracy to predictions. After data analysis and tries I concluded to use **Products.name, Barcoded_Products.category, Barcoded_Products.brand, Barcoded_Products.product_line**

The key attribute that contains the main information of input products is the **Products.name**. To be more specific the Products.name is a string value that each e-shop defines per product. For example:

- **STC Hydration Eye - gel 20ml**
- **Olivia Body Butter Cotton, Κρέμα Σώματος Παχύρευση με Έλαιολάδο - Αμυγδαλέλαιο που Μαλάκωνε-Ενυδατώνει, 200ml**
- **Korres Black Pine - Μαύρη Πεύκη Κρέμα Ματιών 15ml**
- **2 x Synchroline Nutristime Plus, ΠΑΚΕΤΟ (1+1 ΔΩΡΟ) 50 ml x2**
- **‘SYNCHROLINE HYDRA PLUS FACE CR ΕΝΥΔΑΤΙΚΗ ΚΡΕΜΑ ΠΡΟΣΩΠΟΥ 50 ML + ΔΩΡΟ SYNCHROLINE HYDRATime TONIC LOTION ΕΝΥΔΑΤΙΚΗ ΤΟΝΩΤΙΚΗ ΛΟΣΙΟΝ 125 ML’**
- **LIERAC+HOMME+ANTI-FATIGUE+50ML**
- **SYNCHROLINE – ROSACURE INTENSIVE EMULSION SPF30 – 30ml**
As we can see there is very compressed information inside the product name but we can identify many similarities at products of the same barcoded group.

- *Bepanthol Shampoo Greasy Σαμπουάν Καθημερινής Χρήσης Λιπαρά Μαλλιά 200ml*
- *Bepanthol Greasy Σαμπουάν για Λιπαρά Μαλλιά 200ml*
- *Bepanthol Shampoo GREASY, 200ml : Σαμπουάν για λιπαρά μαλλιά, με προβιταμίνη B5, βιταμίνη E, για θρέψη, αναδόμηση και προστασία της τρίχας*

- *A-Derma Exomega Creme Emollient Μαλακτική Κρέμα για Ατοπικό Δέρμα 400ml*
- *ADERMA - Exomega Crème Emollient — 400ml*
- *A-derma Exomega Creme Emolliente 400ml Νέα Γενιά , Κρέμα για ατοπικά δέρματα*

- *BIODERMA SEBIUM GEL MOUSSANT 200 ML & ΔΩΡΟ SEBIUM GEL MOUSSANT 100 ML*
- *Bioderma Sebium Gel Αφρώδες Τζέλ Καθαρισμού 200ml +100ml*
- *BIODERMA Sebium Gel Moussant - 200 ml + 100 ml ΔΩΡΟ!*

An important problem that occurs in product name is that the content is multilingual. As I want to detect the languages and calculate the average percentage of them per product name, I used the langdetect library[5][6] of Python. Then I visualized the results:

![Languages in Product Names](image)

*Figure 2.2: Languages percentages per product name*
The graph shows up that the majority of product names’ words is in Greek (50%), following second the English (25%) and other languages are lower to the average of 1%. So I can’t use any stemer in the preprocessing procedure as the content is multilingual. Furthermore a statistic analysis shows that the product name length is in average 10 words with max and min values 54 and 1 respectively.

The second given feature that I used, is the information of Barcoded Products. category. There are 59 distinct categories and some examples are:

- Anti-Reddening Products
- Liquid Soaps
- Shampoos for Coloured hair

As I tried to see the frequency of each category, I joined the table Products to Barcoded Products and then I created the frequency graph:

![Figure 2.3: Number of Products per Category](image.png)

There is no doubt that the categories are unbalanced but my experimental results prove that it is not a crucial problem. So in my model I trained an algorithm to detect the product’s category as I want to get fewer candidate neighbors.

The third attribute that was used in my model, was the **barcoded_products. brand.** In industry a brand is “A type of product manufactured by a particular company under a particular name”[7]. Furthermore, a company may manage many brands and each brand may has many product lines. However there are companies that use their name as brand as well. Furthermore it was proved
in my experiments that the brand name inside a product name can get closer products from the same barcoded category.

Some Brand names are:

- *Dettol*
- *Korres*
- *Versace 19V69*

Moreover to predict more accurately the brand name, the `Barcoded_Products. product_line` was a key variable. As I mentioned before, many product lines can belong to only a single brand and also there are no overlapping to product lines. Also, it is very important that the product line is included in the majority of product name. Some pairs of brands and product lines are the following:

- *Dettol*: 'DETTOL', 'NO TOUCH'
- *Korres*: 'KORRES SOAPS', 'KORRES BODY CARE', 'BEAUTY SHOTS'
- *Versace 19V69*: 'PREMIUM CAVIAR', 'ITALIA'

### 2.3 Data preprocessing

The main format of data that I had to manage is text. As we saw above the product name is a string of 1-54 words that describes the product. For higher accuracy to my experiments, I applied some preprocessing functions to normalize the data and show up their similarities.

- Replace the punctuations with spaces (except for '%').
- Lower cased the words.
- Remove accents.
- Remove English and Greek stop words.
- Remove white spaces.

The above rules for data preprocessing applied after some evaluations of my experimental results. For example the "%" symbol should not be removed from product descriptions, as it helps to be recognized the discounted products. Also the choice to replace the punctuations with space instead of remove them,
was arised as some product names contain the ’+’ symbol as escape character to strings in place of space(e.g. “LIERAC+HOMME+ANTI-FATIGUE+50ML”). Moreover, I removed the accents as some Greek product names have been written with accents and some without. Finally as the majority of product names are in English or Greek, I used scikit-learn’s lists of stop words and I removed them. The preprocessing function was applied on product names, brands and product lines.
Chapter 3

My Approach

My system should be able to classify the input e-shop’s products with highly accuracy and limited false positives. It is crucial to deter the high precision but also improve the existing matching rate. So my model ought to be able to decide when it is sure about a barcoded prediction and set a label to an input product or it is uncertain and keep the product unlabeled. Additionally, I have to consider that the barcode classes are unbalanced. This is a problem that I have to take into account to select the suitable classification algorithm.

![Barcoded products frequency in matched products](image)

Figure 3.1: Barcoded products frequency in matched products

As the plot is pictured from lower to higher frequency, we can realize that in average an input product (test product) has only 15 other similar products in the database (training products). So some effective classification algorithms
such as linear classifiers, support vector machines or decision trees are unable to separate 38,000 classes with mean support 16 instances per class.

