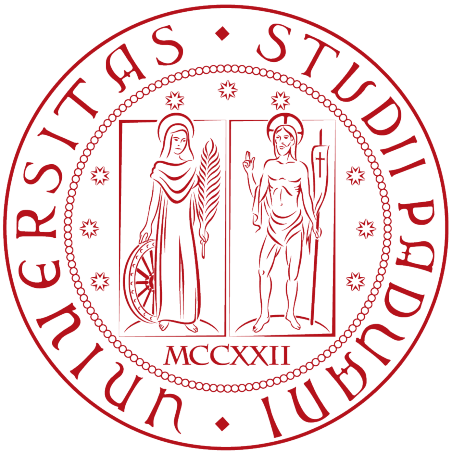
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Text Classification based on Contextualized Word Embeddings: an application of Weakly Supervised algorithms

Master Thesis

Advisors

Prof. Giorgio Satta

Prof. Giovanni Da San Martino

Student

Leonardo Pratesi

Academic Year 2020-2021

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine ai Professori Giorgio Satta e Giovanni Da San Martino, relatori della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute.

Padova, September 2021 Leonardo Pratesi

Contents

List of Figures

Chapter 1

Introduction

The study conducted in this thesis is the result of a collaboration with the startup Eutopia offered by the Erasmus for Traineeship program. Eutopia is a green startup located in Lisbon with the aim of building the biggest green startups database. The company mission is stated as follows:

> "Eutopia was born to increase transparency within the green innovation ecosystem and to provide decision-makers with a structured information framework to accelerate the transition toward a more sustainable economy. We do that by mapping European startups and innovative companies devoted to developing products, services or business models addressing the climate crisis."

The company leverages web scraping techniques to find new companies on the web and aggregates all the data into a unique database. This database is then accessible to anyone interested through an online portal under a paid plan subscription. The data acquired from the web is what is called unstructured data and requires some processing before being ingested into the database. To ease the search of companies through the portal the user needs a set of predefined categories/tags in a way that makes it possible to filter out the companies. Eutopia claims to have data on more than 10,000 company profiles so it is necessary to develop an efficient way of automatically labeling the data from each company discovered. As each company's data offers a brief description of the company activities the employer gave me the task of building a Text Classification model to reduce the amount of manual work required to tag each

company. The role within the startup was Data Scientist focused on NLP techniques to build an Automatic Text Classification system.
Text classification is a machine learning technique that assigns a set of predefined categories to open-ended text. These classifiers can be used to organize, categorize any kind of text. It is one of the fundamental tasks in natural language processing with broad applications such as sentiment analysis, topic labeling and spam detection. Text classification is also a building block in more elaborate natural language tasks. It's estimated that around 80% of all information is unstructured, with text being one of the most common types of unstructured data [Rog19]. Using text classifiers, companies can automatically categorize relevant text in a fast and cost-effective way. Starting from the initial task related to the training of a supervised text classification model developed during the company's internship, I analyse the possible options that can be undertaken when dealing with a problem where a document dataset with the correct classification is not available. In particular, I analyse two Weakly Supervised algorithms and propose some structural modifications to improve performance on business data.

The aim of this thesis is to provide a broad overview of the current state-of-the-art in text classification research by showing the methodologies, techniques and applications carried out in recent years and to present a concrete implementation of two different systems. Starting with the ConWea [MS20], innovations and limitations are analysed then a second algorithm with very similar characteristics LOTClass is compared [Men+20] and results inspected. Minor tweaks are applied to both algorithm to see how the performance is affected on the company dataset. In the final chapter a new algorithm is proposed that combines the initial available information of both algorithms and final performances are assessed.

1.1 Types of Machine Learning Approaches

Data scientists all over the world make use of a variety of Machine Learning algorithms to identify patterns within their data that provide them with interesting insights to support strategic decisions of various kinds. Assuming a high level of abstraction, such algorithms can be classified into two groups, based on their way of learning from data and consequently carrying out predictions. Supervised learning is certainly the best-known and most used approach. It includes algorithms such as logistic regression,

classification, and Support Vector Machines. The name of this class of algorithms comes from the fact that the data scientist will act as a "guide", or rather as a "supervisor" to teach the algorithm how to make certain decisions. These techniques strictly require that the classes assignable to the inputs are known "a priori" and that the input data used for training is already provided with the correct label. E.g., if we want to make a classifier that can identify if a certain document is talking about politics or not, it is necessary to be provided with a series of documents which will each be associated with a label, indicating whether or not the topic of the text is about politics. The dual approach, unsupervised learning, is closer to what today is considered true artificial intelligence, or the idea that a computer can learn to identify processes and patterns of different levels of complexity without any human support, as was needed in the previous case. Although unsupervised learning is notoriously more complex even for the simplest of applications, opens the way for the solution of problems that humans usually don't face. Some algorithms that fall into this category are K-Means Clustering, the Principal Component Analysis (PCA) and Latent Dirichlet Allocation [DJ03] .
Weak supervision is a branch of machine learning in which noisy, limited or imprecise sources are used to provide a supervisory signal for labeling large amounts of training data in a supervised learning context. This approach alleviates the burden of obtaining hand-labeled datasets, which can be expensive or impractical. Instead, inexpensive weak labels are employed with the understanding that they are imperfect, but can still be used to create a strong predictive model. Typically, there are three types of weak supervision. The first is incomplete supervision where only a (usually small) subset of training data is given labels, while the other data remains unlabelled. Such situation occurs in various tasks. For example, in image classification, the gold labels are given by human annotators; it is possible to extract a huge number of images from the Internet, while only a small subset of images can be manually annotated. The second type is imprecise monitoring, where only coarse-grained labels are given. Consider the image categorization task again. It is desirable to have every object in the images annotated; but, most of the time we only have image-level labels instead. The third type is inaccurate supervision, i.e. the labels given are not always the ground truth.

Chapter 2

Supervised Text Classification

In this chapter we introduce the text classification problem, we present some classical supervised algorithms used to tackle the task

2.1 Problem Statement

Text Classification is defined as assigning a discrete label $y \in Y$ to a textual document, where Y is the set of all the possible labels. The first step of classifying a document is deciding the text representation. A common approach is to represent the document as a vector of word counts. The individual weighted counts can be used by a linear classifier in a process of decision making. This representation is called "bag of words", the information stored keeps track only of the word count and ignores the word order. With bag of words grammar, sentence boundaries, paragraphs are ignored. [Jac18].

To predict a class from a bag of words we can assign a score to each word in the vocabulary, representing the compatibility with the label. These scores are called weights and are arranged in a column vector θ. In order to predict a class the score $\Psi(\boldsymbol{x}, y)$ is computed resulting in the measure of compatibility between the bag of words $\boldsymbol{x}$ and the label y. In a linear bag of words classifier, this score is the inner product

between the weights $\boldsymbol{\theta}$ and a so-called feature function.

$$\Psi(\boldsymbol{x}, y) = \boldsymbol{\theta} \cdot f(\boldsymbol{x}, y) = \sum_j \theta_j f_j(\boldsymbol{x}, y) \tag{2.1}$$

The most significant label can be chosen as the label that maximises the score function.

$$\hat{y} = \underset{y \in Y}{\operatorname{argmax}}\, \Psi(\boldsymbol{x}, y) \tag{2.2}$$

One could manually select the most significant words for each label and set the weights but, due to the large number of words and the non-trivial weight selection in practice the data is used to set the weights accordingly. Using already labeled data the weights can be acquired using supervised machine learning.

2.2 Probabilistic classifiers

Probabilistic classifiers make use of statistical models and use the features extracted from texts to identify a correct classification. In the training phase, the parameters of the probability distributions of the classes and features are stored. The classification is then carried out by means of evaluating the probabilities of the features extracted in the different classes.

2.2.1 Naive Bayes

The Naive Bayes classifier is the most used probabilistic classifier in literature. This model calculates the "a posteriori" probability with which a certain feature belongs to a particular class (or label) on the basis of the distribution of words in the document, the use of word features, and the Bayes theorem.

$$\mathrm{p}(y|\boldsymbol{x}) = \frac{\mathrm{p}(\boldsymbol{x}|y)\mathrm{p}(y)}{\mathrm{p}(\boldsymbol{x})} \tag{2.3}$$

Equation 2.3 is the Bayes Theorem. The variable y is the target class variable and $\boldsymbol{x}$ represent the parameters/features in this case the bag of words of a document. Here, $\boldsymbol{x}$ is the evidence and y is the hypothesis. To assign a class y to a bag of words $\boldsymbol{x}$ we

choose the y that maximizes $\mathrm{p}(y|x)$:

$$\hat{y} = \underset{y}{\operatorname{argmax}}\, \mathrm{p}(y|\boldsymbol{x}) = \underset{y}{\operatorname{argmax}}\, \frac{\mathrm{p}(\boldsymbol{x}|y)\mathrm{p}(y)}{\mathrm{p}(\boldsymbol{x})} \tag{2.4}$$

For $\boldsymbol{x}$ fixed, the $\mathrm{p}(y|\boldsymbol{x})$ for various values of y, does not depend on $\mathrm{p}(\boldsymbol{x})$. For this reason the maximum value of y can be calculated without counting the denominator $\mathrm{p}(\boldsymbol{x})$ obtaining the same result. Inferring $\hat{y}$ becomes then equal to maximizing a joint probability $\mathrm{p}(y, \boldsymbol{x})$ and for this reason Naive Bayes is considered a generative model.

$$\hat{y} = \underset{y}{\operatorname{argmax}}\, \mathrm{p}(y|\boldsymbol{x}) = \underset{y}{\operatorname{argmax}}\, \mathrm{p}(\boldsymbol{x}, y) \tag{2.5}$$

$\boldsymbol{x}$ can then be expanded in the set of words $(x_1, x_2, ..., x_K)$. Since estimating $\mathrm{p}(x_1, x_2, ..., x_K|y)$ effectively requires a large amount of data, Naive Bayes approach assumes the x_i to be independent of each other; an assumption that is obviously not true in all real cases. If the independence of variables was not assumed, the expansion of the bag of words $\boldsymbol{x}$ into the singular components would require the calculation of all the conditional probabilities. Instead, by using the independence assumption, the equation becomes the following:

$$\begin{aligned} \hat{y} &= \underset{y}{\operatorname{argmax}}\, \mathrm{p}(x_1, x_2, ..., x_K|y)\mathrm{p}(y) \\ &= \underset{y}{\operatorname{argmax}} \prod_{j=1}^{K} \mathrm{p}(x_j|y)p(y) \end{aligned} \tag{2.6}$$

Given a dataset of N labeled instances $\{\boldsymbol{x}^{(i)}, y^{(i)}\}^N$ assumed independent and identically distributed (IID), we can calculate the joint probability of the dataset as $\prod_{i=1}^{N} \mathrm{p}_{X,Y}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})$. The vector of weights $\boldsymbol{\theta}$, introduced in 2.1 can be set so that it maximizes the joint probability of the labeled documents.

