Counterfeit Product Detection with Deep Learning

Author: Konstantinos Gavriilidis
Supervisor: Dr. Yannis Konstas

A thesis submitted in fulfillment of the requirements for the degree of MSc. Artificial Intelligence with Speech and Multimodal Interaction
in the
School of Mathematical and Computer Sciences

August 2018
Declaration of Authorship

I, Konstantinos GAVRIILIDIS, declare that this thesis titled, 'Counterfeit Product Detection with Deep Learning' and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Konstantinos Gavriilidis

Date: 16/8/2018
“Look up at the stars and not down at your feet. Try to make sense of what you see, and wonder about what makes the universe exist. Be curious.’

Stephen Hawking
Abstract

In this project, a deep learning approach that detects relevant products for a specific brand and spots counterfeit goods was introduced and developed. With the use of the AllenNLP library and supervised learning, the algorithm processes textual content and performs a categorization that produces a different output in each case. As a result, a trained model that receives six different product features (e-commerce platform, search terms, description, price, seller, image tags) and creates predictions for the relevancy or the authenticity of a product was built.

The outcome of this work, was a high performance model that achieved 93% accuracy for relevance detection and 83% accuracy for counterfeit detection as a pipeline.
Acknowledgements

I would first like to thank my supervisor professor Yannis Konstas of the School of Mathematics and Computer Science at Heriot-Watt University. Prof. Konstas was always there whenever I needed advice for my research or the practical part of my dissertation.

I would also like to thank my colleagues from SnapDragon Monitoring who guided me during the three months of my placement for my dissertation. Without their passionate participation and input, the result of my work would not be the same.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my year of study. This accomplishment would not have been possible without them.

Thank you.
Contents

Declaration of Authorship                         i

Abstract                                           iii

Acknowledgements                                   iv

Contents                                           v

List of Figures                                    viii

List of Tables                                     ix

Abbreviations                                     x

1 Introduction                                      1
  1.1 Problem statement                            1
  1.2 Introduction to the project                  2
  1.3 Aims and objectives                          3

2 Background Knowledge                             4
  2.1 Machine Learning                             4
    2.1.1 Supervised learning                      4
    2.1.2 Dataset types                            5
    2.1.3 Machine Learning metrics                 6
    2.1.4 Machine Learning issues                  7
  2.2 Text Classification                          7
    2.2.1 Global Vectors for Word Representation    9
    2.2.2 Bag of words                             10
  2.3 Artificial Neural Networks                   11
    2.3.1 Activation functions                     11
    2.3.2 Dropout                                  12
    2.3.3 Feed-forward neural networks             12
    2.3.4 Recurrent neural networks                13
    2.3.5 Long short-term memory                   14
List of Figures

1.1 The four product infringements with examples for each case (blog.thetrademarkhub.com) 2
1.2 Counterfeit product detection with the use of a trained deep learning model 3

2.1 Depiction of the training phase and the prediction of class values for unlabeled data by the trained model (nltk.org) 5
2.2 Dataset proportions (rpubs.com) 6
2.3 Classification statuses that indicate the correctness of a classification (mathworks.com) 6
2.4 The different steps of text classification, that start from the reading of a document and lead to use of the features by the learning algorithm 9
2.5 Linear substructures (Pennington et al. [2014]) 10
2.6 Neuron structure (ujjwalkarn.me) 11
2.7 Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped. Srivastava et al. [2014] 12
2.8 Feed-forward neural network (ujjwalkarn.me) 13
2.9 Comparison of a recurrent neural network and a feed-forward neural network (towardsdatascience.com) 14

3.1 Adversarial advertisement detection with SVM and cascade model utilization (Sculley et al. [2011]) 17
3.2 Semi-automatic detection of counterfeit products with the use of clustering and reviewer intervention (Wartner et al. [2015]) 17
3.3 Character frequency for two products (Benezeth et al. [2015]) 18

6.1 Representation of the events and artifacts in the scrum framework (scrum.org) 22
6.2 Unit test that oversees function outputs for the relevance predictor 24
6.3 Integration test on the deep learning model 24

7.1 The Swoop architecture that utilizes a scraper for data extraction, stores the data on a S3 bucket and uses the AI API for relevance and counterfeit detection. (mesomorphic.co.uk) 26
7.2 The AI API model with five encoders and a feed-forward neural network. 27
7.3 Tag extraction process for Amazon Rekognition (aws.amazon.com) 28
List of Tables

4.1 Requirements analysis and importance evaluation .................................. 19
7.1 Link reader methods .............................................................................. 29
7.2 Classifier methods ................................................................................ 30
7.3 Predictor methods .................................................................................. 31
7.4 Dictionary generator methods ................................................................. 31
7.5 Configuration parameters ....................................................................... 32
7.6 CLI commands ....................................................................................... 33
8.1 Single feature metrics for relevance detection ......................................... 35
8.2 Metrics for two features for relevance detection ..................................... 35
8.3 Metrics for three features for relevance detection .................................... 35
8.4 Best metrics for different feature set sizes for relevance detection ........ 36
8.5 Single feature metrics for counterfeit detection ........................................ 36
8.6 Metrics for two features for counterfeit detection .................................... 36
8.7 Metrics for three features for counterfeit detection .................................. 37
8.8 Best metrics for different feature set sizes for counterfeit detection ...... 37
8.9 Generated metrics for both categorizations in the pipeline test ............... 37
8.10 Performance for relevance classification with multiple brands ............. 38
8.11 Performance for relevance classification with multiple brands ............. 38
# Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>BOW</td>
<td>Bag Of Words</td>
</tr>
<tr>
<td>LSI</td>
<td>Latent Semantic Indexing</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi Layer Perceptron</td>
</tr>
<tr>
<td>VCS</td>
<td>Version Control System</td>
</tr>
<tr>
<td>CLI</td>
<td>Command Line Interface</td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified Linear Unit</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short Term Memory</td>
</tr>
</tbody>
</table>
Dedicated to my family, friends and colleagues
Chapter 1

Introduction

1.1 Problem statement

Assets like intellectual property represent the investments in research and development and the pursuit of quality by manufacturers (King [2002]). The practice that ignores such assets and involves the unauthorized manufacturing of products that imitate certain characteristics is called product counterfeiting (Vithlani [1998]). Product counterfeiting is a serious threat to industries and brand owners and is no longer confined to branded luxury goods because of the emergence of e-commerce platforms (Staake et al. [2009]). The disadvantages of this phenomenon extend not only to manufacturers that are likely to face a loss of revenue (Montoro-Pons and Cuadrado-Garcia [2006]), but also to consumers that experience the loss of quality and exclusiveness of several products (Wilke and Zaichkowsky [1999]). The most reasonable solution to this problem is a set of measures in the form of enforcement penalties for counterfeiting, product recalls and liability claims for health and safety issues (Feinberg and Rousslang [1990] : Liebowitz [2005]). The most challenging part for brand owners is the detection of counterfeit goods based on four types of infringement: copyright, trademark, design rights and patent infringement (Baldini et al. [2015]), without which the prosecution of counterfeiters is impossible. Figure 1.1 highlights the traits that could be imitated in each infringement. A case of copyright infringement might be the use of a coca cola soundtrack without referencing its creator, while the use of Sainsbury’s slogan could be considered a trademark infringement. A possible design infringement could be the creation of a jacket which is identical to the one that the Columbia brand makes.
1.2 Introduction to the project

This project involves the creation of a tool that automates the detection of relevant products for a certain brand and the spotting of counterfeit products. The company was able to provide two different types of datasets. The first has the same number of relevant and irrelevant links and the second has at least 300 examples for each type of infringement for the same brand. For counterfeit spotting, the company deals with only three of the infringements aforementioned in section 1.1 (copyright, trademark, design rights) and thus, the model uses those categories to describe counterfeit products. For this project, supervised learning is used to train a model with labeled data that have been previously reviewed by other employees and produces good results in a fast manner.

Furthermore, the approach that was selected uses a deep learning model which receives six different features as an input (platform, search terms, description, price, seller, image tags). Once the algorithm extracts the features from the dataset, it encodes them either with a bag of words or a LSTM to make the processing of the data easier and passes the encoded features to a feed-forward neural network for the actual classification. The same architecture was selected for both models (relevance detection, counterfeit detection) and for each scenario, a series of experiments was made in order to determine the best set of inputs. The final test that was implemented, uses both classifiers as a pipeline to first select relevant links from a dataset and then check those for counterfeiting. In Figure 1.2, the basic task of the implementation is demonstrated, where the trained model takes the characteristics of a product as input and gives an output that indicates if a product is genuine.
1.3 Aims and objectives

The aim of this project is the achievement of a significant performance with the available technologies and datasets. The main milestones of this thesis are the following:

1. Extraction of the necessary datasets for the Trtl brand (training, validation and testing set) for each model.

2. Creation of a trained model that predicts the relevance of links by using textual input (e.g. description, price, seller).

3. Creation of a trained model that predicts if a product is a counterfeit and the type of infringement (copyright, trademark, design rights).

4. Use of the two classifiers as a pipeline for the complete automation of counterfeit product spotting.

5. Training and testing of models that deal with each detection for multiple brands.

6. Collection of the necessary results from the experiments.

7. Final evaluation of the results.
Chapter 2

Background Knowledge

2.1 Machine Learning

According to Talwar and Kumar [2013], machine learning is the concept which teaches machines to detect different patterns and to adapt to new circumstances. The success of learning in this project is determined by the number of correct categorizations divided by total number of data points in a dataset. Based on this criterion, a series of adjustments has been made to achieve the best possible performance.

An important factor for machine learning is the amount of data that the model can be trained with. As Domingos [2012] mentions in his paper, a large quantity of data can beat a smart approach and produce better results. Likewise, this implementation uses rich datasets with a plethora of different examples for each detection and that’s one of the main factors that made the generated models so accurate.

Finally, machine learning can be achieved with the use of many different models (e.g. decision trees, naive bayes), but this project is mostly related to neural networks.