My experiments were leaded me to Nearest Neighbors classifiers. These classifiers are suitable as I want to find similar (“neighboring”) products for an input product into the training set. Also they are ideal, not only because the classes are too many to separate by hyper levels but also because that the nearest neighbors’ complexity at train and predict processes doesn’t depend on the number of distinct classes. On the other hand, the most serious drawback at process of Nearest Neighbors execution is the high demands at memory[10] as more training instances, more physical memory is needed to fit the algorithm.

Furthermore, I have to find a way to initialize and calculate a matching function. This function ought to be able to decide if the Nearest Neighbors can predict with high accuracy for an input product or the system has to let it unclassified. In other words, I have to set up a mechanism for the “matching rate” which is used by Convert Group at its algorithm.

3.1 First model

My first implementation consists of two-layer classification. At the first layer, my model predicts the Product.Category for an input product and at the second creates a nearest neighbors classifier with the predicted category’s matching products as training set.

![Figure 3.2: My two layer machine learning model](image-url)
3.1.1 Product Categories Classifier

At the first classification layer, I try to find out the product category in order to reduce the neighbors at the second classification layer and the demands for memory.

The features for the first classification layer extracted from the product name and I used a combination of 1, 2 and 3 grams. The classifiers[8] that I tested are the Logistic Regression, Decision Tree and a K Nearest Neighbors. The hyper parameters tuned with Grid Search[9] processing.

```python
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

param_grid = {'C': [1, 10, 100]}
grids_cv = GridSearchCV(LogisticRegression(), param_grid)
clf_est = grids_cv.fit(X_train[:,], Y_train[:])
c_par = (clf_est.best_params_['C'])
clf = LogisticRegression(C=c_par)
```

The above code uses the GridSearch scikit-learn’s class to create an object for exhaustive search over specified parameter values. Firstly, I declared a set of values (param_grid) for the classification estimator and then I run the fit method with all possible parameters. Finally the best_params is the attribute that contains a dictionary with the best estimated values for the classifier parameters.

The Logistic Regression ‘C’ parameter that represents the inverse of regularization strength set to C=10. At K - Nearest Neighbors the K and leaf size were estimated at 3 and 1 respectively. Also the only parameter of Decision Tree that tuned was the minimum number of samples that is required at internal nodes splitting and set to 2.

<table>
<thead>
<tr>
<th>Type</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNN</td>
<td>0.84</td>
<td>0.83</td>
<td>0.83</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>0.91</td>
<td>0.88</td>
<td>0.89</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.96</td>
<td>0.95</td>
<td>0.95</td>
</tr>
</tbody>
</table>

Table 3.1: Evaluation metrics for Product.Category classifier

From the table 3.1 we can conclude that the Logistic Regression classifier has better performance than the other two classifiers. The evaluation metrics of precision recall and their harmonic average f1 at the Logistic Regression are considerable the best. All the metrics above calculated on training set using 10 fold cross validation[11] with the purpose to prevent the over-fitting to test set.
After I chose the Logistic Regression as suitable classifier, I visualized two important curves. The first is the ROC(receiver Operating Characteristic)\cite{12} curve which plots the true positive rate in function of the false positive rate for different cut-off points. Also the area under the ROC curve (AUC\cite{13}) measures the classifier prediction accuracy and it calculated as well for the micro and macro average of ROC curve. The only different between the two metrics is the calculation of false and true positive average. The true and false positive average values are calculated per instance at micro average method and per category at macro average.

![Figure 3.3: Multiclass average Roc curve for Products.Category Classifier](image)

It is clear from the above diagram that the prediction accuracy of Logistic Regression is extremely high. The true positive rate climbed at 0.9 as the false positive rate took values greater than 0 and stabilized at 1 as the false positive rate got closer to 1. In addition, the micro and macro average AUC touched the 0.98 and 0.96 respectively.

The second is the Precision - Recall curve that helps us to declare the level of precision as the recall increase. In other words, the increasing rate of false positive when the false negatives are reduced.
As can be seen from the graph, the average precision remained at high level as the recall rose from 0 to 1. Also the area under the graph covered the 91% of possible area between precision and recall.

Moreover, I used the psutil\cite{14} in order to calculate the requirements for physical memory at training and testing processes. It is a useful package with many implementations for CPU, memory, disk or network monitoring during a python program execution. The memory output was in byte so I divided it by $2^{20}$ to convert it to megabytes.

So it reported that the pick of memory demand was 180MB for the training of Logistic Regression with 38,404 instances with 70,818 features and C=10. Furthermore, the duration of training was 63.92 sec. The training model saved local in order to be used for faster predictions. The prediction process lasted less than 1 second for the 12,417 testing instances.
3.1.2 Radius Nearest Neighbors

After the process of prediction product’s category, we are at a space with significant fewer neighbors, as the Category with the biggest amount of products has 3000 items (figure 2.3) approximately. At this step, I chose the Radius Nearest Neighbors\[15\] \[16\] to classify the input products. This algorithm permits to set a radius in which an input product is closer to a group of training objects. So it determines a neighborhood of candidate barcoded labels.

Furthermore, it is important that the neighboring labels are distance weighted. So the closer neighbor, the more affect to the given barcoded label. This is a way to overcome precision faults that could appears at K-NN(K Nearest Neighbors) algorithm. At traditional K-NN implementations the majority decides the output label and the distance between testing and training objects doesn’t affect the calculations. Maybe it is not a problem as K set from 3 to 7 but as it takes greater values than 9, the accuracy could fall off. In our occasion, we want to declare a region that possibly consists of 20 or 30 training objects and at extreme situations each neighbor could came from different barcode class. So a weighted calculation is required. Hence the closer objects to an input product will be more influential than the further to the final classification decision.

Besides accuracy profit, the Radius Nearest Neighbors algorithm provides the ability to let an input object unlabeled, if there is no neighbor inside the defined radius. This is ideal as I want to initialize a high precision matching function that will be able to determine “uncertain” predictions at testing inputs. In other words, I will be able to calculate the support rate of nearest neighbors’ predictions with this function that it corresponds to matching rate of Convert Group’s algorithm.