Given that that the dataset instances are independent and identically distributed, the Naive Bayes training can be traced back to the solution of the following equation:

$$\begin{aligned} \hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{N} \mathrm{p}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{N} \log \mathrm{p}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}) \end{aligned} \tag{2.7}$$

This equation is called maximum likelihood estimation. $\boldsymbol{\theta}$ becomes a parameter of the probability function that can be easily calculated by substituting the joint probability with the mathematical production showed in equation 2.6 . The product of probabilities can be converted as a sum of log-probabilities as the logarithm is a monotonically increasing function and has the advantage of numerical stability. The prediction rule for Naive Bayes is to choose the label $\hat{y}$ that maximizes $\log \mathrm{p}(\boldsymbol{x}, y; \boldsymbol{\theta})$. [Jac18]

The Naive Bayes algorithm is limited by the fact that relies only on the bag of words model. Sometimes a lot of information is encoded in prefixes/suffixes, word Capitalization and n-grams (multi words units). Naive Bayes assumes the tokens to be independent, but these kind of features violate this rule. For example Naive Bayes computes the joint probability of a word and its prefix with this approximation :

$$\Pr(\text{word} = \text{unfit}, \text{prefix} = \text{un} - |\text{y}) \approx \Pr(\text{prefix} = \text{un} - |\text{y}) \times \Pr(\text{word} = \text{unfit}|\text{y}) \quad (2.8)$$

By applying the chain rule we convert the equation in this form:

$$\begin{aligned} \Pr(\text{word} = \text{unfit}, \text{prefix} = \text{un} - |\text{y}) = \Pr(\text{prefix} = \text{un} - |\text{word} = \text{unfit}, \text{y}) \\ \times \Pr(\text{word} = \text{unfit}|\text{y}) \end{aligned} \quad (2.9)$$

The probability of the word unfix starting by un- is equal to 1 but as Naive Bayes assumes that every token is IID, it underestimates every probability of positively correlated features (every probability of a random word starting by "un-" is much less than 1).

Naive Bayes systematically underestimates the true probabilities of conjunctions of positively correlated features. A way of solving this problem is by applying an error-driven algorithm that does not make any assumption on the probabilities.

2.3 Linear Classifiers

Like the probabilistic classifier, Naive Bayes, described above, linear classifiers also aim to use feature vectors to identify the correct class of the text they represent. The classification is performed by testing if the dot product between the vector of weights $\boldsymbol{\theta}$ and the document feature representation $\mathrm{f}(\boldsymbol{x}, \hat{y})$ is above a certain threshold t and then

return the l top ones. This approach is the basis of different algorithms: Perceptron, Support Vector Machine and Logistic Regression.

2.3.1 Perceptron

The perceptron algorithm changes the weights for the features accordingly every time a mistake is made. Features correlated with the correct label $y^{(i)}$ are increased, the others are decreased. It is what is called an online learning algorithm. The weights change on each iteration, through a "trial and error" process, contrary to Naive Bayes where the weights are set only once. The training algorithm is the following:

Algorithm 1: Perceptron Online Algorithm

$t \leftarrow 0$

$\boldsymbol{\theta^{(0)}} \leftarrow \mathbf{0}$

repeat

  $t \leftarrow t + 1$

  Select an instance i

  $\hat{y} \leftarrow argmax_y \boldsymbol{\theta}^{(t-1)} \cdot f(x^{(i)}, y)$

  if $\hat{y} \neq y^{(i)}$ **then**

    $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta^{(t-1)}} + f(x^{(i)}, y^{(i)}) - f(x^{(i)}, \hat{y})$

  else

    $\boldsymbol{\theta^{(t)}} \leftarrow \boldsymbol{\theta}^{(t-1)}$

until *tired*;

return $\boldsymbol{\theta^{(t)}}$

The perceptron works by optimizing a loss function:

$$loss_{\text{PERCEPTRON}}(\boldsymbol{\theta}, \boldsymbol{x}^{(i)}, y^{(i)}) = \max_{y \in Y} \boldsymbol{\theta} \cdot f(\boldsymbol{x}^{(i)}, y) - \boldsymbol{\theta} \cdot f(\boldsymbol{x}^{(i)}, y^{(i)}) \tag{2.10}$$

At each instance the perceptron training algorithm moves in the opposite direction of the gradient of 2.10 with respect to the $\boldsymbol{\theta}$ that is equal to $f(x^{(i)}, y) - f(x^{(i)}, \hat{y})$. If the predicted label $\hat{y}$ is equal to the true label $y^{(i)}$ then the loss is zero and no change is applied to the inference function. Further away is the predicted label from the true one and the loss function will return a bigger value. This algorithm is called stochastic gradient descent and it is the standard optimization algorithm for training neural networks that will be introduced in chapter 2.4.1.

Optimization problems are formulated as the minimization of a loss function. The

loss function takes in input the $\boldsymbol{\theta}$ and returns a positive number giving a score to the classifier performance on a specific training instance. The goal is to minimize the sum of all the computed losses across every instance in the training set. If the loss function is defined to be the negative log likelihood, minimizing the loss function is equivalent to maximizing the maximum likelihood, as shown in equation 2.11.

$$
\begin{aligned}
&loss_{\text{NB}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = -\log \text{p}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}) \\
&\hat{\boldsymbol{\theta}} = \underset{\theta}{\text{argmin}} \sum_{i=1}^{N} loss_{NB}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = \underset{\theta}{\text{argmax}} \sum_{i=1}^{N} \log \text{p}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})
\end{aligned}
\tag{2.11}
$$

Some differences and similarities can be observed between the perceptron and Naive Bayes:

- Both optimization functions are convex. Convexity implies that any local minimum is also a global minimum, and for this reason the function is easy to optimize.
- Naive Bayes assumes the features to be conditionally independent given the label. Meanwhile the perceptron does not make any assumption.
- The perceptron loss treats all correct answers equal. Even if $\boldsymbol{\theta}$ give the correct answer by a small margin the loss calculated is zero.

Formally we can prove that the perceptron algorithms converge to a solution thanks to the concept of linear separability. If the data is linear separable it is guaranteed that the perceptron will find a hyperplane such that on each side all samples have the same label. Formally the concept of linear separability has the following definition:

Definition 2.3.1 (Linear separability)**.** The dataset $D = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$ is linearly separable if and only if there exists some weight vector $\boldsymbol{\theta}$ and some margin ρ such that for every instance $(\boldsymbol{x}^{(i)}, y^{(i)})$, the inner product of $\boldsymbol{\theta}$ and the feature function for the true label, $\boldsymbol{\theta} \cdot \text{f}(\boldsymbol{x}^{(i)}, y^{(i)})$, is at least ρ greater than inner product of and the feature function for every other possible label, $\boldsymbol{\theta} \cdot f(\boldsymbol{x}^{(i)}, y^{'})$.

It was proven by Misky and Papert that even simple logical functions like "exclusive-or" are not linearly separable: the model will jump between two weights settings, never converging [MS69]. This discovery hindered and limited the development of

neural networks but was subsequently overcome with the development of non-linear activation functions. In real-life text classification problems, very high dimensional features spaces are involved and it is very likely that the data is indeed separable. But if this requirement does not hold, by placing an upper bound on the number of errors a solution is still obtainable.
Another solution is to use the "Averaged perceptron". This version of the algorithms averages the perceptron weights across all iterations. Even if the data is not separable, the averaged weights will converge thanks to the non-linearity introduced by the average function.

However, converging the algorithm does not mean that the trained model will give useful predictions, in fact a learning algorithm is prone to overfitting on the training set. Overfitting can have several causes: The learning model takes into account in the training set attributes X that are irrelevant to the final decision y, the training data are insufficient, the examples are too few or the training data are not representative of reality. This event can be controlled by using a held-out dataset to keep track of the predictive accuracy and stop the training (early stopping) if it decreases. If is able to make good predictions on never before seen data we say that it "generalizes" well.