2.1.1 Supervised learning

In his research, Kotsiantis et al. [2007] describes supervised learning as the search for algorithms that reason from externally supplied instances to produce general hypotheses, and then make predictions about future instances. Identically for this project, by providing a set of data that has a descriptive value for each instance, it was possible to train a model that finds patterns between different features and relates them to label. Thereupon, this trained model was used to generate predictions on unlabeled data by giving probabilities for each class value.
From the experience that was accumulated from this thesis, it was distinguished that supervised learning contains four different phases. The first two are the data preparation and the training step, where the model iterates though the whole dataset in order to configure the model parameters and reach the optimal state for this task. The other two are, the evaluation step which involves the testing of the trained model on a different dataset and the prediction deployment that produces class probabilities for a new unlabeled dataset. In case the output from the evaluation shows low accuracy or great loss, it is necessary to return back to training step and change either the data structure or the architecture of the model. Figure 2.1 illustrates the procedure, where the extracted features are given to the algorithm and once the trained model is created, it predicts the class of unlabeled data.

![Figure 2.1: Depiction of the training phase and the prediction of class values for unlabeled data by the trained model (nltk.org)](image)

### 2.1.2 Dataset types

Throughout the training and testing of the models, three different dataset types were used to avoid any overlaps and to produce reliable results. In their research paper, Dobbin and Simon [2011] talk about the importance of using different datasets for each phase of supervised learning and how this affects the output of a trained model. The dataset types are:

**Training set:** This is the portion of data with the most entries that is used for the training of the models. The instances are labeled, in order for the model to be able to understand the relation between the other values and the class.

**Validation set:** The training of the models was evaluated with this set. From the validation process, a set of metrics is generated that describes the performance of the model (accuracy, recall, precision).

**Testing set:** This dataset does not contain labels and is used by the predictor module of the project. The predictor creates a set of probabilities for each class value of a link.
by using the trained model. Figure 2.2 displays the proportions that each dataset should have compared to the other two.

<table>
<thead>
<tr>
<th>Original Set</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>Testing</td>
</tr>
<tr>
<td>Training</td>
<td>Validation</td>
</tr>
</tbody>
</table>

**Figure 2.2:** Dataset proportions (rpubs.com)

### 2.1.3 Machine Learning metrics

During the validation of a trained model, a set of metrics is calculated to evaluate its performance. Based on the correctness of a classification, there are four different statuses. As shown in Figure 2.3, when the model correctly predicts that an instance belongs to a label or not, then that action is characterized as a *true positive* or a *true negative*. Meanwhile, when the prediction is a false estimation, then the categorization is described as either a *false positive* or *false negative*.

**Figure 2.3:** Classification statuses that indicate the correctness of a classification (mathworks.com)

In his paper, Japkowicz [2006] defines equations that use the above statuses, to calculate more complex metrics that describe the performance of the trained model during the validation process. Equation 2.1 describes the *accuracy* of a model that represents the total number of correct categorizations divided by the total number of data entries. Also, Equation 2.2 defines the *precision* of a model which is equal to the number of correct categorizations for a certain class divided by the total number of categorizations for the same class. In the final Equation 2.3, the number of correct allocations to a class divided by the correct and false allocations gives the *recall* value.
\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}
\]

\[
\text{Precision} = \frac{TP}{TP + FP} \tag{2.2}
\]

\[
\text{Recall} = \frac{TP}{TP + FN} \tag{2.3}
\]

2.1.4 Machine Learning issues

According to L’heureux et al. [2017], the main difficulties for machine learning come from the complexity of a task and the dataset in use. The issues this section presents, mostly relate to dataset idioms and model architectures.

**Imbalanced data:** In his research, Chawla [2009] refers to this issue and describes how extreme quantity ratios between different class values make a model really good at predicting the majority class, but less effective for the rest of the categories. Furthermore, Tang et al. [2009] and Glauner et al. [2016] state that this imbalance affects metrics like accuracy or recall which are used in most cases and because of that, many projects demonstrate high but unreliable metrics. The dataset that was used for relevance detection, was in this state (97% irrelevant links) and thus, it was preprocessed and reduced in size in order to have an equal number of data entries for both classes.

**Over-fitting:** In their work, Cawley and Talbot [2010] talk about over-fitting and they define it as the state where the model is overly optimistic that it has made a correct categorization. This problem is deeply connected to imbalanced data and is caused by the misconfigured training of the model.

**Model selection:** Initially, when a task is first explored it is really important to select a proper model architecture, in order to achieve a satisfactory measure of success. A research paper that describes an attempt to reduce the validation loss of a model is given by Shalev-Shwartz and Ben-David [2014].

2.2 Text Classification

With the expansion of the web and the increase in documents of text, text classification became a popular technique that involves the connection of a sequence of words to a predefined category (Ker and Chen [2000]). Ikonomakis et al. [2005] mentions that text
Chapter 2. *Background Knowledge*

classification is a special case of supervised learning and in this case, an initial training dataset is also required.

The main obstacle of text classification is the ambiguity of human language which is represented by two terms: polysemy and synonymy. Polysemy refers to the fact that a word can have many meanings and synonymy represents the case were multiple words can have the same meaning (McCaughren [2009]). Another issue is the representation of sequences, where the number of features (words or phrases) can increase the dimensionality of a task (Leopold and Kindermann [2002]). To decrease this effect, either a subset of the initial features is taken (Brank et al. [2002]) or new features are created by modifying the first (Han et al. [2004]).

Figure 2.4 demonstrates the basic steps of text classification.

- **Dataset iteration:** For this step, a dataset reader goes through every data item and extracts a set of features.

- **Text tokenization:** A method inside the dataset reader takes a sentence and creates a token for each word.

- **Stemming:** As reported by Moral et al. [2014], stemming removes misspelled words or words that contain the same stem (e.g. function, functional, functionality) and keeps the most common as a feature. Also, this technique helps with the reduction of the vocabulary size and is performed by the word tokenizer of the AllenNLP library.

- **Vector Representation of text:** Due to the format of text, most practitioners populate vectors with sequences of words or characters to create the vocabulary of a dataset and try to minimize its size at a reasonable level, as long as the accuracy of the trained model does not decline. In section 2.2.1, a way to improve these representations is described.

- **Stopword deletion:** In line with Madsen et al. [2004], stopwords are the most common terms in human language that do not contribute to the training of the model. In this implementation, they are removed during the preprocessing of data.

- **Feature selection:** Conforming to Forman [2003], feature selection deals with the reduction of the used fields based on their relevance and decreases the dimensionality of the task. In section 8.2, I use a technique that assists with the discovery of the best feature set.

- **Feature transformation:** Reduces the size of the vocabulary (Han et al. [2004]) by discovering features co-occurrences. Principal Component Analysis is a famous
technique for feature transformation (Zu et al. [2003]) that tries to learn a transformation matrix to reduce the complexity of the classification task without losing accuracy (Ikonomakis et al. [2005]). This technique is also known as Latent Semantic Indexing.

- **Learning algorithm:** This algorithm uses the preprocessed features to train the model that will be used later for predictions. According to Danso et al. [2014], a large number of class values negatively affects the difficulty of the classification and for such tasks, a larger dataset with many samples for each value is needed. Moreover, the types of class values (nominal, ordinal, interval, ratio) and their order in a dataset can also affect the validation metrics of a trained model (Frank and Hall [2001]). Section 2.3 gives more information about this step.

![Figure 2.4: The different steps of text classification, that start from the reading of a document and lead to use of the features by the learning algorithm](image)

### 2.2.1 Global Vectors for Word Representation

As reported by Pennington et al. [2014], GloVe is a word representation model that captures global corpus statistics like word occurrence in the same context and with these it tries to extract meaning or relationships between different words. An approach that was used before the emergence of GloVe and is still popular involves prediction models for meaning extraction like skip-gram and latent semantic analysis (Baroni et al. [2014]), but the GloVe model utilizes both the word analogy task and corpus statistics in a very efficient way. In practice, if $X_{ij}$ is the word occurrence matrix that tabulates the number of times word $j$ appears in the context of word $i$, then equations 2.4 and 2.5 represent the main functionality of this model.

\[
X_i = \sum_k X_{ik} \quad (2.4)
\]

\[
P_{ij} = P(j|i) = X_{ij}|X_i \quad (2.5)
\]
Chapter 2. Background Knowledge

The first equation calculates the occurrences of any word in the context of i and the second finds the probability of occurrence for word j in the context of word i. Figure 2.5 shows examples of related terms.

Figure 2.5: Linear substructures (Pennington et al. [2014])

2.2.2 Bag of words

In his paper, Peter et al. [2016] defines bag of words as a model that is broadly used to simplify sentence representations in NLP by putting words in a set and ignoring their order or grammar. An example of a BoW in JSON format is the following:

Konstantinos went for a walk yesterday. Helen went for a walk as well.

BoW = {"Konstantinos":1,"went":2, "for":2,
      "a":2,"walk":2,"yesterday":1,"Helen":1,"as":1,"well":1}

The main use of BoW is feature generation (search of patterns in sentences). For feature generation, it is necessary to predefine the strategy that is going to be followed. An important feature that many implementations of counterfeit detection used is frequency search (number of times a word appears in a text).