At the experiment process, I tuned the radius value, defined the suitable distance metric and then I used the Radius Nearest algorithm to predict the barcoded values on test set.

```python
class sklearn.neighbors.RadiusNeighborsClassifier(radius=mraduis,
algorithm='brute', metric='cosine', weights='distance',
leaf_size=1, n_jobs=-1, outlier_label='NONE')
```

The scikit-learn class above, is the Radius Neighbors implementation that I executed in my experiments. In order to tune the metric and algorithm parameters I used a grid search function as I described at Category classifier. Then I got the “brute” and “cosine” as the best estimated values. Moreover the weights parameter set to ‘distance’ and the outlier_label to ’None’ in order to apply the matching process which was analyzed above. The ‘None’ is the label that gets any testing product that its distance from any training product is longer than the defined radius.
The **radius** is the most important parameter that affect the accuracy and support (matching rate) level of my model. It is certain that as the radius level is raised, the support will be increased too. On the other hand, it is expected the algorithm accuracy fall down as the radius levels up. To find out these correlations, I decided to calculate the accuracy and support outputs as the radius is changed in a defined range.

However in order to execute the above experiments, I had to find out the possible radius values as my model’s metric is the cosine similarity. The cosine similarity between two vectors (or two documents on the Vector Space) is a measure that calculates the cosine of the angle between them: $\cos(\theta) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\|\|\vec{y}\|}$

Thus the radius (as absolute value) ranges from 0 to 1.

![Accuracy and Support per Radius](image)

**Figure 3.5: Accuracy and support levels per radius**

It can be seen from the above graph the accuracy and support levels as the radius takes values from 0 to 0.9. Also it is clear that there is a logarithmic correlation between the accuracy and radius. On the other hand, the support and the radius metrics have a negative no linear correlation. That experiment was applied in order to find out suitable pairs of support and accuracy values that meet my problems’ specifications.

<table>
<thead>
<tr>
<th>Radius</th>
<th>Accuracy</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.16</td>
<td>0.90</td>
<td>0.77</td>
</tr>
<tr>
<td>0.20</td>
<td>0.89</td>
<td>0.83</td>
</tr>
<tr>
<td>0.24</td>
<td>0.88</td>
<td>0.88</td>
</tr>
<tr>
<td>0.27</td>
<td>0.87</td>
<td>0.93</td>
</tr>
</tbody>
</table>

**Table 3.2: Accuracy and Support values per radius**
Unfortunately, at results emerged only one acceptable pair. So when the radius is 0.16, the accuracy and the support are set at 0.90 and 0.77 respectively. The next values gave me accuracy under the 0.90 bound. However we can compromise at accuracy standards and accept the intersection point for radius 0.24 where both of parameters are at 0.88. Also it is impressive that the support is shot up at 0.93 as the accuracy set at 0.87 with radius 0.27.

### 3.1.3 Evaluate product classification

With the following experiments I prove the dominance of my model over simpler solutions. I wanted to verify that the combination of two classifiers - predict firstly the product category and then the barcode label via Radius Nearest Neighbors - creates a more reliable system. Furthermore, I pointed out that my model not only has lower memory’s and cpu’s requirements but also it is more efficient in predictions in comparison to a simple RNN classifier that has been trained with whole training set.

The first experiment compares my first model with a simpler approach. The simpler approach (Model 0) does not use the first level of classification and try to find nearest neighbors without consider the product category. The algorithm that I applied was the Radius Nearest Neighbors with the same parameters as before.

![Figure 3.6: Classify with entirely vs per category training set](image)

It is obvious that Model 1 produces more reliable results. On the one hand, the support of simpler approach is a bit higher for radius from 0.16 to 0.9. However it was expected as we know that many barcode classes consist of 16 or less training products. On the other hand, the gap at accuracy’s lines is significant big and getting bigger as the radius is increased. Thus, the classifiers combination offers important better accuracy than the simpler approach. So we can conclude that the contribute of the first level classifier or the separation of
The training set to distinct product categories is vital in order to decrease the initial entropy. An other important asset of my model is the limited requirements in memory and the faster execution. As the Nearest Neighbors classifiers have to calculate and store the distances between testing and training instances, the biggest amount of neighbors at Model 0 demands 3.7 GB of physical memory and 13 seconds to predict classes for the test set. On the other hand, the Model 1 filters the test set and it splits the test instances to product categories and then it trains per category a RNN classifier and predicts the output labels. These processes don’t consume more than 140 MB and are executed (product categories and barcode prediction) on average in 4 seconds. This memory saving of 95% is vital as my model is applied to a millions of products.

The second experiment compared different text features extractions for the RNN classifier. So I wanted to see if other state of the art methods could successfully bring closer products from the same barcode class and estrange products from different classes. Until now, I had used the boolean n-gram model and I decided to give a change to the $tf$ and the $tf-idf$ methods as well.

As my aim is to transform a string $d$ (product name) to a vector of $n$ terms (unique words at product names corpus) the formulas for each term $tr(t_i, d)$, $0 \leq i < n$ are:

- **n-gram boolean**: $tr(t_i, d) = 1$ if $t_i$ occurs in $d$ and 0 otherwise.
- **tf (term frequency)**: $tr(t_i, d) = tf(t_i, d) = \text{frequency of } t_i \text{ in document } d$.
- **tf-idf (term frequency–inverse document frequency)**:

  $$tr(t_i, d) = tf(t_i, d) \times idf(t_i, D)$$

  where:

  - $idf(t_i, D) = \log \frac{N}{|d \in D : t_i \in d|}$
  - $N$: total number of documents in the corpus $N = |D|$;
  - $|d \in D : t_i \in d|$: number of documents where the term $t_i$ appears

Even the $tf$ and $tf-idf$ methods are used at 83% of text-based recommender systems[19] at my problems seems not to outperform the simpler boolean model. The above plot describes the correlation between accuracy and support for the 3 methods of feature extraction. It is clear that the boolean and $tf$ approaches are almost equally suitable for our problem. That was predictable, as in average a product name include 10 terms which frequency seldom gets over 1. On the other hand, the $tf-idf$ has up to 10% lower support for the same accuracy level, in comparison with the others two. This is justified because of the $idf$ term. It seems that the number of documents $N$ is quite bigger than $|d \in D : t_i \in d|$ and it equates the $tf-idf$ values.
The current approach has already enhanced the match rate (support) level of Convert group’s algorithm as it scores 78% for accuracy 90%. Also, it could improve further and touch the 93% as the accuracy is limited to 87%. There is no doubt that the RNN is suitable to overcome problems that occurred due to unbalanced barcoded product classes.

So the next step is to use more information that is included to my dataset such as the brand name, the product line or the quantity of products. The aim now is to improve the support level and keep the accuracy near to 90%.
3.2 Second model

The second model is an optimized version of the previous approach. To paraphrase the aim that I set before, I want to succeed better accuracy score to bigger radius level where the support is high. So I have to get closer input products of the same barcode class.