2.3.2 Support Vector Machine

The SVM classifiers belong to the supervised machine learning methods which, starting from a suitably labeled training set, construct a general model for classification. The training set represents a set of points in a plane. Each example is placed in the plane by exploiting the values of the feature vector that represents it. The SVM algorithm finds the best separation lines that define the largest hyperplanes able to represent, with the minimum error, the different labels of the classification. These lines are obtained by maximizing the distance of the nearest points of the different classes represented in the training set. We define as "Margin" the difference between the score for the highest scoring incorrect label and the correct label $y^{(i)}$. Maximizing the "Margin" allows the model to be more robust, unlike the perception predictions that are characterized by big a variance: very similar features can still be classified in different categories if the correct answer is given by a small margin. Mathematically is expressed like this:

$$\gamma(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = \boldsymbol{\theta} \cdot f(\boldsymbol{x}^{(i)}, y^{(i)}) - max_{y \neq y^{(i)}} \boldsymbol{\theta} \cdot f(\boldsymbol{x}^{(i)}, y) \tag{2.12}$$

Ideally it's not enough to train the data correctly but we want the margin to be as big as possible. This idea turns into a loss function:

$$loss_{\text{MARGIN}}(\boldsymbol{\theta}, \boldsymbol{x}^{(i)}, y^{(i)}) = (1 - \gamma(\boldsymbol{\theta}, \boldsymbol{x}^{(i)}, y^{(i)}))_+ \tag{2.13}$$

where $(x)_+$ is equal to $max(0, x)$. If the margin is at least 1 between the score for the correct label and the best scoring alternative $\hat{y}$ then the loss is zero. This is very similar to the perceptron loss but the hinge point is moved to the right.

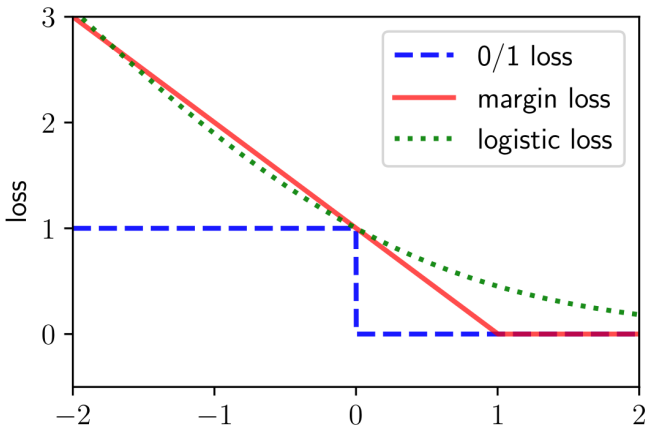


Figure 2.1: Margin, zero-one and logistic loss function.

This margin loss can be minimized by using an online learning rule called "online support vector machine". This learning rule differs from the perceptron in two key differences:

- Instead of selecting the label that maximizes the score of the model, the argmax searches for labels that are strong and wrong as measured respectively by $\boldsymbol{\theta}\text{f}(x^{(i)}, y^{(i)})$ and $\text{c}(y^{(i)}, y)$. If the highest-scoring label is $y = y^{(i)}$, then the margin loss for this instance is zero, and no update is needed. Cost augmentation is only done while learning; it is not applied when making predictions on unseen data.

- The previous weights $\boldsymbol{\theta}_{(t-1)}$ are scaled by $(1 - \lambda)$, with $\lambda \in (0, 1)$. This applies a form of regularization that cause the weights to "decay" back towards zero.

2.3.3 Logistic Regression

As was stated before Naive Bayes is a probabilistic method while perceptron and support vector machine are discriminative error-driven algorithms. Each category has its own advantages: probability allows quantifying the uncertainty about predicted labels but relies on unrealistic independence assumptions. Logistic Regression combines the advantages of both discriminative and probabilistic classifiers. Logistic regression defines a conditional probability $\mathrm{p}_{Y|X}$ instead of a joint probability $\mathrm{p}_{X,Y}$. To achieve this the result of the scoring function $\boldsymbol{\theta} \cdot \mathrm{f}\,(\boldsymbol{x}^{(i)}, y^{(i)})$ is exponentiated so that is guaranteed to be non-negative. The resulting probability is defined as:

$$\mathrm{p}(y|x;\boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta} \cdot f(\boldsymbol{x}^{(i)}, y^{(i)})}{\sum_{y' \in Y} \exp(\boldsymbol{\theta} \cdot f(\boldsymbol{x}^{(i)}, y^{(i)})} \tag{2.14}$$

Given a dataset the weights $\boldsymbol{\theta}$ are estimated through maximum conditional likelihood:

$$\begin{aligned} \log \mathrm{p}(y^{(1:N)}|\boldsymbol{x}^{(1:N)};\boldsymbol{\theta}) &= \sum_{i=1}^{N} \log \mathrm{p}(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}) \\ &= \sum_{i=1}^{N} \boldsymbol{\theta} \cdot \mathrm{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - log \sum_{y' \in Y} exp(\boldsymbol{\theta} \cdot \mathrm{f}(\boldsymbol{x}^{(i)}, y')) \end{aligned} \tag{2.15}$$

By taking the inverse of the second part of the equation we obtain the loss function for the logistic regression, called logistic loss.

$$loss_{\mathrm{LOGREG}}(\boldsymbol{\theta}, x^{(i)}, y^{(i)}) = -\boldsymbol{\theta} \cdot \mathrm{f}(\boldsymbol{x}^{(i)}, y^{(i)}) + log \sum_{y' \in Y} \exp(\boldsymbol{\theta} \cdot f(\boldsymbol{x}^{(i)}, y^{(i)})) \tag{2.16}$$

The key characteristic of this loss is that the logistic loss is never zero, for this reason, the objective function can be always improved. On the contrary, the zero-one loss/hinge loss that returns zero when the target class is predicted correctly.

2.4 Non-linear Classifiers

2.4.1 Neural Networks

As it was pointed out, the disadvantage of a single perceptron is that it is able to learn only linear separable problems. Therefore, for example, a perceptron is not able to approximate the XOR function. The solution to this problem is to build a multilayer feedforward network, which translates into connecting multiple layers of perceptrons. The connection between two neurons is defined by its weight, which is the attenuation or amplification of signals received from the input data. Processing of the input data is done layer by layer. Input data are presented to the neural networks input layer, then are propagated to hidden layer neurons and processed with an activation function as a weighted sum of inputs and related weights.

It is important that the activation function is non-linear, typically a sigmoidal function or hiperbolic tangent (tanh) function. Subsequent layers of the network function as the hidden layer always process the outputs from the previous layer. The output of the neural network is the output of the last layer. If the hidden layer is viewed as a set of latent features, then the sigmoid function represents the extent to which each of these features is "activated" by a given input. However, the hidden layer can be regarded more generally as a nonlinear transformation of the input. The backpropagation algorithm is used for training multilayer neural networks; it is based on the calculation of the error at the output of the neural network and the propagation error propagation through the neural network with the adjustment of weights. [Lac17]

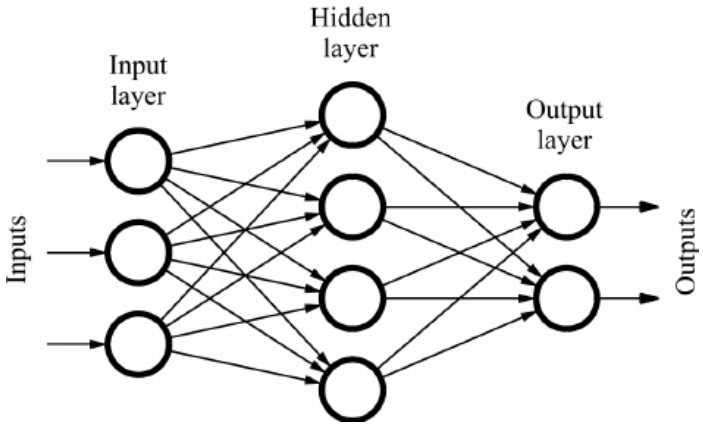


Figure 2.2: Structure Feedforward Neural Network.

Given a feature representation $\boldsymbol{x}$ of a document and a multiclass classification problem, a feedforward neural network can be set up so that it generates in output a distribution over the N classes. This is done by applying a softmax function on the output layer. A softmax output produces probabilities over each possible label.
By applying the softmax layer the training of the neural network becomes equivalent to minimize the conditional log-likelihood. This requires a dataset of documents where the correct label $y^{(i)}$ is known.

$$-L = -\sum_{i=1}^{N} \log \mathrm{p}(y^{(i)}|\boldsymbol{x};\boldsymbol{\theta}) \tag{2.17}$$

The softmax function is defined as follows:

$$\mathrm{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2.18}$$

It takes as input a vector z of K real numbers and normalizes it into a probability distribution consisting of N probabilities proportional to the exponentials of the input numbers. Before applying softmax, some vector components could be negative, or greater than one and might not sum to 1. After applying the softmax function, each component will be in the interval [0,1], and the components will add up to 1 so that they can be interpreted as probabilities.

2.4.2 Input Representation

To use text as input data, it must first be converted into an appropriate representation, which is usually a vector of the text's features. The vector representations of text can be constructed in many different ways. The basic methods of non-neural text representation are one-hot encoding and TF-IDF (term frequency, inverse document frequency). These representations limit the amount of information that can be used, for example, the order of words is not taken into account. One-hot encoding creates a Boolean vector of values for each word. The length of a one-hot vectors is equal to the size of the vocabulary, each element of the vector is encoded with a "0" except the word in focus, which is encoded with a "1". Each dimension in that vector space corresponds to one word. Vectors from that space are used as input for machine learning. To display larger text units (multi-word units such as phrases, sentences, or documents), one-hot vectors have a "1" for each of the words appearing in the

document (bag-of-words representation). TF-IDF vectors extend Boolean values of one-hot vectors with frequencies, normalized by the inverse document frequencies. More advanced neural models teach text representations as vectors in a continuous n-dimensional space. The learned n-dimensional vector space can be structured to include knowledge, semantics, syntax, or other language properties. The process of displaying text elements as continuous and dense vectors is called embedding (e.g. word embedding). The representations can be learned for each unit of text, such as subwords, words, phrases, sentences, and documents. The current challenge in neural learning text representation is to construct task-independent representations, hence representations that generalize well to multiple unrelated tasks. In the next chapter, we will introduce some recent input representations called Word Embeddings and the way they are used to classify textual documents.