<table>
<thead>
<tr>
<th>BoW strategy examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collecting all words of the sentence in one bag-of-words except the currently aligned word.</td>
</tr>
<tr>
<td>Collecting all preceding words in one bag-of-words and all succeeding words in a second bag-of-words.</td>
</tr>
<tr>
<td>Collecting all preceding words in one bag-of-words and all succeeding words in a second bag-of-words except those already included in the source window.</td>
</tr>
</tbody>
</table>
2.3 Artificial Neural Networks

As stated in the report by Goodfellow et al. [2016], an ANN is a computational model that has been inspired by the way the human brain operates. It is consisted of neurons which receive an input from neurons in previous layers and produce an output. For each input the neuron applies a proportionate weight depending on its importance. An additional constant value called bias is also applied to shift the activation function and assist the learning process. The activation function is applied over all of the values in Figure 2.6 to make the training process non-linear.

\[ y = \alpha x \]

Figure 2.6: Neuron structure (ujjwalkarn.me)

2.3.1 Activation functions

"At the heart of every deep network lies a linear transformation followed by an activation f(.). The activation function plays a major role in the success of training deep neural networks." (Ramachandran et al. [2018])

**Linear function:** It is a polynomial function that has a degree not greater than one (Stewart [2012]). This is one of the activation functions that were used in the model and is demonstrated by equation 2.6.

\[ y = \alpha x \]

**Softmax function:** Also known as normalized exponential is a probability distribution (Bishop [2012]) that is used for softmax regression and deals with categorizations that have multiple class values. Softmax is used in the forward function of the classifier implementation in both detections. Its functionality is described by the equations 2.7 and 2.8.
Chapter 2. Background Knowledge

\[ \sigma : \mathbb{R}^n \rightarrow \text{int}(\Delta^{n-1}) \]  \hspace{1cm} (2.7)

\[ \sigma(z) := \frac{\exp(\lambda z)}{\sum_{j=1}^{n} \exp(\lambda z)} , \lambda > 0 \]  \hspace{1cm} (2.8)

**Rectified Linear Unit:** In agreement with Jarrett et al. [2009] and Nair and Hinton [2010], ReLU is considered the most popular activation function, thanks to its simplicity and effectiveness. Its functionality is defined by equation 2.9.

\[ f(x) = \max(x, 0) \]  \hspace{1cm} (2.9)

In his research, Krizhevsky et al. [2012] says that the inclusion of ReLU in model architectures, made deep neural networks a top choice for supervised learning. ReLU is also the second activation function that was used in this implementation.

### 2.3.2 Dropout

Dropout is a technique for deep learning models that is used to reduce the chance of having over-fitting. In order to reduce the weight over-adaption, this algorithm randomly removes neurons from the training process along with their connections (Srivastava et al. [2014]). Figure 2.7 shows the aforementioned procedure.

![Figure 2.7: Dropout Neural Net Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped. Srivastava et al. [2014]](image)

### 2.3.3 Feed-forward neural networks

A feedforward neural network has many nodes contained in three different types of layers.
• **Input layer**: It receives inputs from its environment and passes them to the hidden layers. No calculations take place inside this layer.

• **Hidden layer**: Inside these layers the usual neuron actions are performed and then the results are given to the output layer.

• **Output layer**: This layer performs some extra calculations and then delivers the outcome of the whole process.

Moreover, extra weights exist in the connections between the neurons of different layers and the data only move forward. Depending on the number of hidden layers, this type of network can be characterized as Single layer perceptron or Multilayer perceptron.

![Feed-forward neural network](https://ujjwalkarn.me)

**Figure 2.8**: Feed-forward neural network (ujjwalkarn.me)

### 2.3.4 Recurrent neural networks

A recurrent neural network has the same structure as a feed-forward neural network, but each neuron also has a closed loop that provides feedback (Fausset [1994]). There is also a fully connected version, that does not have an input layer and each neuron receives the output of the other interconnected neurons (Medsker and Jain [1999]).

The RNN model is ideal for sequential or varying time pattern learning and this is one of reasons that it performs so well with text classification (Liu et al. [2016]). It also doesn’t have an input quantity threshold and it can process the information up to the current time frame, in contrast to multilayer perceptrons (Schuster and Paliwal [1997]). Future inputs which appear after the current time frame can also be used for prediction
(Schuster and Paliwal [1997]) and that in turn helps with improving the performance of a system (Robinson [1994]).

For this implementation, a bidirectional recurrent neural network was used to encode features. This approach takes into consideration both the right-wise and left-wise time directions (Cui et al. [2018]) and removes the prediction limitations of recurrent neural networks.

![Recurrent Neural Network vs Feed-Forward Neural Network](towardsdatascience.com)

**Figure 2.9: Comparison of a recurrent neural network and a feed-forward neural network (towardsdatascience.com)**

### 2.3.5 Long short-term memory

The LSTM model was originally invented in order to overcome loss of gradients during the training process (Hochreiter and Schmidhuber [1997]). Its functionality is very similar to recurrent neural networks, but instead of having regular neurons in the hidden layer, it uses another type of unit called memory cell. The following characteristics apply to every LSTM.

- Memory cells are composite units of simpler nodes that use recurrent edges to maintain gradients across many steps (Lipton et al. [2015]).

- A memory block is a set of memory cells which contains three different types of gates that control the flow of information (Sak et al. [2014]), the input, output and forget gates.

- Input gates control the input of memory cell and output gates deal with the activation outputs to the rest of the network (Sak et al. [2014]).

- Forget gates evaluate the condition of a cell and whether they should reset its memory or not (Gers et al. [1999]).
• Each cell of a block is able to get feedback from the gates in order to gain information about the timing of outputs (Gers et al. [2002]).

To summarize, “a LSTM network computes an input sequence to an output sequence by calculating the network unit activations” (Sak et al. [2014]).
Chapter 3

Related work

As Sculley et al. [2011] and Wartner et al. [2015] state, there are not many researches that would help establish some baselines, especially for supervised learning approaches with deep learning models. This implementation uses some of the embeddings (bag of words) and methodologies (reviewer supervision) which are mentioned, takes advantage of different feature types (string-based, non-textual content-based, account-level) and uses a deep learning model that utilizes different technologies like feed-forward neural networks and LSTMs. As a result, an algorithm with state of the art techniques that recasts this task on text classification has been created.

3.1 Adversarial advertisement detection

This study was conducted by Google scientists and includes not only counterfeit products but also other types of adversarial advertisements like unclear or deceptive billing. In their research they mention some of the main challenges like false positives and false negatives (2.1.3), class imbalance (2.1.4) and unavailability of data and propose a solution for the detection of these advertisements. They also make a brief description on the different types of available features and describe their implementation. In his implementation, Sculley et al. [2011] uses multiple models and performs a multi-class categorization that utilizes support vector machines and cascade models. In cases of great uncertainty, the system directs outputs to experienced reviewers. Lastly, it should be mentioned that this research paper had important ethical and professional insights about counterfeit detection that influenced this project. Figure 3.1 provides a visual representation of the aforementioned approach.
Chapter 3. Related work

3.2 Semi-automatic identification

In his research, Wartner et al. [2015] introduces a semi-automatic methodology for counterfeit product detection that contains unsupervised learning and human reviewers. The workflow involves a user that specifies some search criteria for the data collection and with the use of a scraper, specific features are extracted from html code (Mitchell [2018]). Later on, the dataset instances are clustered per brand by detecting feature similarities and with the use of a scoring function, the likelihood for counter-fitting is estimated. Finally, a pool of potentially counterfeit products is generated and counterfeit specialists proceed to manual verification. It should be mentioned that whenever analysts from SnapDragon start investigating a brand, they also perform data extraction from the html code of e-commerce platforms with a configured scraper for each website. Figure 3.2 depicts the procedures for the semi-automatic identification and the order in which they’re executed.

Figure 3.1: Adversarial advertisement detection with SVM and cascade model utilization (Sculley et al. [2011])

Figure 3.2: Semi-automatic detection of counterfeit products with the use of clustering and reviewer intervention (Wartner et al. [2015])
3.3 Bag-of-word based brand recognition using Markov Clustering Algorithm

In his paper, Benezeth et al. [2015] describes a technique for trademark detection. His algorithm relies on the strong visual identity of products (product color, trademark) and looks for character sequences inside an image.

He used three main technologies for his implementation: The BoW that calculates the frequency of words, the Scale Invariant Feature Transform (SIFT) that extracts features from an image and an unsupervised learning algorithm (Markov Clustering Algorithm). The general idea of this approach is that SIFT performs the feature extraction for every image, the MCA algorithm puts the images in different clusters by analyzing the features and the BoW calculates the word frequencies which are used in a supervised classification that makes the final brand prediction. Figure 3.3 shows the word frequency of two separate products.

![Figure 3.3: Character frequency for two products (Benezeth et al. [2015])](image-url)
Chapter 4

Requirements analysis

In order to reach the milestones mentioned in 1.3, the system must cover the following requirements:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>MoSCoW Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>The datasets should have balanced class values in both cases (relevance /</td>
<td>Should have</td>
</tr>
<tr>
<td>counterfeit detection)</td>
<td></td>
</tr>
<tr>
<td>Each dataset should be split into three different parts (training, validation,</td>
<td>Should have</td>
</tr>
<tr>
<td>testing)</td>
<td></td>
</tr>
<tr>
<td>Investigation on related projects for baselines</td>
<td>Should have</td>
</tr>
<tr>
<td>Model architecture definition (neural network types, embeddings, complexity</td>
<td>Must have</td>
</tr>
<tr>
<td>of model)</td>
<td></td>
</tr>
<tr>
<td>Training with relevant technologies must be commenced</td>
<td>Must have</td>
</tr>
<tr>
<td>Implementation of dataset reader and deep learning model</td>
<td>Must have</td>
</tr>
<tr>
<td>Experiment commencement</td>
<td>Should have</td>
</tr>
<tr>
<td>Result analysis and evaluation</td>
<td>Must have</td>
</tr>
<tr>
<td>Improvement and adjustment of approach</td>
<td>Could have</td>
</tr>
<tr>
<td>Unit and integration testing on software</td>
<td>Should have</td>
</tr>
<tr>
<td>Software documentation</td>
<td>Should have</td>
</tr>
</tbody>
</table>

Table 4.1: Requirements analysis and importance evaluation
Chapter 5

Professional, legal, ethical, and social issues

On the aspect of the implementation and the evaluation of this project, the participation of third parties was not necessary and thus there was no safety issue. The main topic that must be discussed is the outcome of this work. A counterfeit product detection system would be very useful, especially when e-commerce platforms are filled with imitated products that have lower quality and might cause health hazards. If every e-commerce platform had an embedded system like the one that was created for this project, then the quantity of counterfeit products would drop and the revenue loss for several brands would be minimized.

While this system will be able to detect imitated products, it may also at times classify a product as a counterfeit even though it is not. Especially the production of a false positive (falsely classify a link as innocent) or a false negative (falsely categorize product with an infringement) could lead to the justification of a fake product or the incrimination of an original product of a startup brand that tries to make profit and expand. Therefore, it is necessary to ensure the reliability of this framework, in order to avoid any legal problems and to ensure that other systems will not have second thoughts about using such a tool. It is believed that this project will not affect the community in a negative way. On the contrary, it will restore the trust that people have in brands and originality and it will create a better environment in terms of quality.