3.2.1 Detect first model classification issues

To figure out some causes of false positive and false negative, I decided to visualize some examples. So firstly I split the training set to two random subsets that the 70% of it is the new training set and the rest 30% the new test set.

```python
from sklearn.model_selection import train_test_split
X_train, X_test = train_test_split(df_train, test_size=0.30, random_state=None)
```

Then I created an output function that print the false predictions and let me find out some roots of the fault classifications. The system out has the below structure:

```
input_product name
correct barcoded name
predicted barcoded name
```

At this point, I should mention that the barcoded name is a product title/description that eRetail team has define for each barcode class (Barcoded_Products.name).

Let’s see some characteristic examples of product names(after the preprocessing) which were classified wrongly and set them to distinct categories.

- **Extra product as gift:**
  An extra product is offered with the main product as promotion package.

  - **input_product name:** apivita natural serum ενυδατωση με αλοη υαλουρονικο οξυ 15ml 1 l
  - **correct barcoded name:** apivita natural serum ενυδατωση με αλοη υαλουρονικο οξυ 15ml δωρο μασκα ενυδατωσης με αλοη 15ml
  - **predicted barcoded name:** natural serum ενυδατωσης 15ml.
- **input_product_name**: korres χρέμα 24ωρης ενυδατωσης λαμψή αγριο τριαντάφυλλο κανονικές 40ml.
  **correct barcoded name**: αγριο τριαντάφυλλο κρέμα 24ωρης ενυδατωσης λαμψή κανονικές ξηρες επιδερμιδές spf6 40 ml.
  **predicted barcoded name**: αγριο τριαντάφυλλο κρέμα 24ωρης ενυδατωσης λαμψή κανονικές μικτές επιδερμιδές 40ml. 

- **input_product_name**: vichy πακετο aqualia legere 50ml και δωρο pure thermal 15ml και aqualia thermal night spa 15ml.
  **correct barcoded name**: vichy aqualia thermal light 50ml gift purete thermal gel 15ml aqualia thermal night spa 15ml.
  **predicted barcoded name**: vichy promo set aqualia thermal κανονικές μικτές επιδερμιδές 50ml δωρο ιματικο νερο 50ml.

- **Extra quantity of product as gift:**
  Extra quantity of the same product is offered as promotion package with the initial.

  - **input_product_name**: rene furterer forticea shampooing stimulant 2 200ml.
    **correct barcoded name**: rene furterer forticea shampoo stimulant προσφορα 200ml 200ml.
    **predicted barcoded name**: forticea stimulating shampoo 200ml.
  - **input_product_name**: korres 1 1 σαμπουαν για κανονικά μαλλια με αλοη και δικταμο 200ml.
    **correct barcoded name**: σαμπουαν για κανονικά μαλλια με αλοη δικταμο 200ml 200ml.
    **predicted barcoded name**: korres σαμπουαν αλοη δικταμο 200ml.
  - **input_product_name**: 1 1 free tonic anti hair loss shampoo with cystine and minerals 2x250 ml.
    **correct barcoded name**: σαμπουαν κατα της τριχωπτοσης με κυστινη γνωστοχεία 250ml 250ml.
    **predicted barcoded name**: korres σαμπουαν κατα της τριχωπτοσης με κυστινη 250ml 200ml.

- **Differences in quantity:**
  The predicted and the correct product quantities are dissimilar.

  - **input_product_name**: boderm hairgen shampoo κατα της τριχωπτοσης 300 ml.
    **correct barcoded name**: hairgen shampoo 300ml.
    **predicted barcoded name**: hairgen shampoo 200ml.
  - **input_product_name**: bioderma sebium h2o 500ml διαλυμα καθαρισμου για δερμα λιπαρο.
    **correct barcoded name**: sebium h2o διαλυμα καθαρισμου 500ml.
    **predicted barcoded name**: sebium h2o διαλυμα καθαρισμου 250ml.
- **input product name**: Ducray Anaphase Shampoo 动态补充营养护发素 400ml.
  **correct barcoded name**: Anaphase shampooing cream stimulant 400ml.
  **predicted barcoded name**: Ducray Anaphase Shampoo 动态补充营养护发素 200ml.

- **Small differences in the descriptions**:
  There are disparities in few words such as an ingredient or the type of user.

  - **input product name**: Apivita Express Beauty Hydrating Mask with honey 2x8ml.
    **correct barcoded name**: Express μασκα ενυδατωσης θρεψης με μέλι 2x8ml.
    **predicted barcoded name**: Express beauty μασκα ενυδατωσης με αλοη 2x8ml.

  - **input product name**: Korres shampoo with KVP for women 1 1 δωρο 2x250ml.
    **correct barcoded name**: shampoo for the 雌性掉发 250ml 250ml.
    **predicted barcoded name**: shampoo for the antrικης τριχοπτωσης με κυστινη 250ml 250ml.

  - **input product name**: Nuxe Ftaiche Creme Suractivee for dry skin 50ml.
    **correct barcoded name**: creme fraiche concentree de beauty suractivee 24h moisturizing creme dry very dry skin 50ml.
    **predicted barcoded name**: Creme ftaiche de beaute suractivee 24h energising moisture emulsion normal skin 50ml.

- **Consequence errors in training set**:
  There are some classification issues which result from wrong or duplicated labels at few training products.

  - **input product name**: Korres 24ωρης ενυδατωσης αγριο τριανταφυλλο για λιπαρες μικτες επιδερμιδες 60ml.
    **correct barcoded name**: Korres αριο τριανταφυλλο κρεμα 24ωρης ενυδατωσης 40ml.
    **predicted barcoded name**: 24ωρης ενυδατωσης αγριο τριανταφυλλο για λιπαρες μικτες επιδερμιδες 60ml.
The examples above summarize some key causes of my model’s classification failure. Some of them are difficult to overcome but some others could be manipulated. For example there are various different ways with which the product offers have been inserted at e-shop’s databases as single product: ‘2x50ml’, ‘2χ 50ml’, ‘1 1 δωρο’ or ‘50ml 50ml’. Some other simple but serious mistakes at product names are the spaces between important information, such as at the quantity phrases: 150ml, 150 ml, 2x 50ml, 2 x 50ml. We should not forget that the product names have been defined from e-shop’s employees. So their spelling mistakes or unconformity at entries typing could reduce the level of similarity that products of the same barcode class have. So for example the entities ‘apivita express beauty hydrating mask with honey 1 1 8ml’ will be closer with the product ‘apivita express beauty hydrating mask with honey 8ml’ instead of the ‘apivita express beauty hydrating mask with honey 2x8ml’

Despite the wrong classification decision, my model produces some misclassified products. As I said before, my aim is to reduce the amount of mislabeling inputs in order to increase the matching rate. An ideal system ought to classify any input product that its barcode class has been established from previous product input and set any other to the unlabeled set. So I decided to clarify too some e-shops’ wrong policies at products names typing. That I discovered is that the brand name, which it is crucial for product clarification, is missing or it has been typed wrongly at many input products. For instance:

- **la rocheposay** effaclar gel.
  - la roche posay effaclar gel καθαρισμού.
  - larocheaposay gel καθαρισμού.