Chapter 3

Word Embeddings

3.1 Introduction

Neural approaches have been explored to address the limitations of Bag of Words models. The most important component of these approaches is a machine learning embedded model that transforms text into low-dimensional continuous feature vectors. One important application of the Word Embeddings is their use in Language Models based on neural networks they provide the ability to represent words in various aspects, and neural networks are used to train these vectors using corpora.
A Language Model is a machine learning model that has been trained to predict the next word or words in a text based on the preceding words or neighobouring words and it is part of the technology that suggest the next word when writing text in modern messaging applications.

In 2001, Bengio et al. propose the first neural Language Model trained on 14 million words based on a feed-forward neural network. These early embedding models underperformed comparing to classical models using hand-crafted features, and thus were not widely adopted. A change of direction began when much larger embedding models started to be developed using huge amounts of training data. Google started building a series of word2vec models that were trained on 6 billion words and immediately became popular for many NLP tasks. In 2017, the University of Washington developed a contextual embedding model based on a bidirectional LSTM with 93M parameters trained on one billion words. The model, called ELMo, was an incredible improvement over word2vec because they were able to capture contextual information.

In 2018, OpenAI researchers developed embedding models using Transformer, a new NN architecture developed by Google. Transformer is an attention based system that substantially improves the efficiency of large-scale model training on TPU or GPU; it will be introduced in 3.7. Their first model is called GPT, which is now widely used for text generation tasks. The same year, Google developed BERT based on bidirectional transformer. BERT is made of 340M parameters, is trained on 3.3 billion words, and is the current state of the art in embedding models. The trend to use larger models and more training data is set to continue. By the time this paper is published, OpenAI's latest GPT-3 model contains 170 billion parameters, and Google's GShard contains 600 billion parameters. [She+20]

The basic idea of a Language Model is the following: words within a text that are in the same context are very likely to share some kind of meaning [Har54]. The probability of a sequence of words $w_1, w_2..., w_t - 1$ occurring can therefore be formulated by means of Bayes' rule:

$$\mathrm{P}[W] = \prod_{t=1}^{N} P[w_t|w_1, ..., w_{t-1}] = \prod_{t=1}^{N} \mathrm{P}[w_t|h_t] \tag{3.1}$$

Where $\mathrm{P}[W]$ is the joint distribution of the sequence W, and h_t represents the context words appearing before the word w_t. The objective is to evaluate the probability of the word w_t occurring given the information about its context. Most models of this type belong to the class of unsupervised models. Typically, the objective of the neural network-based Language Model is to maximize or minimize a certain cost function, which is very often a log-likelihood function. Suppose we have a feed-forward neural network and a sequence $w_1, w_2, ..., w_n$ of words in the corpus, we are interested in maximizing the log-likelihood in the neural network represented.

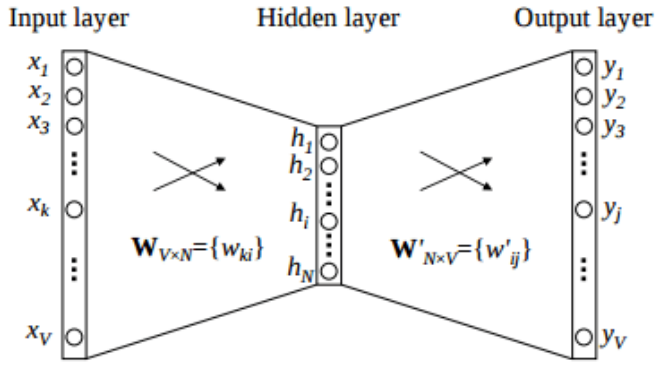


Figure 3.1: Feed fordward Neural Network for Word Embedding computation

3.2 Word2vec

An alternative model to the one just presented is the Word2Vec model proposed by Mikolov et al. in [Tom+13]. Two frameworks are presented: Skip-gram and CBOW (Continuous Bag-of-Words). The simplicity of the models and the use of various tricks to keep the computational costs low have made it the reference model in the field of Word Embeddings.

The aim of the Continuous Bag-of-Words and Skip-Gram models is to train Word Embeddings using neural networks. Such training is carried out trying to optimize an objective function that is nothing more than the calculation of the probability of a certain central word or of a set of context words within a window of 2D dimension.

The mock task that is used, in this case, to train the network, is the following: starting from a specific word in the middle of a sentence (called input word), randomly choosing a word "in its vicinity", the network will have to return a probability of one word in the middle of a sentence. Therefore, the probabilities that will be returned in the output, will refer to how much plausible to find each vocabulary word in the vicinity of our input word. For example, following training, giving the word "Soviet", the probability of output will be higher for words like "Union" and "Russia" than for unrelated words such as "pasta" or "koala".

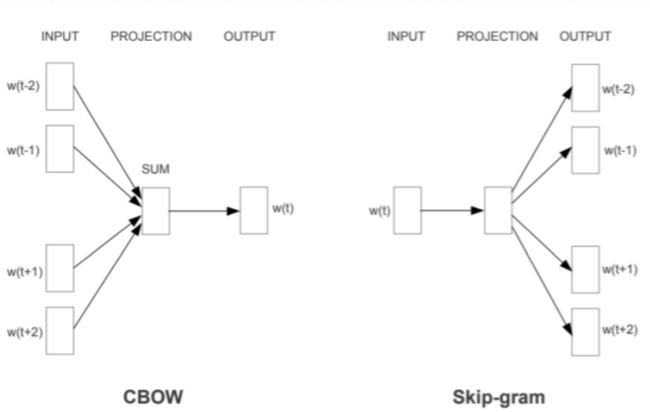


Figure 3.2: Skip Gram and CBOW models

3.2.1 CBOW

The Continuous Bag-of-Words model deals with the training of Word Embeddings through the optimization of an objective function that calculates the probability of the current central word given the Bag-of-Words of the context words.

The output is generated from the input as follows: given the word x at position t the value of the hidden layer h is calculated as:

$$h = \frac{1}{2D} W \cdot \sum_{-D<j<D, j\neq 0} x_{t-j} \tag{3.2}$$

An average of the input vectors weighted by the matrix W is then performed . The inputs for the calculation of the output layer are then:

$$u_i = (v'_{w_i})^T \cdot h \tag{3.3}$$

Where v'_{w_i} is the i-th column of the matrix W. The output y_i is then calculated using the Softmax function:

$$y_i = P(w_{y_i} | w_{t-D}, ...w_{t-1}, w_{t+1}, ..., w_{t+D}) = \frac{\exp(u_j)}{\sum_{i=1}^{V} \exp(u'_j)} \tag{3.4}$$

3.2.2 Skip-Gram

The Skip-Gram model deals with the training of Word Embeddings through the optimization of an objective function that calculates the probability of the context words

$w_t-D, ..., w_t-1, w_t+1, ..., w_t+D$ in a certain window of dimension $2D$ given the current central word w_t. The model input is the one-hot vector x relative to the current central word w_t while the output is the set of vectors $\{y_{t-D}, ..., y_{t-1}, y_{t+1}, ..., y_{t+D}\}$ related to the context words $\{w_{t-D}, ..., w_{t-1}, w_{t+1}, ..., w_{t+D}\}$.. The matrix W of dimension $V \times d$ is the matrix of the weights that connects the input layer to the hidden layer whose i-th row represents the weights related to the i-th word of the vocabulary (it is the matrix containing the vectors relative to the central words). In addition, there is a matrix W_0 of dimension $d \times V$ containing the weights related to the context words. In the hidden layer, a product is made between the one-hot vector of the current central word and the matrix of weights W. As the vector x is a one-hot matrix product returns the vector of weights for the central word, i.e. the k-th word.

$$h = x^T W = W_{k_r} = v_{w_t} \tag{3.5}$$

In the input layer, unlike the model seen in the previous section, the calculation of $2D$ distributions, one for each context word in the window, is carried out by applying the Softmax function. The input of the j-th node for the c-th word in the output is:

$$P(w_{c,j} = w_{O,c}|w_l) = y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V} \exp(u'_j)} \tag{3.6}$$

A calculation is made of the probability that the vector produced by the model is actually the vector relative to the c-th context word within the given the central input word.

3.2.3 Optimizations

The authors of the word2vec paper propose 2 solutions to improve the speed to the training process:

- Hierarchical Softmax: a possible solution for lowering the computational complexity of calculating the Softmax function on large dictionaries is to perform an approximation through the technique of the Hierarchical Softmax, This approach consists in evaluating the Softmax function on a logarithmic number of words $\log_2(V)$, instead of the whole dictionary V, through the use of a binary tree representation of the output layer with the V words as leaves and, as internal nodes, the relative probabilities of the children nodes. This type of representation

defines random paths that assign probabilities to the words.

- Negative sampling allows to only modify a small percentage of the weights, rather than all of them for each training sample. Instead of trying to predict the probability of being a nearby word for all the words in the vocabulary, we try to predict the probability that our training sample words are neighbors or not. The problem is further simplified by randomly selecting a small number of "negative" words k to update the weights.