On my part, I should be objective with the results that I am going to present and not overestimate the performance of the final system. By applying these ideals in my work, I can ensure the integrity of the project and the delivery of trustworthy metrics for future work.
Chapter 6

Methodology

Under the supervision of SnapDragon Monitoring, a set of methodologies was followed for the development of software, which included some state of the art techniques used by many development teams. Because of the change of subject and the creation of new tasks, the initial timetable and the literature review were edited. In sections 6.1, 6.2, 6.3 and 6.4, there are definitions of these methodologies and explanations about their use in this project.

6.1 Agile software development

As Dingsøyr et al. [2012] states in his research, this is a collection of practices that has been created by many experienced consultants and its purpose is the success of the development process despite the uncertainty that might exist. Some basic concepts of this methodology are the user stories, which represent a part of the overall product, the milestones that indicate an important point for the project and the personas - user profiles of future customers. During the whole implementation, this development technique was used to create and prioritize tasks and to avoid getting into deadlocks that might affect the timetable.

6.2 Version Control Systems

Before writing code for this project, a separate repository has been created on a version control system called bitbucket. VCS systems are very useful for projects that involve more than one developer, because they display the commits along with descriptions that highlight the changes. Moreover, merges can be done in case there are conflicts between
multiple code changes and separate branches can be created, in order for different teams to work on their own copy of the software, leaving the master branch intact. All of the above features were needed to share the implementation progress with the rest of the development team and unify the AI API with the Swoop API which is the software that SnapDragon Monitoring has developed. Bitbucket also has many utilities for report generation and project management (Jira).

6.3 Scrum

Scrum is a popular approach for task definition and workload management. According to Lima et al. [2012], its purpose is to break complex methodologies into smaller and manageable steps and to include teams with different roles in the decision making process. The teams that usually take part into this are: the development team, the product owners and the scrum master. In this case, the author belonged to the development team, giving insights on the artificial intelligence part of the project, the product manager of the company had the role of the product owner, which was very important since it determined the goals of each sprint and the technical lead was the scrum master that guides the process and advises the rest of the teams. Figure 6.1 demonstrates both scrum events and artifacts and how they relate to each other.

**Figure 6.1:** Representation of the events and artifacts in the scrum framework (scrum.org)

**Sprint:** It is a pre-defined time period, during which a set of tasks must be completed.

**Sprint planning:** The goals of the next sprint and the workload management are defined in this event.
Daily scrum: On a daily basis, the development team would gather to discuss what was accomplished on the previous day and what each member is going to work on the current one.

Sprint review: At the end of each sprint, there was a discussion about the new features that have been implemented and the product backlog was edited accordingly.

Sprint retrospective: With each of these events, there was a conversation about improvements that could be enacted during the next sprint.

Product backlog: It is a list that contains every feature that should be implemented for the software product.

Sprint backlog: It is a subset of the product backlog that is going to be implemented during the current sprint.

Increment: It is the list of features that have been implemented by the end of a sprint.

6.4 Tests

6.4.1 Unit tests

For each JUnit test that has been created in this project, the output of a function is inspected with the use of assertions. For each assertion, if the output of a function has the same range and type with the expected result, then that single assertion returns a true value. When every assertion has a true value, the test is considered successful. With these tests, a better understanding of the procedures inside the software was achieved and the possibility of an error on a larger scale was reduced. Figure 6.2 shows a unit test that inspects the functionality of the relevance predictor.

6.4.2 Integration tests

While unit tests deal with a single method, integration tests evaluate the interaction between larger parts of the application. According to Chan et al. [2002], an integration test can be conducted based on a collection of criteria like the state transitions of a program, the relationships between events, or even testing against formal specifications which occur with the comparison of two objects of different classes. Luckily, AllenNLP has some test case modules that can be easily used. Figure 6.3 demonstrates an integration test that evaluates the interaction between the relevance classifier and its dataset reader.
Chapter 6. Methodology

Figure 6.2: Unit test that oversees function outputs for the relevance predictor

```python
class TestRelevancePredictor(TestCase):
    def test_user_nameed_inputs(self):
        archive = load_archive('/home/username/Desktop/ali4nlp_outputs/brand_output_dir_complete/model.tar.gz')
        predictor = Predictor.from_archive(archive, 'relevance-classifier')
        file_path = '/home/username/Desktop/trtl_dict.json'
        with open(cached_path(file_path), 'r') as file:
            json_dict = json.load(file)
            for i in range(3):
                inputs = {
                    'search term': json_dict['search term'][i],
                    'description': json_dict['description'][i],
                    'price category': json_dict['price category'][i],
                    'seller': json_dict['seller'][i]
                }
                result = predictor.predict_json(inputs)
                label = result.get('label')
                print(inputs)
                print(result)
                assert label in ['False', 'True']
                class probabilities = result.get('class probabilities')
                assert class probabilities is not None
                assert all(cp > 0 for cp in class probabilities)
                assert sum(class probabilities) == approx(1.0)
```

Figure 6.3: Integration test on the deep learning model

```python
from allennlp.common.testing import ModelTestCase
class RelevanceClassifierTest(ModelTestCase):
    def setUp(self):
        super(RelevanceClassifierTest, self).setUp()
        self.setup_model('/home/username/swoop-machine-learning/config_files/relevance_classifier.json',
                         '/home/username/Desktop/data/verify-test-Glencairn.json')
        def test_model_can_train_save_and_load(self):
            self.ensure_model_can_train_save_and_load(self.param_file)
```
Chapter 7

Implementation

7.1 Architecture rationale

The software that was created, is intended for the classification of product links inside the Swoop platform which is a property of SnapDragon Monitoring. The general idea is that the AI API will act as a mediator that takes link data and returns only the relevant as output. This way, the reviewing of links by counterfeit specialists and the generation of new labeled data is becoming easier, since the results will be in context with the brand that they’re dealing with.

The second functionality that the model provides is counterfeit product detection. The second categorization takes the relevant links that the first classifier provides and predicts if they are innocent/counterfeit. In case a link has the second class value, then the type of infringement is defined, as discussed in 1.2. It should also be mentioned that the implementation of both classifiers was easier, because they both used the same set of features and thus the project has a single classifier implementation and two link readers.

The python library that was used for this implementation is called AllenNLP. The mentioned framework, is built on top of the PyTorch library and provides modules for rapid NLP development. More details about its features will be given, along with the choice of training models and the utilized techniques in the next sections. Figure 7.1 gives a representation of the Swoop ecosystem that utilizes the AI API, a scraper for data collection, the amazon cloud service and the main platform of Swoop.
7.2 Model

Figure 7.2, depicts the structure of the model which is used in both relevance and counterfeit detection. Every feature except from the price of a product is initially converted to a collection of tokens that later the algorithm encodes with the use of either a LSTM or a BoW. The price value is passed to the neural network as it is, because of its numeric format and there is no need for tokenization or encoding. Then, all of these features are given as input to the feedforward neural network and an output is generated that differs for each classification. For relevance detection, the classification is binary because it has only two class values (relevant, irrelevant) and counterfeit detection, handles four different class values (innocent, trademark, copyright, design) that describe if a product is genuine or if it has one of the three types of infringement aforementioned in 1.2. Equation 7.1 describes the functionality of the model and how its output is dependant to the six features.

\[
P_{out} = p(k|n_1, ..., n_i)
\] (7.1)
7.3 Implemented technologies

7.3.1 AllenNLP

As discussed in 7.1, AllenNLP is a library that utilizes deep learning for natural language processing. Its three main aspects are the dataset reader, the deep learning model and the configuration files. The dataset reader, reads and tokenizes multiple features of a dataset for better processing of data by the model. Then, the feature tokens are placed in a Field variable (e.g. TextField) that inherits methods for batching and padding (these techniques help with the homogeneity of inputs) and many feature fields are placed in a single instance variable.

Afterwards, the classifier receives these instances, embeds the tokens of a field with the use of the embedder class (e.g. TextFieldEmbedder), transforms the text into a binary sequence with the use of a mask function and a vocabulary and encodes the fields. For the final phase, the forward function feeds the encoded features to the neural network and based on the process that utilizes this method, an output is produced (metrics or class probabilities). The classification process can be edited without changing the model implementation with the use of configuration files.

Configuration files are taken into consideration at the very beginning of a procedure and that is the reason they are always included in the CLI commands shown in section 7.4.6. Whenever the train command is called, the software accesses the configuration file, it takes the configuration parameters with the use of the Registry class and creates objects of the DatasetReader and Classifier classes with the from_params method. According to Gardner et al. [2017], this workflow allows fast-paced experimentation with different combinations of encoders, embedders and model architectures.
7.3.2 Amazon Rekognition

One of the features that both of the implemented classifiers use is a string that has concatenated common labels of product and brand images. Since this implementation concentrates on text classification, the image tags are extracted with the use of Amazon Recognition and embedded in the datasets. Amazon recognition is an image classifier that recognizes traits like objects, text, activities and faces in an image and returns their tags as output. Figure 7.3 shows the procedure that the image classifier goes through to return the output.

![Tag extraction process for Amazon Rekognition](aws.amazon.com)

**Figure 7.3:** Tag extraction process for Amazon Rekognition (aws.amazon.com)
7.4 Code review

7.4.1 Link Reader

The Link Reader module is used for feature extraction from a JSON file and the preparation of the data for the classification. Although there are two link readers (relevance, counterfeit detection), only one of those is demonstrated because of their similarity. In appendix B.2 there is the implementation of the reader with the following functions:

<table>
<thead>
<tr>
<th>Method</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
<td>It works as the constructor of this class and initializes the tokenizer and the laziness of the dataset reader. The tokenizer is an object that is used to tokenize sentences and the laziness defines the access method to the dataset (e.g. all samples at once).</td>
</tr>
<tr>
<td>_read</td>
<td>Accesses the dataset file and by iterating through every instance it forwards the features to text_to_instance function.</td>
</tr>
<tr>
<td>tokenize</td>
<td>This function is called inside text_to_instance, splits sentences to words and puts them into a vector.</td>
</tr>
<tr>
<td>text_to_instance</td>
<td>It first tokenizes the features and then returns an instance. An instance is a dictionary that contains multiple feature names and tokenized feature values.</td>
</tr>
<tr>
<td>from_params</td>
<td>It receives the parameters (tokenizer, token_indexers) from the configuration file and creates a link reader object. In this implementation only the default values from the initializer are used.</td>
</tr>
</tbody>
</table>

Table 7.1: Link reader methods
7.4.2 Classifier

This class is the core of the software which defines the functionality of the deep learning model. Its main operation is in the forward method and it is the same one aforementioned in 2.3.3. The AllenNLP library uses data structures (e.g. LongTensor) and methods (e.g. Functional.softmax) from the PyTorch library for this functionality. In appendix B.1, an implementation of this module is provided and table 7.2 gives the description for each function.