- **la roche posay** effaclar gel 150ml.
  - lrp effaclar gel moussant 150ml.
  - effaclar gel moussant 150ml.

- **korres** showergel coconut sand 250ml.
  - korre showergel coconut 250ml.
Then I realized that I have to develop solutions to hold these phenomenons and bring closer products that belong to the same barcode class.

### 3.2.2 Normalize quantity dissimilarities

At this chapter I present my following practices that helped me to overcome some of the mentioned above problems. The regular expressions and the string functions helped me to bridge the product’s differences at quantities, offers and promotions expressions.

Firstly, I focused on quantities expressions. As I said before a space between the number and the type of quantity could afflict the products’ similarity. Also some e-shop use the Greek ‘χ’ instead the Latin ‘x’ at the quantities’ offers. So I created a scalable function that correct these dissimilarities.

```python
# remove the spaces between number and quantity type:
#’200 ml ’ —> ’200ml’
clear_data2 = re.sub(r'(?<=\d)\s+(?:(?:ml|gr)\b)', '', clear_data)

# remove the spaces between number and multiplication at packages:
# ’2 x50ml’ —> ’2x50ml’, ’2 x 50ml’ —> 2x 50ml, ’2 χ50ml’ —> ’2x50ml’
clear_data3 = re.sub(r’(?<=\d)\s+(?!x)\b', '', clear_data2)

# remove the spaces between multiplication symbol and quantity phrase at packages:
#’2x 50ml’ —> ’2x50ml’, ’2χ 50ml’ —> ’2x50ml’
clear_data4 = re.sub(r’(?<=\d)\s+(?:(?:x|\chi)\b)', '', clear_data3)

# replace the greek ‘χ’ with latin at offers
#’2χ50ml’ —> ’2x50ml’
clear_data4 = re.sub(r’(?<=\d)x\b', 'x', clear_data4)
```

Secondly, the offers and the packaged products ought to be expressed through homogeneous format. So I limited the faults that belong to categories “Extra quantity of product as gift” and “Extra product as gift”. The corrections were applied to phrases like ‘1 1 δωρο 50gr’, ‘2 τεμ 100gr’ in order to rephrase them to ‘2x50gr PROMO’ and ‘2x100gr’ respectively.

At this step, I detected product’s names which quantity was described as ‘{number} τεμ’ and they were replaced with ‘{number}X{quantity}’.
At the third step, I found product packages that offered an extra product as gift and transformed them to ‘2x{quantity} PROMO’.

To sum up the corrections that fulfilled at the above steps, bridged over some existing differences in products quantities definitions. As the quantity is an important feature for our models learning, even small modifications could improve our accuracy. So I eliminated the spaces at quantities phrases (“250gr”, “2x10ml”), transformed the product packages descriptions to the general quantity format (“4x50gr”) with Latin ‘x’ and the offers to PROMO patterns (“2x250ml PROMO”). All the modifications at product names were chose carefully and evaluated to training set to ascertain their values.
It is clear that the import of quantity normalize method improved significant the correlation between accuracy and support. So if we keep stable the accuracy at 0.90 the support increased from 0.74 to 0.80. The supremacy of new model is all over the range of the radius values. Additionally at the 3.2.4 section, the improvement is confirmed with experiments on test set.

3.2.3 Revise brand names misquotations

Despite the corrections in the quantity format, I tried to control the brand name misquotations at product descriptions. The brand name forms the second most important learning feature for my classifiers after the quantity. As I analyzed before, some product names include spelling mistaken brand names or the brand names are totally missing at them. So I used the training set’s brand names and product lines to correct the products’ descriptions.

Generally, I tried to create a function that could specify a brand name as using the product name information. At industry a brand name could coincide with a company name or could be a totally distinct name. For example the company “Unilever” has the brands “DOVE”, “LUX”, “REXONA” or “SUNSILK”. However the company “Loreal” owns brands as “LOREAL” and “LA ROCHE POSAY”. Additionally each brand may have multiple product lines. So in my data set, the product lines “CREAM OIL”, “FINE SILK”, “BEAUTY BATH”, “PURELY PAMPERING” and “GENTLE EXFOLIATING” belong to brand “DOVE”.

My first attempt is to figure out the percentage of product names that are include the brand name correctly. Thence, as I collected all the unique brands
from the table Brands, I cleared them through my preprocessing function (the same process as product names before) and then I created a brands set. My next step was to join the brand set with each training product name and find common values that could be possibly brand names. In order to apply the join, I used a CountVectorizer and split the product name from 1 to 5 n-grams as the longest brand name has 5 words. So I joined the brand name set to product name 1-5 gram set and kept the matched values. However I set a convention and if the result of a join had more than one brand name, I kept the one with more words. It may seems a wrong decision but as the average product name consists of 10 terms, it is impossible to contain a phrase with 3 words and that information not be corresponded to brand or product line name.

On the other hand, if the join procedure returned empty set, the brand for the corresponding product inputs remained empty. Moreover, as for each bar-coded product has already determined the correct brand name from Convert Group team, the evaluation is possible.

\textbf{Brand set} : [L'Oreal, La Roche Posay ,A.Vogel ,DOVE,...]  
\textbf{Clear Brand set} : [loreal, la roche posay ,avogel ,dove,...]

The above experiment result is that we got an no-empty prediction for the 84% of training records. So we can say that the prediction support is 84%. In addition, I calculated the average precision, recall and f1 score per brand name.

<table>
<thead>
<tr>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 3.3: Evaluation metrics for first brand prediction

The too perfectly results means that in majority of training products, the brand name is included correctly in their names and also that there are not overlaps with the rest phrase of product descriptions. So we are pretty sure that this method is reliable.