3.3 FaxtText Embeddings

FastText is a free, lightweight, open source library that allows users to learn text representations and text classifiers. It introduces an upgrade over classic word2vec embeddings by adding sub-word information to the process. [Arm+17a]. In addition, they introduce some ideas on improving text classification models [Arm+17b]. Word2vec ignores word morphology, assigning a separate vector to each word. This limits the model capabilities, especially for languages with a large vocabulary and a lot of rare words. Fasttext uses the skip-gram model but also represents each word as a bag of n-grams of characters. A vector representation is associated with each n-gram character and the words are represented as the sum of these representations.
This method is fast, allowing us to quickly train models on large corpora and calculate word representations for word tokens that did not appear in the original training data.

Each w word is represented by a bag of n-gram characters. Special limit symbols <and> are added at the beginning and at the end of words, making it possible to distinguish prefixes and suffixes from other characters sequences. The original word w is included in its set of n-grams, to learn a representation for each word.

> Taking the word where and n = 3 as an example, it will be represented by the character n-grams: <wh, whe, her, ere, re> and the special sequence <where>.

The sequence <her>, corresponding to the word "her" is different from the tri-gram her from the word where. In practice, all the n-grams for n greater or equal to 3 and smaller or equal to 6 are extracted. This is a very simple approach, and different sets of n-grams could be considered, for example taking all prefixes and suffixes.

A vector representation $\boldsymbol{z_g}$ is associated to each n-gram g. A word is represented by the sum of the vector representations of its n-grams. Thus obtaining the scoring function where G_w is the set of n-grams appearing in w:

$$s(w, c) = \sum_{g \in G_w} \boldsymbol{z_g}^T \boldsymbol{v}_c \tag{3.7}$$

This model allows sharing the representations across words allowing to learn consistent representation for rare words. The word vectors have dimension 300. For each positive example, 5 negatives are sampled at random, with probability proportional to the square root of the uni-gram frequency. A context window of size c is used, and uniformly sample the size c between 1 and 5. In order to subsample the most frequent words, a rejection threshold of 10-4 is used. When building the word dictionary, the words that appear at least 5 times in the training set are kept.

FastText for Text Classification

Fasttext introduces a new text classification model based on the FastText embeddings that has performance comparable to the most recent Convolutional Neural networks but is much faster to train. Unlike unsupervised trained word vectors from word2vec, FastText word features can be averaged together to form good sentence representations. These representations can be fed into a multinomial logistic classifier. In several tasks, FastText obtains performance on par with recently text classification methods inspired by deep learning, while being much faster. The softmax function f is used to compute the probability distribution over the predefined classes. For a set of N documents, this leads to minimizing the negative log-likelihood over the classes:

$$-\frac{1}{N} \sum_{n=1}^{N} y_n \log(\mathrm{f}(BAx_n)) \tag{3.8}$$

Where the first weight matrix A is a look-up table over the words, x_n is the normalized bag of features of the n-th document, y_n the label, A and B the weight matrices. B is the matrix of the effectively learned parameters. Bag of words is invariant to word order but taking explicitly this order into account is often computationally very expensive. Instead, a bag of n-grams is used as additional features to capture some partial information about the local word order. This is very efficient in practice while achieving comparable results to methods that explicitly use the order. [Kil+09]

Although FastText has fast performance, the best prediction accuracy is obtainable by using the state-of-the-art Bert model that leverages word Embeddings, Recurrent Neural networks, and Attention mechanism to encode textual information. In the following chapters, we will introduce the main concepts that cooperate inside the BERT Language Model.

3.4 RNN

Recurrent Neural Network is a class of artificial neural networks used in prediction tasks, able to analyze Time Series and predict future trends: the price of shares, the trajectory of a vehicle, the next note in a melody, and much more. They were introduced in the late 1980s to process sequential input. [Jor09]
The well-known sequencing processing methods used are the Hidden Markov Model (HMM) and n-gram Language Model [Arm+17b] [L R86]. In working with traditional neural networks, the assumption has always been that all inputs (and outputs) are independent of each other. This is not true for many tasks. For instance, predicting a word in a sentence requires the knowledge of the words that came before it. RNNs work in a similar way, so the recurrent part of RNNs is a result of its ability to perform the same task for every element of a sequence with the output being dependent on the previous computations. RNN is able to save the history of previously processed elements so that at every given point in time, this stored history is used to predict the next output from the process. This is the RNN architecture:

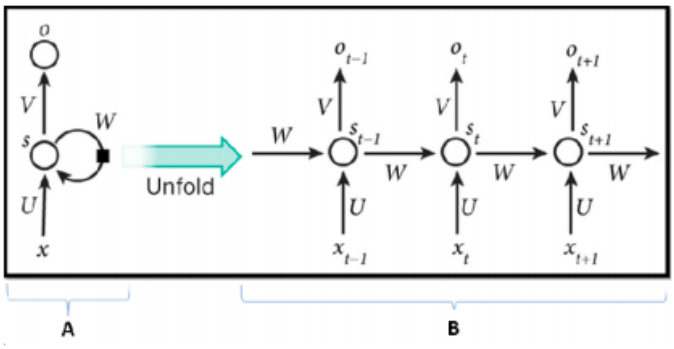


Figure 3.3: RNN architecture, folded A, unfolded B

As it has more than 1 hidden layer it can be considered as a deep learning neural network. As the unfolded architecture shows, there are 3 different matrices of weights:

U, V, and W. W represents the weights between the hidden state S. V connects the hidden state to the output O and U connects the input X to the hidden state S.

To update these weights the RNN relies on the Back Propagation Algorithm through time (BPTT). X_t is the input at time step t. Represent a given word in a sentence. S_t is the hidden state at time step t. It's the memory of the network and is calculated by combining the previous hidden state $(S_t - 1)$ and the input at the current state.

$$St = \mathrm{f}\,(UX_t + WS_{t-1}). \tag{3.9}$$

The function f is a non-linear activation function such as tanh or ReLU, which is required to calculate the first hidden state. O_t is the output at step t. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary.

$$O_t = \mathrm{Softmax}(VS_t) \tag{3.10}$$

In RNN, the load of neurons parameters is shared at all time steps within the network and then the gradient at each output depends on the present and former time step calculations. To calculate the gradient at time $t = 5$, backpropagation has to be executed 4 steps back in time and all results have to be summed up.

3.4.1 Vanishing and Exploding Gradient

Recurrent neural networks (RNNs) tend to suffer from vanishing/exploding gradients. The basic architecture of the RNNs affects negatively the ability of the network to learn long-distance dependencies between the elements of the sequence. [RB09]. In particular, if the matrices involved in the calculation of the gradient have small values - less than 1 - the resulting subsequent gradients will tend to 0, leading to the so-called vanishing gradient problem. On the other hand, if the value of the matrices is greater than 1, the gradient will tend to infinite, also called exploding gradient problem. In the following paragraph, two architectures able to overcome this problem based on "memory" and "forget" cells will be introduced. One handles the vanishing problem through the Long Short-Term Memory (LSTM) architecture, the latter addressed the same issue with the Gated Recurrent Unit (GRU). [HS97]

3.4.2 LSTM

Long Short-Term Memory neural networks [HS97] architecture is based on the concept of gate and cell state. Similarly to the RNN, the cell state carries the important information down all the network, like a memory state, allowing the network to model dependencies from distant parts of the input. The forget and update gates allow to remove or insert information in the cell state.

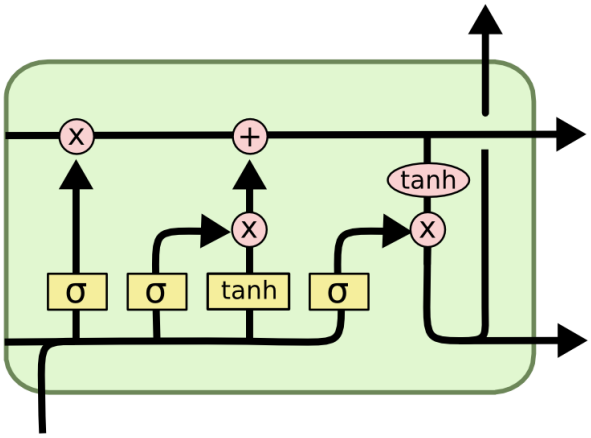


Figure 3.4: LSTM neuron architecture

As it can be seen in the figure above, the hidden state h_{t-1} is concatenated with the input element x_t. The forget gate consist in a sigmoid (σ) function returning a value between 0 and 1. If the value is close to zero the network will forget the information going through.

$$T_f = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{3.11}$$

Then the same concatenation between h_{t-1} and x_t is given in input to the sigmoid and the hyperbolic tangent function. The former regularizes the network, the latter decides what information will contribute to the cell update. The two values are then

multiplied. This is the "update gate".

$$\begin{aligned} T_u &= \sigma(W_u[h_{t-1}, x_t] + b_u) \\ c_u &= \sigma(W_c[h_{t-1}, x_t] + b_c) \end{aligned} \tag{3.12}$$

The time-step is then computed after the cell is updated:

$$\begin{aligned} c_t &= T_u \cdot c_t + T_f \cdot c_{t-1} \\ h_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \cdot \tanh(c_t) \end{aligned} \tag{3.13}$$

3.5 Encode Sentences

The architecture of the Recurrent neural networks can be used to obtain a vector representation of full sentences by condensing all word embeddings of a sentence into a single vector. The RNN acts as an encoder converting variable-length source sequence to a fixed-length vector that can be then used to classify the whole sequence. An encoder is a many-to-one neural network able to condense the information coming from the words of a sentence in a vector representation and are probably the most natural way to represent text sequences. Paired with a one-to-many neural network at the output able to convert the vector representation in a sequence of outputs, it creates what is called an encoder-decoder architecture used for sentence prediction and language translation tasks. The innovations introduced by using RNNs as sentence encoder-decoder for language generation tasks became fastly obsolete with the discovery of the Transformer Architecture. [ANN17]

3.6 Attention Mechanism

Encoding a sentence with a Recurrent neural network proved to be inefficient in handling long-range dependencies. According to Bahdanau et al [DKY16] this was due to the fixed length of the input vector and they stressed the importance of selectively focus on specific parts of the sequences. This consideration led to the development of a new way of encoding text sequences with a new mechanism called "attention". [ANN17] The main drawback of classical RNN/LSTM is that the encoder executes each timestep computation by looking at the current input state and only the last

hidden state. All previous computations are ignored and the information passed to the next step is entirely encoded in one hidden state vector, if this vector contains badly encoded information also the final condensed vector will not be able to effectively identify the correct class. Further observations hinted that the encoder creates a bad vector summarization when it tries to understand longer sentences. This issue of RNN/LSTMs is called the long-range dependency problem. Cho et al. [al14], author of the encoder-decoder network, showed how the performance of the encoder network worsens rapidly as the length of the input sentence increases.