<table>
<thead>
<tr>
<th>Method</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
<td>Initializes properties like the text field embedder, the feature encoders, the neural network type and defines the metrics (accuracy, f1_measure, cross entropy loss) for the validation output.</td>
</tr>
<tr>
<td>forward</td>
<td>This method represents the feed-forward process of the neural network. Thus, it receives the tokenized features, it reprocesses them (embedding, encoding) and sends them to the classifier for the categorization. In the end, this method produces a series of metrics and probabilities (for prediction) for every class value.</td>
</tr>
<tr>
<td>get_metrics</td>
<td>It extracts each metric from the dictionary where they are saved and presents them in an appropriate manner.</td>
</tr>
<tr>
<td>decode</td>
<td>It produces the output for the probability representation in prediction.</td>
</tr>
<tr>
<td>from_params</td>
<td>This method gets the parameters from the configuration file and creates a classifier object.</td>
</tr>
</tbody>
</table>

Table 7.2: Classifier methods
7.4.3 Predictor

This module is used once the trained model is prepared. Currently, AllenNLP provides only an implementation that receives a JSON dictionary as an input and under those circumstances another module that changes the dataset format for the predictor was created. The code of this module can found in sections B.4, B.5 and table 7.3 contains its definition.

<table>
<thead>
<tr>
<th>Method</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>_json_to_instance</td>
<td>Receives a feature dictionary, creates an instance by using the _text_to_instance method from the link reader and returns the probabilities of an instance belonging to any of the class values.</td>
</tr>
<tr>
<td>use_predictor_on_dataset</td>
<td>By providing the locations of a trained model and a test dataset along with the name of the predictor in use, multiple predictions are performed on a dataset.</td>
</tr>
</tbody>
</table>

Table 7.3: Predictor methods

7.4.4 Dictionary Generator

This module is mostly used for turning the initial dataset into a JSON dictionary for the prediction process. Its code can be found in appendix B.7 and it includes the following functions:

<table>
<thead>
<tr>
<th>Method</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_json</td>
<td>Opens a dataset file and stores the desired features to a class variable</td>
</tr>
<tr>
<td>create_json_dict</td>
<td>Saves the JSON dictionary to a file.</td>
</tr>
<tr>
<td>return_json_dict</td>
<td>Returns the class variable that contains the feature dictionary.</td>
</tr>
</tbody>
</table>

Table 7.4: Dictionary generator methods
7.4.5 Configuration file

The configuration file located in B.6, provides a fast way to define various elements for both the link reader and the classifier and removes the need for complex implementations of neural networks. As mentioned in 7.1, AllenNLP is an ideal library for rapid NLP development and the main reason is the existence of this file that gives the flexibility to construct sophisticated deep learning models in a smart and fast way. Table 7.5 provides a description for every configuration parameter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset_reader</td>
<td>With this parameter, the algorithm detects the link reader module. It achieves this by making use of the annotation above the class declaration of LinkReader.</td>
</tr>
<tr>
<td>train_data_path</td>
<td>It includes the destination of the training dataset.</td>
</tr>
<tr>
<td>validation_data_path</td>
<td>It takes the destination of the validation dataset.</td>
</tr>
<tr>
<td>model</td>
<td>It contains specifications for various parameters of the model that will be discussed next.</td>
</tr>
<tr>
<td>text_field_embedder</td>
<td>Defines the properties of the embedder (tokens, trainable, token embedding dimensions, glove dataset location).</td>
</tr>
<tr>
<td>encoder parameters</td>
<td>The number of these parameters is the same with the encoded features and for each, a type is defined (boe, lstm) based on the size / complexity of the feature.</td>
</tr>
<tr>
<td>classifier_feedforward</td>
<td>Here, the number of inputs, the number of layers, the activation functions and the dropout per layer are defined for the classifier.</td>
</tr>
<tr>
<td>iterator</td>
<td>It defines the way the data are put into buckets, the size of the batches and the sorting method.</td>
</tr>
<tr>
<td>trainer</td>
<td>Describes the number of dataset iterations, the patience for non improved metrics, the use of cpu / cuda, the gradient clipping, the validation metrics and the optimizer in use.</td>
</tr>
</tbody>
</table>

Table 7.5: Configuration parameters

7.4.6 Shell scripts

In sections B.9 and B.10, two shell scripts that take advantage of the command line interface for AllenNLP are given. The first is used to train a model, while the second takes a trained model and evaluates it on a testing set. The command format is given in table 7.6.
## CLI commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td><code>allennlp train [configuration file] -s [output destination] -include-package [module name]</code></td>
</tr>
<tr>
<td>Evaluate</td>
<td><code>allennlp evaluate [trained model] --evaluation-data-file [test set] -include-package [module name]</code></td>
</tr>
</tbody>
</table>

**Table 7.6:** CLI commands
Chapter 8

Experiments and results

8.1 Experimental setup

The use of a questionnaire was not needed for this type of project, but it was necessary to test the performance of the models with a series of experiments. Initially, the generated metrics from the experiment in 8.2, where both of the models were used with different feature sets, were a first indication of the rate of success. In experiment 8.3, both classifiers are set linearly and the output of relevance detection is led to counterfeit detection, in order to validate the performance of the whole pipeline. In experiments 8.4 and 8.5, models that make predictions for multiple brands were created. The computational load of the training was so high, that a remote cuda enabled amazon instance was used for the tryouts. With the use of the remote instance, the experimentation phase lasted for only 1/3 of the estimated time.

8.2 Ablation tests

The ablation test is a trustworthy technique that helps finding the best combination of features for the classification process (Stymne [2016]). During this experiment, a model was trained with the complete feature set and by reducing the feature set size by one per tryout, the best collection of features was selected. The next subsections show the performances of the different models for both relevance and counterfeit detection.
8.2.1 Relevance detection results

According to table 8.1, the feature with the highest accuracy is common labels. This behavior was expected, because the image tags of both a brand and a link are immediately compared and the relevance of these two entities is mostly determined by the tag resemblance. Search terms and description also have high accuracy, since the classifier related the most used words in both features to a class value.

<table>
<thead>
<tr>
<th>Features</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>common labels</td>
<td>0.976</td>
<td>0.977</td>
<td>0.974</td>
</tr>
<tr>
<td>search terms</td>
<td>0.889</td>
<td>0.974</td>
<td>0.800</td>
</tr>
<tr>
<td>description</td>
<td>0.873</td>
<td>0.809</td>
<td>0.976</td>
</tr>
</tbody>
</table>

Table 8.1: Single feature metrics for relevance detection

For the tryouts with two features, all of the three top sets contained the common labels feature. The interesting part of these results is that one of the combinations also contains the seller feature. Thus, the model produced high metrics by partially relating the relevance of a product to its seller. Table 8.2 shows those results.

<table>
<thead>
<tr>
<th>Features</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>search terms, common labels</td>
<td>0.985</td>
<td>0.985</td>
<td>0.986</td>
</tr>
<tr>
<td>seller, common labels</td>
<td>0.968</td>
<td>0.978</td>
<td>0.957</td>
</tr>
<tr>
<td>description, common labels</td>
<td>0.970</td>
<td>0.945</td>
<td>0.998</td>
</tr>
</tbody>
</table>

Table 8.2: Metrics for two features for relevance detection

For the three feature training, price also contributed in two of the top combinations, even though it had the lowest performance for the single feature tryouts. From this attempt, it is visible that if the price is combined with other features, then the model achieves more successful categorizations. Table 8.7 demonstrates these results.

<table>
<thead>
<tr>
<th>Features</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>search terms, seller, common labels</td>
<td>0.982</td>
<td>0.980</td>
<td>0.983</td>
</tr>
<tr>
<td>seller, sigmoid price, common labels</td>
<td>0.978</td>
<td>0.973</td>
<td>0.983</td>
</tr>
<tr>
<td>search terms, sigmoid price, common labels</td>
<td>0.986</td>
<td>0.977</td>
<td>0.995</td>
</tr>
</tbody>
</table>

Table 8.3: Metrics for three features for relevance detection

In the end, the metrics of the complete feature set did not exceed the top metrics of the second tryout. Based on that, now the concept of feature selection becomes more clear and the best feature set for relevance detection has been found. The final results are visible in table 8.4.
Chapter 8. *Experiments and results*

### 8.2.2 Counterfeit detection results

The performance of counterfeit detection was not as high as with the first categorization. The reason for that was the class value variance. In the first case, the classification was binary because it had only two class values (relevant/irrelevant). But for the second categorization, the identification of a counterfeit product is not enough and the type of infringement (copyright, trademark, design) is also needed.

From table 8.5, the three most valuable features for counterfeit detection are given, with the description achieving the best accuracy.

<table>
<thead>
<tr>
<th>Features</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>0.746</td>
<td>0.702</td>
<td>0.623</td>
</tr>
<tr>
<td>platform</td>
<td>0.699</td>
<td>0.844</td>
<td>0.474</td>
</tr>
<tr>
<td>common labels</td>
<td>0.678</td>
<td>0.525</td>
<td>0.598</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Features</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>platform, description</td>
<td>0.803</td>
<td>0.723</td>
<td>0.837</td>
</tr>
<tr>
<td>search terms, description</td>
<td>0.768</td>
<td>0.712</td>
<td>0.694</td>
</tr>
<tr>
<td>description, common labels</td>
<td>0.797</td>
<td>0.727</td>
<td>0.738</td>
</tr>
</tbody>
</table>

The next ablation tryout metrics in table 8.6, look more promising. The accuracy metrics for all three cases are close to 80% and the possibility for false categorizations has dropped.