Suppose that we want to increase the support level, we have to find a second brand prediction method for the rest 16% of products. At the next step, I tried something little different. For the brand names that consist of more than one word (e.g. “la roche posay”) I decided to remove the spaces and searched for them to a product name. Similarly the spaces were removed from the product names. This thought was born as I noticed that some e-shops as make their entries, skip some spaces. For example:
So I applied this method similarly with the first procedure and thus the support value increased to 86%, while the accuracy metrics remained highly.

<table>
<thead>
<tr>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>0.99</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Table 3.4: Evaluation metrics after the second brand prediction method

The third and last method was applied to unsupported subset (14%) of the two previous processes. At this function, I used the product line information in order to predict the brand name, as each product line name could belong only to one brand. Thence this property could be important for products that the brand name has been included to their description with spelling mistakes, at a abbreviation version, with greeklish characters or when it is totally missing. Some useful examples are:

- “lrp effaclar gel moussant...”: la roche posay
- “korre black pine - μαύρη πεύκη...”: korres
- “κορρες almond blossom κρεμα ενυδατωσης...”: korres
- “olivia σαπουνι για ξηρα...’: papoutsanis
- “exomega baume emollient για ξηρες: a derma

To proceed the join between the product line set and each product name, I followed the same process as the previous join between the brand names and the product description. It is very important that after the usage of product line information, the support shot up to 94%. However not only was an increase of 8% at support value but also the evaluation metrics did not fall down.

<table>
<thead>
<tr>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.99</td>
<td>0.98</td>
<td>0.98</td>
</tr>
</tbody>
</table>

Table 3.5: Evaluation metrics after the third brand prediction method

To confirm the ability of my methods to specify the brand name I tested them on test set. The metrics below came after the execution of each method which had as input the empty prediction set of the previously.
<table>
<thead>
<tr>
<th>Method</th>
<th>Support</th>
<th>F1</th>
<th>Overall Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>0.84</td>
<td>0.99</td>
<td>0.84</td>
</tr>
<tr>
<td>+ 2nd</td>
<td>0.86</td>
<td>0.99</td>
<td>0.86</td>
</tr>
<tr>
<td>+ 3rd</td>
<td>0.94</td>
<td>0.98</td>
<td>0.92</td>
</tr>
</tbody>
</table>

Table 3.6: Evaluation metrics for brand name prediction on test set

At the above table we can see the support and the f1 levels per method. Moreover, the overall accuracy refers to correct predictions of entirely test set. That means that the empty predictions of each method were identified as wrong labeling.

There is no doubt that my scalable process can find out the brand name via the product name with highly accurateness. Furthermore, the final support level is important higher in comparison to my first classification model. So the brand name could contribute to improve the overall performance as it is used with suitable way. The simplest solution is to add each predicted brand to the corresponding product name. However this method may not be the ideal. So I decided to replace the wrong inputs of brand name with the predicted brand name. To be more accurately, I split the predicted brand to terms, I removed its term of the product name and then I placed the predicted name at the beginning of product name. Some examples of the application:

- “laroche posay effaclar εξισορροπησης για εντονη λιπαροτητα” → la roche posay effaclar εξισορροπησης για εντονη λιπαροτητα”
- “avogel bioforce cream 35γρ” → a vogel bioforce cream 35γρ
- “lissea shampooing lissage” → rene furterer shampooing lissage
- “rene f acanthe baume velours perfecteur boucles 150ml” → rene furterer f acanthe baume velours perfecteur boucles 150ml

The next step was to combine the above processes that revise the quantity and the brand name expressions at product names and then execute again the two level classifier in order to calculate the overall improvement.
3.2.4 Evaluate product classification

After the new preprocessing state, I repeated the same functions as at my first model. For each input product, I cleared its product name, normalized its quantity phrase and then corrected or added its brand name. Then I predicted the product category, trained the corresponding RNN classifier and finally predicted the barcode class.

![Diagram of process]

Figure 3.9: My second model’s processes

Now let’s compare the support levels of two models as we keep the accuracy fixed. The first comparison took place for accuracy at 90% which was the initial aim.

<table>
<thead>
<tr>
<th></th>
<th>Model 1</th>
<th>Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.90</td>
<td>0.90</td>
</tr>
<tr>
<td>Support</td>
<td>0.77</td>
<td>0.86</td>
</tr>
<tr>
<td>Radius</td>
<td>0.16</td>
<td>0.20</td>
</tr>
</tbody>
</table>

Table 3.7: Compare the two models’ performances for 90% accuracy

It is clear that the support (or matching rate) was significantly improved. Additionally, we can observe that the radius in which the RNN succeed 90% was increased by 4 units. However if we compromise and accept a lower accuracy level, the second model could succeed even better performance. So for 89% and 88% accuracy I got the followings:
<table>
<thead>
<tr>
<th></th>
<th>Model 1</th>
<th>Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accuracy</strong></td>
<td>0.89</td>
<td>0.89</td>
</tr>
<tr>
<td><strong>Support</strong></td>
<td>0.83</td>
<td>0.91</td>
</tr>
<tr>
<td><strong>Radius</strong></td>
<td>0.20</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Table 3.8: Compare the two models’ performances for 89% accuracy

<table>
<thead>
<tr>
<th></th>
<th>Model 1</th>
<th>Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accuracy</strong></td>
<td>0.88</td>
<td>0.88</td>
</tr>
<tr>
<td><strong>Support</strong></td>
<td>0.88</td>
<td>0.96</td>
</tr>
<tr>
<td><strong>Radius</strong></td>
<td>0.24</td>
<td>0.33</td>
</tr>
</tbody>
</table>

Table 3.9: Compare the two models’ performances for 88% accuracy

There is no doubt that the modifications that utilized at second model are beneficial for the prediction accuracy. In addition, the support score of 96% limited the unsupported set at a level that permit the real time barcoded prediction. Despite the above results I plotted the accuracy and support values per radius from 0.1 to 0.90.

Figure 3.10: Model 1 vs Model 2 on accuracy and support per radius

The dominance of Model 2 over the Model 1 is at the entirely range of radius. It is clearly that with Model 2, the accuracy was increased at radius levels where the support is important higher. So I succeed to improve the support(or matching rate) metric up to 9% with the second model. To put it differently, similar products got closer and dissimilar were distanced. About the memory requirements I used the psutil as I described at the comparison of Model 0 with Model 1. The function gave me that the top memory usage remained stable at 150 MB. However, the duration of an execution increased from 4 second to 40
seconds as the corrections of quantities and brand names in training and test set lasted 37 seconds. It is expected as the joins of brand names and product lines to product names are demanding to calculations.
Chapter 4

Conclusions

Firstly, my project aim was clearly specify. It was a supervised problem in which I had to matched some input products with others that have already inserted to a database. The mainly source of the training product collection was 59 Pharmaceutical e-shops. Moreover, the known products had matched with some defined and distinct barcoded products which constitute the final classes. In addition the Convert Group’s team had collected and determined some useful attributes for the database’s products and they had distributed them to multiple tables. So I chose the suitable attributes in order to develop my solution.