A solution to this problem was proposed by Bahdanau [ANN17] that came up with an idea suggesting to give importance to the relative importance between input words when calculating the final context vector. The main idea is to leverage the dynamic vector representation of the original sentence, called context vector c_i, which is calculated as a weighted sum of the hidden states of the encoder. This represents a selective focus over the source words in order to output the current target word. Therefore, during every time-step i, the decoder will use a context vector that is different from the context vector at time-step $t-1$ to output the target word. In other words, the decoder can search for part of the source sentence x that it thinks are relevant to predict the target work y_i. The context vector to produce the target word y_i is given by:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{3.14}$$

α_{ij} represents the probability the output word y_j matches the input word x_i. This approach allows the network to capture long-range dependencies, and to outperform the previous state-of-the-art in machine translation. This mechanism was also the main idea behind the Transformer architecture being the state-of-the-art NLP technology.

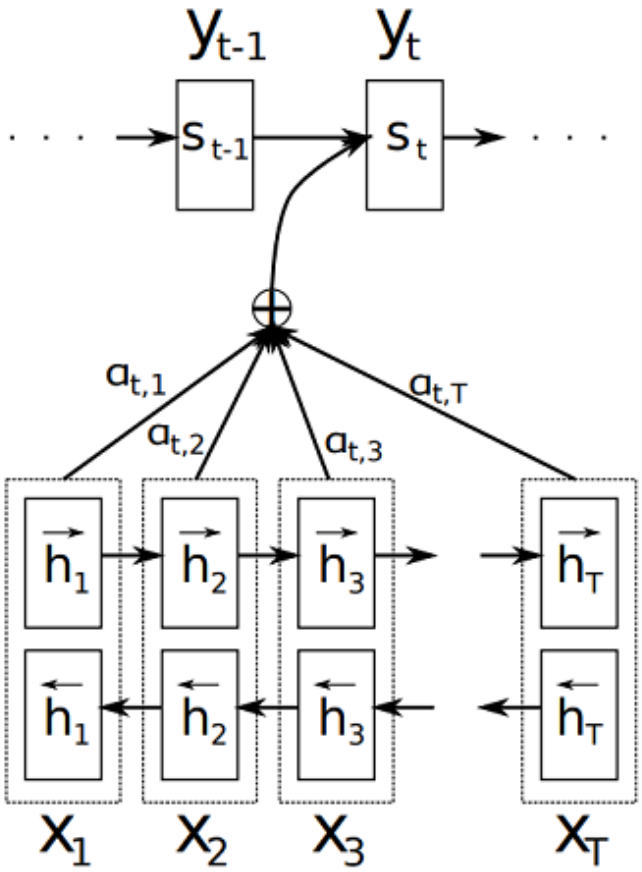


Figure 3.5: Attention Mechanism

3.7 Transformer

The Transformer is an encoder-decoder architecture that uses the attention mechanism to improve the translation and speed up the training. It maps the sentence into a vector representation and the decoder reconstructs the sentence.

As seen in the figure 3.6, the Transformer is a very complex model. The encoder is composed of 6 identical stacked layers where each layer has 2 sub-layers. The first layer is a multi-head self-attention mechanism and the second is a fully connected feed-forward neural network. The dimension of all embeddings is 512.
The decoder is composed also of 6 layers (N) but in addition to the 2 sublayers, there is a third one that performs multi-head attention on top of the encoder stack. Each computation is followed by layer normalization which normalizes the sum of the input

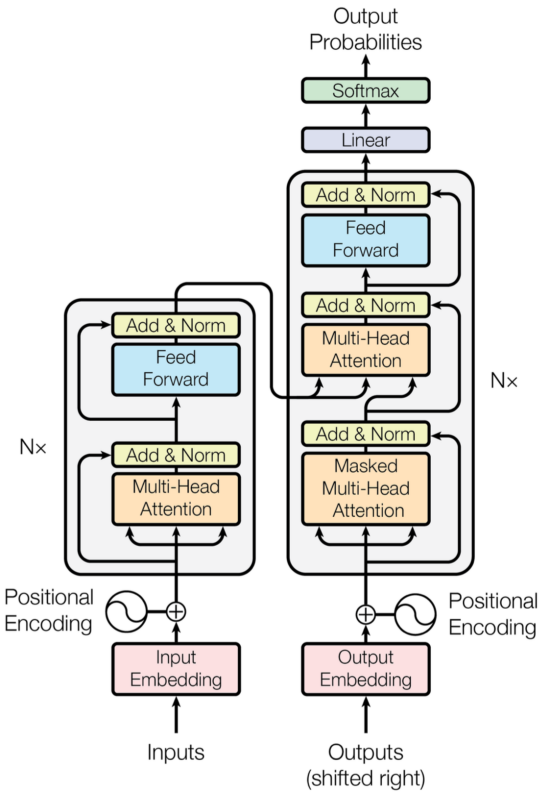


Figure 3.6: The Transformer model architecture

of the sub-layer with the output of the sub-layer itself. Also, the self-attention sub-layer is modified in the decoder stack to prevent positions from attending to subsequent positions. This is called masking and combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

The main novelty is the multi-head attention layer that is a modified version of the original one introduced in [ANN17]. Every word embedding generates query Q, key K, and value V that are vectors of d_k dimension. They are obtained by multiplicating the embeddings with three different matrices W^Q, W^K and W^V randomly initialized previously. The task of the attention is to map the value vector V to the output using a combination of keys K and query Q to infer the correct amount of each weight. By multiplying the query by all the keys, dividing by $\sqrt{d_k}$ and applying a softmax function the attention is computed. The multiplication of it for the value vector gives in the output the Scaled Dot-Product Attention matrix.

$$Attention(Q, K, V) = softmax\frac{QK^T}{\sqrt{d_k}}V \tag{3.15}$$

To improve the model multiple representations of the input sequence are calculated in different subspaces. [ANN17] The attention calculation is executed $h = 8$ times (called heads), every time with a different Q, V, and K thanks to the random generation of the matrices $W^Q, W^K, andW^V$. This makes it possible for the system to focus on different positions of the sequence, balancing the importance of words. The output of each head is concatenated and projected again.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, ...head_h)W^O \tag{3.16}$$

where

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{3.17}$$

The multi-head approach allows for a high level of parallelization since each head is computed independently from the others.

The transformers offers two main benefits:

- These models do not process an input sequence token by token rather they take the entire sequence as input in one go which is a big improvement over RNN based models because now the model can be accelerated by the GPUs.

- no labeled data is needed to pre-train these models. Providing a huge amount of unlabeled text data is enough to train a transformer-based model. This trained model can be used for other NLP tasks like text classification, named entity recognition, text generation, etc. This is how transfer learning works in NLP.

3.8 Bert

All the concepts introduced in the previous sections are the build blocks of the BERT model that combines the transformer architecture with a new training procedure. [JT18] The implementation of BERT is solely based on the encoder architecture of the Transformer. In particular, the authors created two main models, “BERTBASE” –

with several stacked encoders $L = 12$, the hidden dimension $H = 768$, and the number of self-attention heads $A = 12$ – and "BERTLARGE" that has $L = 24$, $H = 1024$ and $A = 16$ The input sequence is initially tokenized into a vector representation using the WordPiece tokenizer that proved more effective by traditional word tokenizers [Arm+17a] and its vocabulary. Every sequence is pre-posed with a [CLS] token and each sentence in a pair is separated by the [SEP] token. This allows BERT to handle sentence and word-related tasks. The system is built around two main steps: pre-training and fine-tuning, as shown in the figure below.