<table>
<thead>
<tr>
<th>Features</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>platform, description</td>
<td>0.803</td>
<td>0.723</td>
<td>0.837</td>
</tr>
<tr>
<td>search terms, description</td>
<td>0.768</td>
<td>0.712</td>
<td>0.694</td>
</tr>
<tr>
<td>description, common labels</td>
<td>0.797</td>
<td>0.727</td>
<td>0.738</td>
</tr>
</tbody>
</table>

The categorization with three features, had the best validation accuracy along with the five feature set, but the precision and recall metrics were lower. Also, platform and description appear in the majority of the top feature sets.

Evidently, the best feature set for counterfeit product detection has a total of five features according to table 8.8.
Chapter 8. Experiments and results

### Table 8.7: Metrics for three features for counterfeit detection

<table>
<thead>
<tr>
<th>Features</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>platform, description, common labels</td>
<td>0.816</td>
<td>0.725</td>
<td>0.808</td>
</tr>
<tr>
<td>platform, description, sigmoid price</td>
<td>0.805</td>
<td>0.725</td>
<td>0.808</td>
</tr>
<tr>
<td>platform, search terms, description</td>
<td>0.803</td>
<td>0.707</td>
<td>0.831</td>
</tr>
</tbody>
</table>

### Table 8.8: Best metrics for different feature set sizes for counterfeit detection

<table>
<thead>
<tr>
<th>Features</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>complete set</td>
<td>0.810</td>
<td>0.745</td>
<td>0.827</td>
</tr>
<tr>
<td>5</td>
<td>0.816</td>
<td>0.747</td>
<td>0.819</td>
</tr>
<tr>
<td>4</td>
<td>0.813</td>
<td>0.745</td>
<td>0.802</td>
</tr>
<tr>
<td>3</td>
<td>0.816</td>
<td>0.725</td>
<td>0.808</td>
</tr>
<tr>
<td>2</td>
<td>0.803</td>
<td>0.723</td>
<td>0.837</td>
</tr>
<tr>
<td>1</td>
<td>0.746</td>
<td>0.702</td>
<td>0.623</td>
</tr>
</tbody>
</table>

### 8.3 Pipeline test

After the creation of both classifiers and the inspection of their functionality, their combined performance as a pipeline was tested. Initially, a dataset was given to the relevance classifier as input and then the links that were classified as relevant were sent to the second classifier for counterfeit detection. Then, the final results were compared to the gold data and the rate of correct classifications was estimated. The comparison showed that 1591 out of 1707 links were correctly classified for relevance and 1351 out of 1610 links were put in the correct infringement category. Table 8.9 shows the performance of the pipeline.

<table>
<thead>
<tr>
<th>Detection</th>
<th>Accuracy</th>
<th>True Positives / Negatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>relevance</td>
<td>0.932</td>
<td>1591 out of 1707</td>
</tr>
<tr>
<td>counterfeit</td>
<td>0.839</td>
<td>1351 out of 1610</td>
</tr>
</tbody>
</table>

### Table 8.9: Generated metrics for both categorizations in the pipeline test

### 8.4 Relevance classification for multiple brands

The experiments up to this point only dealt with a single brand, but the main interest is to create a trained model that can perform predictions for multiple customers. So for this experiment, a dataset that contains products from six different brands was used. The problem was that the company could not provide a dataset with every infringement class value and thus this experiment was conducted only for the first classifier. According to table 8.10, the validation metrics were satisfactory but the loss was very high and that shows how much the difficulty increases for this task. An observation from this
test is that when additional brands are included, the possibility of a false classification increases, mainly because of the similarity for some features across different brands.

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.895</td>
<td>0.858</td>
<td>0.946</td>
</tr>
</tbody>
</table>

Table 8.10: Performance for relevance classification with multiple brands

8.5 Single infringement detection across different brands

Because of data unavailability for the copyright and design infringements, a dataset that contains examples of innocent and trademark infringed products for 111 brands was used. The validation metrics were better than the relevance detection of six brands, even though the involved brands for this experiment are many more. The results of this test, show the great distinction between a genuine and an imitated product and because of that a generic classifier for counterfeit detection is not an impossible task. Table 8.11 shows the validation accuracy, precision and recall for the single infringement detection.

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.924</td>
<td>0.896</td>
<td>0.914</td>
</tr>
</tbody>
</table>

Table 8.11: Performance for relevance classification with multiple brands

8.6 Evaluation

All of the objectives in 1.3 and the requirements in 4 have been met with the completion of this phase.

For the ablation tests in 8.2, the most valuable single features and pairs have been determined for both detections. From the validation metrics, it was also visible that counterfeit detection is a harder task compared to relevance detection but the results in each case were very encouraging.

For the pipeline test in 8.3, the results proved that both models can be used together in order to automate the complete procedure performed by the counterfeit specialists. But it should be mentioned that this experiment was performed for a single brand and possibly the accuracy will drop in a case with many different products.

The other two tests for many brands in 8.4 and 8.5 also had significant metrics and the tryout for counterfeit detection achieved greater accuracy, because its class values were reduced to two (innocent, trademark). Also, the first of the two tests might have achieved a non realistic accuracy due to the similarities of values across different brands.
Chapter 9

Conclusion and Future Work

9.1 Conclusion

For this project, two classifiers have been created for relevance and counterfeit product detection with the use of the AllenNLP library which utilizes deep learning for natural language processing. The algorithm uses state of the art technologies like LSTMs and bag of words for feature encoding and a deep neural network as the main classifier in both cases. It also uses dropout for minimizing the over-fitting risk, a glove dataset to improve the word representations and the adagrad algorithm that optimizes the gradient descent. Moreover, the model uses common features that every product has (description, seller, price, e-commerce platform) and features which are provided by the Swoop platform and Amazon Rekognition (search terms, common labels). A lot of effort was also required for data extraction and preprocessing, because the data had imperfections like falsely categorized instances and class imbalance.

Additionally, a series of experiments has also been conducted to gain further insight on the task and evaluate the performance of the model. The ablation tests in 8.2 showed that with the current feature set, the most valuable features were the search terms, the description and the common labels and the feature with the least contribution was the price. The first insight was expected since the neural network finds common words between the search terms and the description and compares the image labels for a brand and a link. But it was not expected for the price to produce lower metrics compared to the rest of the features based on the information given by the counterfeit specialists. According to the ablation tests with many features, the price contributes more to the model training when it is combined with other features.

Furthermore, the experiment in 8.3 dealt with the performance measurement of the whole pipeline. The metrics were lower if they are compared to the separate testing of
the counterfeit detection, but they were still satisfactory. In 8.4 and 8.5, an attempt to perform both detections on testing sets that contained multiple brands was also made. For counterfeit detection, the test estimated the ability of the model to detect only the trademark infringement across multiple brand instances, because of the unavailability of examples for the rest of the infringements.

Overall, every objective mentioned in section 1.3 has been completed and all of the requirements in section 4 have been met, making this project a complete effort for counterfeit detection.

9.2 Future work

As it has been mentioned before, the data collection was the biggest obstacle for this research and because of that several mitigation strategies were used. For the next attempts on counterfeit detection, the use of sets that contain multiple instances with distinctive feature values would allow the construction of more precise and trustworthy models that make less false categorizations. Another great addition would be the inclusion of extra features like item quantity and the use of different model architectures that might be more effective for this task.

An important point for this project is that with the current implementation, the model can only deal with labeled data. If the platform was to interact with a new brand, the intervention of counterfeit specialists would be inevitable. Data annotation is always a time and effort expensive procedure and a method for avoiding this step or minimizing the necessary effort must be discovered. A design that could provide great results and innovation would be the combination of supervised and unsupervised learning. The idea is that initially a procedure like clustering is used on the unlabeled data to create different clusters of products. Then, the initial link review becomes easier and the creation of trained models for new brands becomes possible.

Finally, the last remark concerns the combination of text and image classification for counterfeit detection. Until now, the image labels were generated with the use of Amazon Rekognition which is a separate service that only returns a set of tags for an image. According to the related work section in Chapter 3, images are the most valuable source of information in counterfeit detection and for that reason, an image classifier should be included in future implementations. Such a functionality would enable the extraction of more information from an image and increase the performance of counterfeit detection even more.
Appendix A

Project Planning

A.1 Project Timetable

<table>
<thead>
<tr>
<th>Task</th>
<th>Week</th>
<th>Duration (days)</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>1, 2</td>
<td>14</td>
<td>7/5/2018</td>
<td>20/5/2018</td>
</tr>
<tr>
<td>Data preparation</td>
<td>3</td>
<td>7</td>
<td>21/5/2018</td>
<td>27/5/2018</td>
</tr>
<tr>
<td>Model development</td>
<td>4, 5, 6, 7, 8</td>
<td>35</td>
<td>28/5/2018</td>
<td>1/7/2018</td>
</tr>
<tr>
<td>Model testing</td>
<td>9, 10</td>
<td>14</td>
<td>2/7/2018</td>
<td>15/7/2018</td>
</tr>
<tr>
<td>Report write up</td>
<td>11, 12, 13</td>
<td>21</td>
<td>16/7/2018</td>
<td>5/8/2018</td>
</tr>
<tr>
<td>Draft submission</td>
<td>14</td>
<td>1</td>
<td>6/8/2018</td>
<td>6/8/2018</td>
</tr>
<tr>
<td>Poster creation</td>
<td>14</td>
<td>6</td>
<td>7/8/2018</td>
<td>12/8/2018</td>
</tr>
<tr>
<td>Feedback receival</td>
<td>15</td>
<td>1</td>
<td>13/8/2018</td>
<td>13/8/2018</td>
</tr>
<tr>
<td>Report adjustment</td>
<td>15</td>
<td>2</td>
<td>14/8/2018</td>
<td>15/8/2018</td>
</tr>
<tr>
<td>Thesis and poster submission</td>
<td>15</td>
<td>1</td>
<td>16/8/2018</td>
<td>16/8/2018</td>
</tr>
<tr>
<td>Poster session</td>
<td>16</td>
<td>1</td>
<td>23/8/2018</td>
<td>23/8/2018</td>
</tr>
</tbody>
</table>
## A.2 Risk Management Plan