Secondly, it was easy for me to understand that the key information that I had for the products was the “product name”. Essentially the product name is the description that each e-shop has set for its database products. However the products’ description length and the vocabulary are depends on the e-shop as well as the language that the product name have written. So in average each product name consisted of 10-15 words that came from 2-3 different languages.

Thirdly the variety pattern in product names entities wasn’t the only problem that I had to face. The major obstacle was the unbalanced barcode classes that have defined in my training set. Specifically the training products were 38,400 and they had distributed to 3841 barcode classes. Nevertheless the distribution of products to the classes was nonuniform and it had as result to exist classes with 900 instances, others with only 2 instances and in average each barcode class consisted of 16 products. So it was forbidden the application of linear or probabilistic classifiers and thus I tried the Nearest Neighbors algorithms. Actually I used the Radius Neighbors Classifier that permits to find the neighbors of an input product in a defined radius. Additionally, it gave me the ability to measure out my matching rate as it set each input product without neighbors inside the radius, into an unlabeled set. However in order to
overcome the problem of high memory demanding and to increase the accuracy of my predictions I initialized a product categories classifier before the RNN level. So I set up a Logistic Regression classifier that predicted in which of the 59 categories, an input product belongs. The next step was to initialize a Radius Nearest Neighbors per category and predict the final barcode class. The 2 level classifier reduced the max memory usage 95% and increased the overall accuracy and the support rate.

At the final step I created functions that spotted the quantities at product input, figured out the offer packages and then transformed the corresponding phrases to a defined format. Moreover I used the product line and the brand information to correct or place the brand name into a product name. These practices helped me to shoot up the matching rate to 86% as the accuracy was preserved to 90%. This score is 11% higher of the Convert Groups algorithm’s matching rate. However with my implementation, we can increase the supported population up to 96% as the accuracy is reduced to 88%.

![Figure 4.1: Model 1 vs Model 2 on accuracy and support per radius](image)

The mainly conclusions is that the product matching is a challenging problem with many mannerisms. It is not a common classification problem with well defined categories as the classes could be a few thousand and extremely unbalanced. However there is no doubt that my dataset was carefully collected without inconsistencies at products’ barcode classes. So the RNN classifier was ideal for product matching and it was proven that the two level classifier created a very efficient solution. Nevertheless the normalization of product name expressions was necessary as the product entities come from different sources.
Chapter 5

Future Work

My approach is a solution in the field of product matching that could be improved or combined with different models. The last years as the online market is growing rapidly, a few implementations[20][21] have been developed to help the user to find similar product in a specific e-shop or they try to collect all the possible sites that offer a product that the user have searched for. However some of their ideas would be amble to apply with my model to improve the prediction accuracy.

First of all, I could collect more features. As I mentioned before, the database products were gathered from e-shops with the Google analytics. If it was possible to be collected with the product names and the webpage url, many informations could be crawled from the eshops web site. Generally, a product webpage consist of an image, a price, an analytic description or a specification report. Thus not only our text content will be enriched but also new feature types could extracted as prices or images. On the other hand, this task requires the construction of a crawler per e-shop as any website has its own structure. Furthermore it is very important the timely crawling of product informations as many times the products are removed from websites in short time.

Secondly as we have collected more data, a multi classification system could be developed. For example the images could train a deep learning system to classify our products. This classifier will return probabilities for an input product to belong to each barcode class. As we modified the RNN classifier to return the probabilities of each possible label or each smallest distance for an input product from all neighboring classes then the combination of the outputs could give us a more reliable decision. Moreover a parallel big data system like Spark may could train a linear classifier even though the classes are some thousands and a local system with limited computation power can not be used. With this ways we could construct and combine many independently decision systems.
Finally, the hierarchical classification is an other way to approach the solution of our problem. A pharmacy product has many specifications like brand, product line, quantity, price, type of user or form. If it was possible to extract all of these features from the product names or descriptions, then a system like decision tree could be used to find the similar product or to reduce the search space. This thought is quite similar with the 2 level classifier that I created above. However in order to create a reliable decision system, on the one hand the feature extraction/prediction should be without faults and on the other hand the training set ought to be consist of many instances for all the possible features. Otherwise the final decision system will produce many false positive labels.
Chapter 6

Technical Appendix

This chapter contains information about the system that executed the experiments and the code files that I created to develop my solutions. Also there are descriptions for the functions per code file and the ways to pass the arguments for the models executions.

First of all my system was a laptop with intel i7-4700HQ CPU @ 2.40GHz (4 cores, 8 threads), 8GB DDR3 1600MHZ physical memory with SSD hard drive and Ubuntu 16.04 operation system. The program language with which I developed my models is Python 3.6 and the libraries that I used were pandas, numpy, scikit-learn and matplotlib.

I created 10 Python files and 1 Jupyter notebook to manage my data, apply my algorithms and visualize my results. The “clean_library.py” and the “brand_detect.py” contain the product name preprocessing functions, the “model0_experiments.py” produces the outputs of simplest RNN Model 0, the “model1_experiments.py” and “model1_main.py” implement the Model 1 and the “model2_experiments.py” and “model2_main.py” consist of the corresponding functions for Model 2. Moreover, the “categories_classifier.py” was developed as the first level classifier that predicts the product category, the “nearest_neighbors_library.py” contains all needed functions for RNN classifier and the “plot_lib_f.py” and the Jupyter file “Visualization_Scripts.ipynb” include the visualization scripts. The functions are following per file:
clean_library.py:

- def f_clear_data_1(string):
  This preprocessing function removes html tags, replaces punctuations (except for '%') with spaces, removes white spaces and turns the characters to lower cases. It is used by model 1 and model 2.

- def f_clear_data_2(string):
  This preprocessing function use the f_clear_data_1 to clear the product names and apply the normalization at quantity and offer phrases. It is used by model 2.

brand_ditect.py:

- def correct_name(array):
  This function gets as input an array with product names and returns as output the product names with corrections at brand names. The process has been split at three stages that try to detect the brand name using the product name and the sets of brand names and product lines in the database. Each stage try to predict the brand names that the previous stage couldn’t predict. Finally, as my method figures out the brand names, they are added to product names or they are replace possible wrong brand names’ formulations in the product names. It is used by model 2.

categories_classifier.py:

- def ngramms_extraction(CountVectorizer,array):
  This function gets as inputs a CountVectorizer object and an array of strings. For each input vector returns a term-document transformation. If the CountVectorizer hasn’t initialize(its vocabulary is empty) then it keeps a vocabulary dictionary of all tokens in the raw documents.