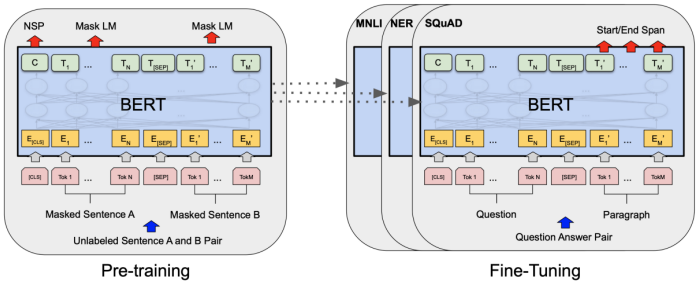


Figure 3.7: Bert Pre-Training and Fine-Tuning

The pre-training procedure follows the paradigm of "transfer learning". Transfer learning takes the knowledge gained during the solving of a problem and applies it to a special but related problem. The original model being trained on a large dataset can then be adapted with the knowledge contained in a smaller dataset. Such deep learning model is called "pre-trained model". The best known examples of pre-trained models are the deep learning computer vision models trained on the ImageNet dataset. Therefore, it is better to use a pre-trained model as a starting point for solving a problem than to create a model from scratch. In NLP tasks the system learns the Language Model of the data it is trained on. Then, when fine-tuning for a given purpose, this knowledge is further exploited to initialize the system for a new task. BERT is pre-trained over two NLP tasks: Masked Language Modelling and Next Sentence Prediction. The Masked Language Modelling works by exploiting bidirectional training. Part of the text corpus (15%) is randomly masked with a special token [MASK]. To avoid inconsistencies with the fine-tuning procedure, where the mask token is not used, given the 15% of the tokens to be replaced, only 80% are masked with [MASK], while 10% are replaced with a random token and the remaining 10% left unchanged. A cross-entropy function is used and the embedding T_i is returned for the i-th token.

The Next sentence prediction instead task aims at predicting whether a sentence is a sentence coming immediately after a given one. The dataset for this pre-training procedure can be built with any text corpus: given a sentence pair, 50% of the time the second sentence is the actual following sentence, labeled as "isNext", the other 50% a random sentence from the corpus, labeled as "NotNext". This procedure is very effective for Question Answering and Text Classification tasks.

The fine-tuning procedure depends on the final task we want the model to achieve and can be achieved in multiple ways.

- Train the entire architecture, the entire pre-trained model can be further trained on specific datasets and feed the output to a softmax layer. In this case, the error is propagated through the entire architecture and the pre-trained weights of the model are updated based on the new dataset.
- Train some layers while freezing others: another way to use a pre-trained model is to train it partially. This can be done by freezing the weights of initial layers while training only the other layers.
- Architecture freeze – by freezing all the layers of the Bert model and attaching a few neural network layers on top of the output Bert . Note that the weights of only the attached layers will be updated during model training.

For the task of Text Classification the most common approach is the third one as the training of Bert hidden layers requires extensive resources.

Chapter 4

Weak Supervised Task

The idea of trying weakly supervised approaches came while tackling the task of creating a model in a context where no labeled data was available. Initially, during the internship, we decided to undertake a supervised approach. Due to the lack of a labeled data, the first move was to build a dataset by aggregating resources available online. For this purpose, a Web Crawler was developed and a dataset of 47017 companies was built, where each company was assigned with multiple tags. Once this dataset was built, extensive research was taken on to find the best supervised models solving a multilabel classification problem. The model implemented were a FastText classifier and then a Bert model, that are not further discussed in the thesis. As the retrieval of annotated data took a long time to be carried out, the company supervisor suggested to research new models capable of learning without the burden of using annotated data. Using the already build dataset as a benchmark, we chose some innovative models leveraging only a fixed set of keywords information for each class and analyzed the performance. The new models would then be used to solve similar classification problems. Since only few existing researches were tackling the multilabel weak supervised problem, we decided to conduct the model analysis using a subset of the created dataset. It consists of documents having only a single label. From the original multilabel dataset, a smaller multiclass dataset was created by removing all documents being associated with multiple classes. The multiclass dataset will be described in the next section. The datasets are property of Eutopia Green and cannot be disclosed in their entirety without the company's consent. The weakly supervised models analyzed are ConWea [MS20] and LOTClass [Men+20]. The two papers prove some remarkable achievements

when handling standard academic datasets. Besides that, it is hard to predict the performance on a specific task with all the problems that come with it. They both take advantage of the newly researched BERT contextualized embeddings in order to better use the keywords given as input to the weak classification model. The problem when using words as a source of information is the uncertainty regarding the word meaning in the specific context where it is used. Such words, with multiple meanings, are called ambiguous words. The process of finding the exact meaning of an ambiguous word is called Word Sense Disambiguation. A classic example of ambiguous words is "bank", meaning both financial institution and bed of the river depending on the context is used. Original word embeddings could not tackle this task since the mentioned word had the same representation regardless of the context. Contextualized models offer an alternative solution to this problem offering different vectorial representations for the same words depending on the context. ConWea deals with the disambiguation problem by finding embeddings clusters for each keyword and associating it to different data classes. With a similar approach, LOTCLass creates vocabularies of words having similar embeddings and benefits from them to identify the correct usage of each word. Both models have different pros and cons. While ConWea requires extensive computational resources but it is applicable to all types of data classes; LOTClass is extremely versatile and faster requiring just a few, but less reliable, information depending on the data domain. These two models are introduced in sections 4.2 and 4.4.
In the next paragraph we introduce the dataset built trough our Web Crawler. It exposes the task we are dealing with and the problems that we had to face.

4.1 Dataset Description

As introduced before, an initial multilabel dataset was created using a web crawler where each business description is tagged with multiple labels. From this multilabel dataset, a multiclass dataset was obtained by keeping only the documents having an unique tag. The two datasets consist of a set of companies operating in various sectors, each company being associated with a description of varying size, between 10 and 2000 characters, and classification according to areas of expertise.

The first dataset was constructed in collaboration with the company. The data provided by the web crawler was mapped into 13 categories by associating the retrieved

tags into macro-categories. Each company can have one, two, three, or four different classifications. The resulting dataset contains 47107 companies.

The second dataset was constructed by removing companies associated with more than one label to test performance on the single-label classification. This dataset contains 36044 companies.

The categories are as follows:

- Agriculture
- Buildings
- Constructions
- Energy
- Financial services
- Food & beverage
- Healthcare
- Logistics
- Manufacturing
- Mining
- Public administration
- Transportation
- Utilities (electricity, water, waste)

Each label defines the economic sector in which the company operates, some tags are self-expressive but others require a more strict definition. For instance with the category Buildings, we want to tag all companies handling the administrative part related to renting offices, houses, and warehouses. While with Construction, we want to tag all companies involved in the construction of structures and buildings. These two classes will prove difficult to discriminate because of their apparent similarity. Another example is related to the connection between the Mining class that is referred to the various practices related to harvesting minerals from the soil. And the Utilities class that is more heterogeneous and includes services related to water supply, electricity,

and waste recycling. The energy class should not be confused with the electricity subclass. The former defines all companies involved in energy production, while the latter deals with services related to the transfer and handling of energy.

An example for the category Utilities is:

Nvent is a global provider of electrical connection and protection solutions.

This short sentence contains all the information available for the company "Nvent". Even if it is quite short, the word 'electrical' allows us to guess the correct category to be utilities. The length of each document is highly variable as it can be seen in 4.1. By using a simple tokenizer, splitting each word on the whitespace character, the mean length is 81 tokens with 62% of documents having a shorter length.

The dataset is characterized by a large class imbalance as can be seen in 4.2 The largest class is healthcare with 6632 instances, while the smallest is Public Administration with 669 instances. This imbalance required the use of appropriate metrics for analyzing the accuracy of the models.

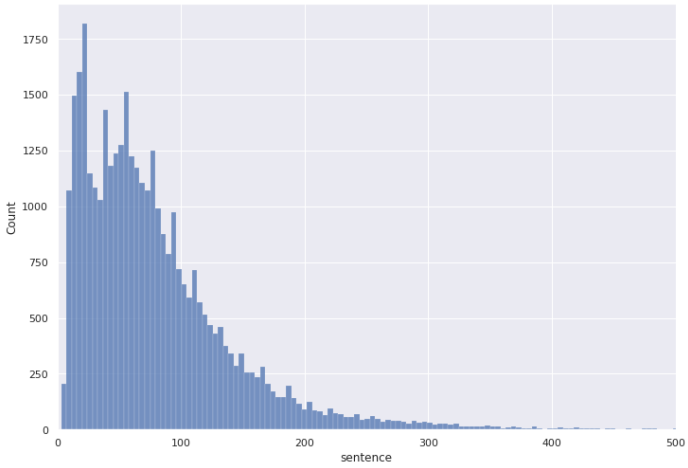


Figure 4.1: Distribution of document length (with simple tokenizer)

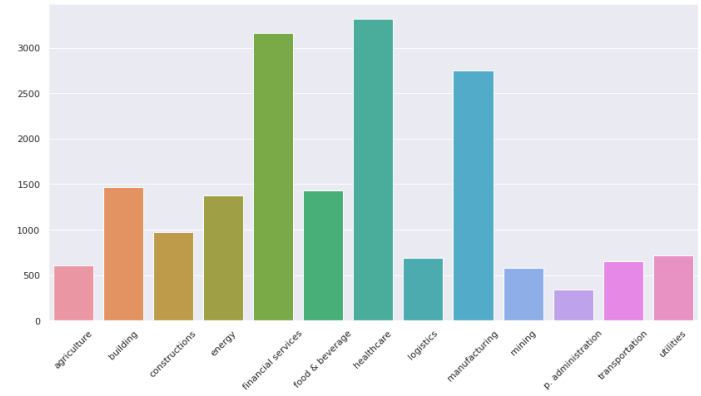


Figure 4.2: Distribution of categories samples.

4.2 ConWea, Contextualized Weak Supervision for Text Classification

A recent approach, created in order to tackle the weak text classification task called ConWea, was proposed by Mekala and Shang[MS20]. This innovative research, published in 2020, proposes to train an attention model over multiple iterations. It uses an initial dictionary of category indicative words that gets upgraded with new words at the end of each iteration. The algorithm is divided into three steps: Contextualization, Classification and Expansion.