<table>
<thead>
<tr>
<th>Risk</th>
<th>Likelihood</th>
<th>Impact to project</th>
<th>Mitigation Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>False model architecture</td>
<td>Medium</td>
<td>Medium</td>
<td>Model modification and performance checking</td>
</tr>
<tr>
<td>Training has not been completed in time</td>
<td>Medium</td>
<td>High</td>
<td>Timetable modification and continuation of the training along with the implementation</td>
</tr>
<tr>
<td>Appearance of extreme test results</td>
<td>Medium</td>
<td>Medium</td>
<td>Testing evaluation and explanation of results (architecture choices, data properties)</td>
</tr>
<tr>
<td>Unavailable / unbalanced data</td>
<td>Medium</td>
<td>High</td>
<td>Data generation or annotation of unlabeled data</td>
</tr>
</tbody>
</table>
A.3 Gantt chart

1. Practical Part
   1.1 Python training
   1.2 AllenNLP training
   1.3 Data Extraction
   1.4 Data Preparation
   1.5 Link reader development
   1.6 Model development
   1.7 Ablation tests
   1.8 Pipeline testing

2. Report & Poster
   2.1 Report writing
   2.2 Poster creation
   2.3 Thesis and poster submission
Appendix B

Code Appendix

B.1 Classifier

```python
@Model.register("classifier")
class Classifier(Model):

    def __init__(self,
        vocab: Vocabulary,
        text_field_embedder: TextFieldEmbedder,
        platform_encoder: Seq2VecEncoder,
        search_terms_encoder: Seq2VecEncoder,
        description_encoder: Seq2VecEncoder,
        common_labels_encoder: Seq2VecEncoder,
        seller_encoder: Seq2VecEncoder,
        classifier_feedforward: FeedForward,
        initializer: InitializerApplicator = InitializerApplicator(),
        regularizer: Optional[RegularizerApplicator] = None) -> None:
        super(Classifier, self).__init__(vocab, regularizer)
        self.f1_measure = F1Measure(positive_label=1)
        self.text_field_embedder = text_field_embedder
        self.num_classes = self.vocab.get_vocab_size("labels")
        self.platform_encoder = platform_encoder
        self.search_terms_encoder = search_terms_encoder
        self.description_encoder = description_encoder
        self.common_labels_encoder = common_labels_encoder
        self.seller_encoder = seller_encoder
```

44
self.classifier_feedforward = classifier_feedforward
self.metrics = {"accuracy": CategoricalAccuracy(), "f1": self.f1_measure}
self.loss = torch.nn.CrossEntropyLoss()
initializer(self)

""
Starts the operation of the neural network.
""

@overrides
def forward(self, 
    platform: Dict[str, torch.LongTensor],
    search_terms: Dict[str, torch.LongTensor],
    description: Dict[str, torch.LongTensor],
    sigmoid_price: torch.FloatTensor, 
    seller: Dict[str, torch.LongTensor],
    common_labels: Dict[str, torch.LongTensor],
    label: torch.LongTensor = None) -> Dict[str, torch.Tensor]:

    embedded_platform = self.text_field_embedder(platform)
    platform_mask = util.get_text_field_mask(platform)
    encoded_platform = self.platform_encoder(embedded_platform, platform_mask)

    embedded_search_terms = self.text_field_embedder(search_terms)
    search_terms_mask = util.get_text_field_mask(search_terms)
    encoded_search_terms = self.search_terms_encoder(embedded_search_terms, search_terms_mask)

    embedded_description = self.text_field_embedder(description)
    description_mask = util.get_text_field_mask(description)
    encoded_description = self.description_encoder(embedded_description, description_mask)

    embedded_seller = self.text_field_embedder(seller)
    seller_mask = util.get_text_field_mask(seller)
    encoded_seller = self.seller_encoder(embedded_seller, seller_mask)

    embedded_common_labels = self.text_field_embedder(common_labels)
    common_labels_mask = util.get_text_field_mask(common_labels)
    encoded_common_labels = self.common_labels_encoder(embedded_common_labels, common_labels_mask)

    logits = self.classifier_feedforward(
torch.cat(
    [encoded_common_labels, encoded_platform, encoded_search_terms, encoded_seller, 
     encoded_description, sigmoid_price],
    dim=-1))

class_probabilities = f.softmax(logits, dim=1)

output_dict = {"class_probabilities": class_probabilities}

if label is not None:
    loss = self.loss(logits, label.squeeze(-1))
    for metric in self.metrics.values():
        metric(logits, label.squeeze(-1))
    output_dict["loss"] = loss

return output_dict

""
With this method, the model tells the training code which metrics it's computing.
""

@overrides
def get_metrics(self, reset: bool = False) -> Dict[str, float]:
    metrics = {}
    for metric_name, metric in self.metrics.items():
        if metric_name == "f1":
            precision, recall, f1_measure = metric.get_metric(reset)
            metrics.update({"precision": precision, "recall": recall, "f1": f1_measure})
        else:
            metrics.update({metric_name: metric.get_metric(reset)})

    return metrics

""
Takes the output of forward and does any necessary inference or decoding on it,
and it converts integers into strings to make things human-readable.
```python
@overrides
def decode(self, output_dict: Dict[str, torch.Tensor]) -> Dict[str, torch.Tensor]:
    predictions = output_dict['class_probabilities'].cpu().data.numpy()
    argmax_indices = numpy.argmax(predictions, axis=-1)
    print(argmax_indices)
    labels = [self.vocab.get_token_from_index(x, namespace="labels")
              for x in argmax_indices]
    output_dict['label'] = labels
    return output_dict

@classmethod
def from_params(cls, vocab: Vocabulary, params: Params) -> Classifier:
    embedder_params = params.pop("text_field_embedder")
    text_field_embedder = TextFieldEmbedder.from_params(vocab, embedder_params)
    platform_encoder = Seq2VecEncoder.from_params(params.pop("platform_encoder"))
    search_terms_encoder = Seq2VecEncoder.from_params(params.pop("search_terms_encoder"))
    description_encoder = Seq2VecEncoder.from_params(params.pop("description_encoder"))
    common_labels_encoder = Seq2VecEncoder.from_params(params.pop("common_labels_encoder"))
    seller_encoder = Seq2VecEncoder.from_params(params.pop("seller_encoder"))
    classifier_feedforward = FeedForward.from_params(params.pop("classifier_feedforward"))
    initializer = InitializerApplicator.from_params(params.pop("initializer", {}))
    regularizer = RegularizerApplicator.from_params(params.pop("regularizer", {}))

    return cls(vocab=vocab, 
                text_field_embedder=text_field_embedder, 
                platform_encoder=platform_encoder, 
                search_terms_encoder=search_terms_encoder, 
                description_encoder=description_encoder, 
                common_labels_encoder=common_labels_encoder, 
                seller_encoder=seller_encoder, 
                classifier_feedforward=classifier_feedforward, 
                initializer=initializer, 
                regularizer=regularizer)
```
B.2 Link Reader

```python
This class deals with the data manipulation and gets the input in an appropriate form for the deep learning model.
```

```python
@DatasetReader.register("link_reader")
class LinkReader(DatasetReader):
    
    Determines the tokenization behaviour or the word representations with the use of dependencies as inputs.
    
    def __init__(self,
        tokenizer: Tokenizer = None,
        token_indexers: Dict[str, TokenIndexer] = None,
        lazy: bool = False) -> None:
        super().__init__(lazy)
        self._tokenizer = tokenizer or WordTokenizer()
        self._token_indexers = token_indexers or {"tokens": SingleIdTokenIndexer()}

    Handles fields with none value inside the JSON file.
    
    @staticmethod
    def none_to_empty(value):
        return value if value is not None else ""

    Unifies the string of an array and returns them as a sentence.
    
    @staticmethod
    def concatenate_strings(string_array):
        return " ".join(string_array)

    Receives the image tags from a brand and a link and returns the common links, sorted alphabetically for easier testing.
    
```
```python
@staticmethod
def get_common_labels(first, second):
    return " ".join(sorted(list(set(first).intersection(second))))

""" Pull labels out of the supplied image list and combines into a single array
"""
@staticmethod
def get_image_labels(images):
    labels = []
    for image in images:
        if image is not None:
            for label in image["labels"]:
                labels.append(label["description"])  
    return labels

"""

Takes a file path or even a url of a JSON as an input and while iterating through the lines of the file, it creates instance with certain field formation.

"""
@overrides
def _read(self, file_path: str):
    with open(cached_path(file_path), "r") as file:
        image_labels = []
        for line in file.readlines():
            link_json = json.loads(line)
            for brand in link_json["brands"]: 
                image_labels.append(self.get_image_labels(brand["assets"]))
            for platform in link_json["platforms"]: 
                link_labels = self.get_image_labels(link["images"]) 
                yield self.text_to_instance( platform["name"], 
                                         self.concatenate_strings(self.none_to_empty(link["quotesingle.ts1"]), 
                                         self.get_common_labels(link_labels, image_labels), 
                                         self.search_terms))
```
def tokenize(self, thing):
    return TextField(self._tokenizer.tokenize(thing), self._token_indexers)

@overrides
def text_to_instance(self, platform: str, search_terms: str, common_labels: str, description: str, sigmoid_price: float, seller: str, class_field: str = None) -> Instance:
    fields = {
        "seller": self.tokenize(seller),
        "platform": self.tokenize(platform),
        "description": self.tokenize(description),
        "search_terms": self.tokenize(search_terms),
        "common_labels": self.tokenize(common_labels),
        "sigmoid_price": NumericField(sigmoid_price),
    }

    if class_field is not None:
        fields["label"] = LabelField(class_field)

    return Instance(fields)

"""
Receives a JSON dictionary and constructs the dataset reader.
The tokenizer and token indexer are initialized internally.
"""
@classmethod
def from_params(cls, params: Params) -> LinkReader:
    pass

file.close()
tokenizer = Tokenizer.from_params(params.pop('tokenizer', {}))
token_indexers = TokenIndexer.dict_from_params(params.pop('token_indexers', {}))
params.assert_empty(cls.__name__)
return cls(tokenizer=tokenizer, token_indexers=token_indexers, lazy=False)

B.3 Numeric Field

class NumericField(Field):

    def __init__(self, number: float) -> None:
        self.number = number

    @overrides
def get_padding_lengths(self) -> Dict[str, int]:
        # pylint: disable=no-self-use
        return {}

    @overrides
def as_tensor(self, padding_lengths: Dict[str, int], cuda_device: int = -1) -> torch.Tensor:
        # pylint: disable=unused-argument
        tensor = torch.FloatTensor([self.number])
        return tensor if cuda_device == -1 else tensor.cuda(cuda_device)

    @overrides
def empty_field(self):
        return NumericField(0.0)

    def __str__(self) -> str:
        return f"NumericField with value: {self.number}."