- def classifier_parameters_estimating(classifier, dictionary,array,vector):
  This function gets as input a dictionary with parameters for tuning, an array with the instances and an array with correct labels. The dictionary contains as keys the parameters names and as values a list with possible values. It returns the best estimators after a grid search.

- def print_classification_metrics(classifier,array,vector):
  This function gets as inputs a classifier, an array with the instances and an array with correct labels. It executes a 10 Kfold method to evaluate the classifier and prints precision, recall, and their harmony average, the f1 per class and in average.
• def create_categ_model(array,str,boolean,boolean): This function gets as inputs an the training array, classifier type, a flag for the evaluation and a flag for the parameter tuning. It saves in local folder the countvectorizer and the trained model for future category prediction.

If the evaluation flag is True, then it calls the print_classification_metrics for 10 Kfold evaluation. Respectively, if the param_tunning flag is True, then the function classifier_parameters_estimating is executed to estimate the parameters with grid search.

• def pred_categ(array):
  This function uses as input a testing set of strings, a classifier model(loaded from file) and CountVectorizer(loaded from file) to predict the final categories.

• main(str,boolean,boolean):
  The main function loads the training, the testing set and the barcoded_products. It joins the barcoded_products on training and testing set to find the categories_id( the categories labels). Then it calls the create_categ_model function to create and save the category model and the CountVectorizer. Finally it predicts the categories of testing set and calculates the prediction. As input takes 3 arguments. The first refers to classifier type {'LR','DsTr','KNN'} and the rest 2 are boolean {'True,False'} for evaluation and parameter tuning applications.

nearest_neighbors_library.py:

• def tfidf_extraction_initialize(array,int,int,flag,list):
  This function gets as input an array of text records, an idf reweighting flag, a ngram range upper bound, a ngram range lower bound and a list of stop_words. It initializes a CountVectorizer and a TfidfTransformer to transform the text array. Moreover the user can choose and get a TF transform as set the flag use_idf fl to False. The function returns the transformed array as well as the CountVectorizer and TfidfTransformed objects.

• def tfidf_extraction_fit(array,TfidfTransformer):
  This function gets as input an array of text records, a CountVectorizer object and a TfidfTransformer object. It transforms the input texts to TF-IDF format and returns the output object.

• def ngrams_extraction_initialize(array,int,int,list):
  This function gets as input an array of text records, a ngram range upper bound, a ngram range lower bound and a list of stop_words. It initializes a CountVectorizer to transform the text array to n gram combination. The function returns the transformed array as well as the CountVectorizer object.
• def ngrams_extraction_fit(array, CountVectorizer):
  This function gets as input an array of text records and a
  CountVectorizer object. It transforms the input texts to n gram format
  and returns the output object.

• def neighrest_neighbors_eval(array, array, array, vector, int, float):
  This function predicts for a test set the product barcodes using the RNN
  algorithm. Furthermore it keeps statistics for the accuracy per product
  category and a list with all the false or no predicted records. The items
  in the list has the following format: ('Product name', predicted
  label, 'true label')

• def r_neighrest_neighbors(array, array, list, float):
  This function gets as input the X train and X test to ngram format as
  well as the Y labels and the support radius for RNN algorithm. As it
  executes the classifier, it predicts the labels for the test instances and
  the quantity of unsupported records. The RNN uses cosine similarity,
  brute algorithm, parallel processing in predictions and set 'NONE' label
  at any testing instance that it doesn’t have any neighbor into the define
  radius.

module1_main.py:
• main(float, str):
  This function gets as arguments the radius \{[0.1, 1.0]\} for the RNN
  algorithm and the feature extraction method \{tf, tfidf, boolean\}. As it
  loads the training and test set clears the product names and extracts the
  selected features. Then it predicts the categories for the test instances
  and trains a RNN for each product category. Finally it predicts the
  barcode labels for the testing products.

module1_experiments.py:
• main(str):
  This function gets as argument the feature extraction
  method \{tf, tfidf, boolean\}. As it loads the training and test sets, clears
  the product names and extracts the selected features. Then it predicts
  the categories for the test instances and trains a RNN classifier for each
  product category. The next step is to predict the labels of testing
  instances per category for all radius values of range \[0.1, 0.9\]. At each
  execution it calculates the micro and macro accuracy and the support
  scores per radius. The scores are saved to folder 'outputs/' in list format.
model0_experiments.py:

- main():
  This function doesn’t get any argument. As it loads the training and test set clears the product names and extracts the selected features. The next step is to train the RNN classifier for all radius values of range [0.1,0.9] and predithe the barcode labels for the testing instances. At each step it calculates the micro and macro accuracy and the support scores per radius. The scores are saved to folder ‘outputs/’ in list format.

model2_main.py:

- main(float,str):
  This function gets as arguments the radius∈[0.1,1.0] for the RNN algorithm and the feature extraction method∈{tf,tfidf,boolean}. As it loads the training and test set, clears the product names, corrects the product quantities and the brands in product names and it extracts the selected features. Then it predicts the categories for the test instances and trains a RNN for each product category. Finally it predicts the barcode labels for the testing products.

model2_experiments.py:

- main(str):
  This function gets as argument the feature extraction method {tf,tfidf,boolean}. As it loads the training and test set clears the product names, corrects the product quantities and the brands in product names and then it extracts the selected features. Then it predicts the categories for the test instances and trains a RNN classifier for each product category. The next step is to predict the labels of testing instances per category for all radius values of range [0.1,0.9]. At each execution it calculates the micro and macro accuracy and the support scores per radius. The scores are saved to folder ‘outputs/’ in list format.

Finally I have to mention that in order to execute the Python codes, the ‘datasets’ folder is needed at the parent folder of folder ‘code/’. Furthermore, the execution of ‘category_classifier’ is necessary in order the file ‘categ_model.sav’ be created for the product category predictions. All the output files of experiments are stored into the ‘outputs/’ folder.
Bibliography


[6] Python: Package Index - langdetect - 1.0.7
   https://pypi.python.org/pypi/langdetect

   https://en.oxforddictionaries.com/definition/brand