B.4 Predictor

@Predictor.register('rel-classifier')
class RelevancePredictor(Predictor):
Appendix B. Code Appendix

---

### Appendix B. Code Appendix 52

```python
class PredictorWrapper:
    @staticmethod
    def use_predictor_on_dataset(archive_path: str, predictor_name: str, file_path: str):
        archive = load_archive(archive_path)
```

---

B.5 Predictor Wrapper

---

Applies a defined predictor on multiples entries of a brand dictionary.

---

class PredictorWrapper:

@staticmethod
def use_predictor_on_dataset(archive_path: str, predictor_name: str, file_path: str):
    archive = load_archive(archive_path)
```
predictor = Predictor.from_archive(archive, predictor_name)
dict_gen = DictGenerator()
dict_gen.read_json(file_path)
json_dict = dict_gen.return_json_dict()

results = []

for link in json_dict['links']:
    results.append(predictor.predict_json(link))

return results

B.6 Configuration file

{
    "dataset_reader": {
        "type": "link_reader"
    },
    "train_data_path": "https://s3-eu-west-1.amazonaws.com/snapdragon-prod/1-of-3-trtl-relevance-2-Trtl.json",
    "validation_data_path": "https://s3-eu-west-1.amazonaws.com/snapdragon-prod/2-of-3-trtl-relevance-2-Trtl.json",
    "model": {
        "type": "classifier",
        "text_field_embedder": {
            "tokens": {
                "type": "embedding",
                "pretrained_file": "https://s3-us-west-2.amazonaws.com/allennlp/datasets/glove/glove.6B.50d.txt.gz",
                "embedding_dim": 50,
                "trainable": true
            }
        },
        "platform_encoder": {
            "type": "boe",
            "embedding_dim": 50
        },
        "search_terms_encoder": {
            "type": "lstm",
            "bidirectional": true,
            "input_size": 50
        }
    }
}
"hidden_size": 50,
"num_layers": 1,
"dropout": 0.2
},
"description_encoder": {
  "type": "lstm",
  "bidirectional": true,
  "input_size": 50,
  "hidden_size": 50,
  "num_layers": 1,
  "dropout": 0.2
},
"common_labels_encoder": {
  "type": "lstm",
  "bidirectional": true,
  "input_size": 50,
  "hidden_size": 50,
  "num_layers": 1,
  "dropout": 0.2
},
"seller_encoder": {
  "type": "boe",
  "embedding_dim": 50
},
"classifier_feedforward": {
  "input_dim": 401,
  "num_layers": 2,
  "hidden_dims": [
    150,
    2
  ],
  "activations": [
    "relu",
    "linear"
  ],
  "dropout": [
    0.2,
    0.0
  ]}
B.7 Dictionary Generator

class DictGenerator:
    def __init__(self):
        self.json_dict = {"links": []}

    def read_json(self, file_path: str):
        image_labels = []
        with open(cached_path(file_path), "r") as file:
            lines = file.readlines()
for line in lines:
    link_json = json.loads(line)
    brands = link_json['brands']
    for brand in brands:
        assets = brand['assets']
        for asset in assets:
            for label in asset['labels']:
                image_labels.append(label['description'])

platforms = link_json['platforms']
for platform in platforms:
    for link in platform['links']:
        link_labels = []
        search_terms = LinkReader.concatenate_strings(LinkReader.none_to_empty(link['search_terms']))
        images = link['images']

for image in images:
    if image is None:
        continue

    for label in image['labels']:
        link_labels.append(label['description'])

self.json_dict['links'].append(
    { 'platform': LinkReader.none_to_empty(platform['name']),
      'search_terms': LinkReader.none_to_empty(search_term),
      'description': LinkReader.none_to_empty(link['description']),
      'sigmoid_price': LinkReader.none_to_empty(float(link['sigmoid_price'])),
      'seller': LinkReader.none_to_empty(link['seller']),
      'common_labels': LinkReader.none_to_empty(LinkReader.get_common_labels(link_labels, image_labels))
    })

@staticmethod
def create_json_dict(json_dict, path):
    with open(path, 'w') as file:
        file.write(json.dumps(json_dict))
def return_json_dict(self):
    return self.json_dict

B.8 Overlap Detector

class OverlapDetector:
    
    Calculates the overlap of the basic fields for two brand datasets.
    
    def calculate_overlap(self, first_json_path: str, second_json_path: str):
        
        # extracts values from json files
        search_terms_train, descriptions_train, price_categories_train, sellers_train = 
            self.extract_link_values(first_json_path)
        search_terms_val, descriptions_val, price_categories_val, sellers_val = 
            self.extract_link_values(second_json_path)

        # Here we calculate the overlap of a field between the training and the testing set
        search_terms_val, common_search_terms = self.find_overlap_between_fields(search_terms_train, search_terms_val)
        descriptions_val, common_descriptions = self.find_overlap_between_fields(descriptions_train, descriptions_val)
        price_categories_val, common_price_categories = self.find_overlap_between_fields(price_categories_train, price_categories_val)
        sellers_val, common_sellers = self.find_overlap_between_fields(sellers_train, sellers_val)

        print("Search term overlap: ", len(common_search_terms) / len(search_terms_val) * 100)
        print("Descriptions overlap: ", len(common_descriptions) / len(descriptions_val) * 100)
        print("Price categories overlap: ", len(common_price_categories) / len(price_categories_val) * 100)
        print("Sellers overlap: ", len(common_sellers) / len(sellers_val) * 100)

    
    @staticmethod
    def find_overlap_between_fields(training_field, validation_field):
        validation_field, common_terms = validation_field[:], [e for e in training_field if 
            e in validation_field and (validation_field.pop(validation_field.index(e)))

        return common_terms
def extract_link_values(file_path):
    search_terms = list()
    descriptions = list()
    price_categories = list()
    sellers = list()
    counter = 0
    relevant_counter = 0
    irrelevant_counter = 0
    with open(file_path, "r") as file:
        lines = file.readlines()
        for line in lines:
            link_json = json.loads(line)
            brands = link_json["brands"]
            platforms = brands["platforms"]
            for platform in platforms:
                for query in platform["search_terms"]:  
                    search_terms.append(query["search_term"])
                    counter = counter + len(query["links"])
                    for link in query["links"]:  
                        if LinkReader.none_to_empty(link["relevant"]):
                            relevant_counter = relevant_counter + 1
                        else:
                            irrelevant_counter = irrelevant_counter + 1
                        descriptions.append(LinkReader.none_to_empty(link["description"]))
                        price_categories.append(
                            LinkReader.categorize_price(float(LinkReader.none_to_empty(link["converted_price"]))))
            print("Link size: ", counter, "Relevant links: ", relevant_counter,
                  "Irrelevant links: ", irrelevant_counter)
    return search_terms, descriptions, price_categories, sellers
### Calculates the matches of each review status

```python
@staticmethod
def view_review_status(file_path):
    with open(file_path, "r") as file:
        innocent_counter = 0
        counterfeit_counter = 0
        irrelevant_counter = 0
        lines = file.readlines()
        for line in lines:
            link_json = json.loads(line)
            brands = link_json['brands']
            platforms = brands['platforms']
            for platform in platforms:
                for query in platform['search_terms']:
                    for link in query['links']:
                        if LinkReader.none_to_empty(link['review_status']) == 'innocent':
                            innocent_counter = innocent_counter + 1
                        elif LinkReader.none_to_empty(link['review_status']) == 'irrelevant':
                            irrelevant_counter = irrelevant_counter + 1
                        else:
                            counterfeit_counter = counterfeit_counter + 1
            print("Innocent links: ", innocent_counter, "Irrelevant links: ", irrelevant_counter, "Counterfeit links: ", counterfeit_counter)
```

### B.9 Train.sh

```
rm -rf /tmp/output
allennlp train config_files/relevance_classifier.json -s /tmp/output --include-package swoop_ai
```

### B.10 Evaluate.sh

```
allennlp evaluate /tmp/output/model.tar.gz --evaluation-data-file https://s3-eu-west-1.amazonaws.com/snapdragon-prod/2-of-3-trtl-relevance-2-Trtl.json --include-package swoop_ai
```
Appendix C

Dataset Structure
Appendix C. Dataset Structure

Brands

Trtl

Assets

Labels

Score EntityID Description

Entities

Score EntityID Description

Platforms

Name Currency Links

Glencairn

Search terms Seller Description Sigmoid price images class

Score Description
Appendix D

MACS Risk Assessment Form (Project)

Student: Konstantinos Gavrilidis

Project Title: Counterfeit Product Detection with Deep Learning

Supervisor: Yannis Konstas

Risks:

<table>
<thead>
<tr>
<th>Risk</th>
<th>Present</th>
<th>Control Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Office environment- includes purely software projects</td>
<td>✔ Code loss</td>
<td>Use of VCS</td>
</tr>
<tr>
<td>Unusual peripherals e.g. Robot, VR helmet, haptic device, etc.</td>
<td>✗</td>
<td>Nothing</td>
</tr>
<tr>
<td>Unusual Output e.g. Laser, loud noises, flashing lights etc.</td>
<td>✗</td>
<td>Nothing</td>
</tr>
<tr>
<td>Other risks</td>
<td>✗</td>
<td>Nothing</td>
</tr>
</tbody>
</table>
Bibliography