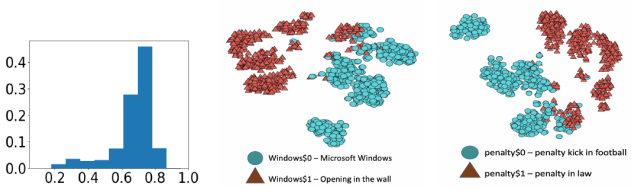
4.2.1 Contextualization

The main problem that the algorithm tries to fix is the word sense disambiguation. It is referred to a computational linguistics problem concerned with identifying which sense of a word is used in a sentence. The word "windows" acquires very different meanings depending on whether it is used as the proper name of the software company with the same name, or as the plural of an "opening in a wall". For this reason, the authors decided to leverage the Bert embeddings representation of each word by gathering all the embedding, of all the occurrences of the same word. Then, they applied a K-Means clustering algorithm to group word embeddings. Different word senses are encoded in the text adding a suffix $1, $2 .., depending on the nearest cluster center. This clustering process aims to distinguish the different meanings of the same words.

The K-Means algorithm works by clustering all contextualized representations

into K clusters, where K is the number of interpretations. It is a suitable clustering algorithm since it is fast and the cosine similarity is equivalent for unit vectors. The value of K clusters needs to be set before starting the algorithm. The authors introduce an approach based on a similarity threshold τ to automatically choose the suitable number of clusters. If the cosine similarity of the two cluster centers is greater than τ, they belong to the same word interpretation. Therefore, K is chosen to be the largest number such that the cosine similarity between any two cluster centers is no more than τ .

$$K = argmax_K\{cos(c_i, c_j) < \tau \forall i, j\} \tag{4.1}$$



(a) Similarity Distribution: Windows (b) Cluster Visualisation: Windows (c) Cluster Visualisation: Penalty

Figure 4.3: Clustering Example of words "windows" and "penalty".

τ is chosen making two assumptions:

1. For each seed word the majority of its occurrences follow the interpretation of the user.

2. The majority of the seed words are not ambiguous, meaning that they only have one interpretation.

Given these assumptions, τ is chosen by applying the median of all pairwise similarities between all occurrences of the same word and then applying the median on the median of each seed word.

$$\begin{aligned} \tau(s) &= median(\{sim(b_{s_i}, b_{s_j})\forall i, j\} \\ \tau &= median(\{\tau(s)|\forall s\}) \end{aligned} \tag{4.2}$$

After this process, we obtain the *contextualized corpus* that will be used by the Classification step.

4.2.2 Classification

To train a weak classifier, a function is needed in order to assign labels to some documents using an appropriate heuristic. For this purpose, the author chooses a simple heuristic based on counting. Each document is assigned a label whose aggregated term frequency of seed words is maximum. Given $\text{tf}(\hat{(}w), d)$ and the term-frequency of a contextualized word in the document d, S_c the set of seed words of class c, the label is assigned to the document d as follows:

$$l(doc) = argmax_l\{\sum_i \text{tf}(s_i, d)|\forall s_i \in S_l\} \tag{4.3}$$

Once the documents are labeled, a Hierarchical Attention Neural Network (HAN) [Zic+16] is trained. The intent is to derive sentence meaning from the words and then derive the meaning of the document from those sentences. As not all words are equally important, the attention model is used so that sentence vector can have more attention on "important" words. The Attention model consists of two parts: Bidirectional RNN and Attention networks [ANN17]. While bidirectional RNN learns the meaning behind those sequences of words and returns vector corresponding to each word; the Attention network gets weights corresponding to each word vector using its own shallow neural network. Then, it aggregates the representation of those words to form a sentence vector i.e it calculates the weighted sum of every vector. This weighted sum embodies the whole sentence. The same procedure applies to sentence vectors so that the final vector represents the meaning of the whole document. Since it has two levels of attention model, therefore, it is called hierarchical attention networks. Once the HAN model is trained on the contextualized corpus, using the generated pseudo-labels. The predicted labels are used in the Expansion step.

4.2.3 Expansion

Given the contextualized documents and their predicted class labels, the process executes an algorithm that ranks contextualized words and adds the best ones into the seed word sets. An ideal word s for the label l should satisfy some conditions: should be an unusual word that appears only in documents belonging to l with significant frequency. A suitable ranking function is defined following three aspects:

1. Label Indicativeness : the posterior probability of a document belonging to the

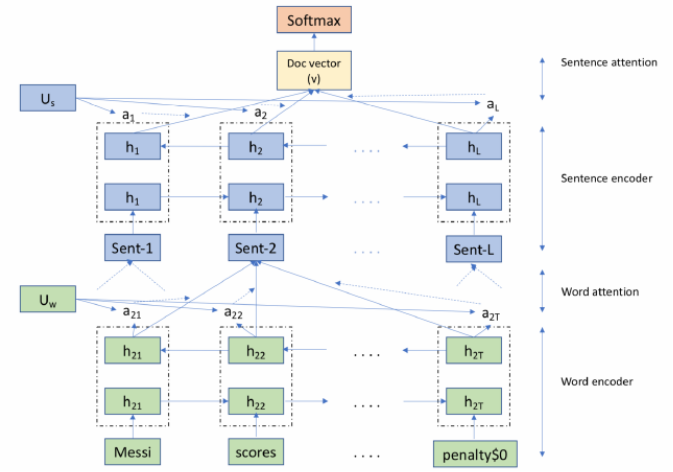


Figure 4.4: HAN neural network structure.

class C_j after observing the presence of word w $P(C_j|w)$ should be very high.

$$LI(C_j, w) = P(C_j|w) = \frac{f_{C_j,w}}{f_{C_j}} \tag{4.4}$$

f_{C_j} refers to the total number of documents that are predicted as class C_j , and among them, $f_{C_j,w}$ documents contain the word w. These counts are based on the predictions of the results on the input unlabeled documents.

2. Frequency: a seed word s of label l should appear in the documents belonging to label l with significant frequency. To measure the frequency score, the average frequency of seed word s is computed in all the documents belonging to label l. Since average frequency is unbounded, than function is applied to scale it, resulting in the frequency score.

$$F(C_j, w) = \tanh(\frac{f_{C_j}(w)}{f_{C_j}}) \tag{4.5}$$

3. Usuality: Seed words should be unusual. This requirement is added to the ranking function by considering the inverse document frequency (IDF). Let n be the number of documents in the corpus D and $f_{D,w}$ represents the document frequency of word w, the IDF of a word w is :

$$IDF(w) = \log(\frac{n}{f_{d,w}}) \tag{4.6}$$

It should be observed that IDF does not require the document classification and can be computed only once.

These three measures are combined using the geometric mean giving a score $R(C_j, w)$ of a word w for a class C_j.

$$R(C_j, w) = (LI(C_j, w) \times F(C_j, w) \times IDF(w))^{1/3} \tag{4.7}$$

Based on the score function, the top words for each label are added to the initial set of seed words. There is one last problem that needs to be solved. In the first step all the documents were contextualized but the initial seed words never did. As they appear as unique words without any context, it is not possible to get the embedding beforehand and find the nearest cluster to assign the correct word meaning. The author proposes to use the prediction of the first model trained to disambiguate the seed words. Using the classified documents and the ranking function, all possible interpretations of the same initial seed word are ranked. Because the majority of seed word occurrences are assumed to belong to the user-specified class, the intended interpretation should be ranked the highest. Therefore the only top-rated interpretation is retained.

4.3 ConWea Applied to Companies Dataset

Having discussed with my supervisor the idea of using a weak supervised algorithm to tackle the company problem, it was decided to try this algorithm on the available labeled dataset to check the performance. Once the performance on this small benchmark dataset was assessed, the next step would have been to leverage the whole amount of unlabeled data available to train this algorithm and solve similar tasks. In collaboration with some company employees, a dictionary of seed words was to build as instructed by the authors of ConWea. An initial dictionary with 3 words for each class was built similarly to the original experiments. This was deemed necessarily to have comparable results with the one obtained by the authors.

This is the initial dictionary of seed words and care has been taken to use mostly unambiguous words. Furthermore, as stated previously each class is linked to 3 unary keywords.

1. Public Administration: education, government, administration.
2. Healthcare: health, healthcare, pharmaceutical .
3. Food & Beverage: food, beverages, restaurant .
4. Building: property, rent, housing .
5. Transportation: transportation, vehicle, mobility .
6. Constructions: construction, demolition, architecture .
7. Manufacturing: manufacturer, fabrication, manufacturing.
8. Utilities (electricity, water, waste): water, waste, electricity .
9. Energy: solar, energy, renewable, .
10. Mining: mining, excavation, ores .
11. Financial Services: bank, finance, financial .
12. Agriculture: agriculture, farming, cultivation .
13. Logistics: delivery, shipping, logistics .

In addition to this dictionary of keywords, we tried to make the dictionary as exhaustive as possible to further improve the initial classification. The following is the complete dictionary of keywords available for each class. On average, a label is associated with 9 keywords. Some of them are n-grams and they will be converted into a unigram during the preprocessing step by removing whitespaces:

1. Public Administration: education, government, administration, regulation, public service, governance, public authority, public office, government agency, department, public affairs, legislation.
2. Healthcare: health, diseases, blood, healthcare, dental, pharmaceutical, death, cancer, therapeutic, medical.
3. Food & Beverage: food, beverages, eat, drink, restaurant, ingredients, food production, food industry, beverage industry, food retail, food and drink, beverage production, catering, food delivery, plant-based, meat, cheese, insects, protein, vegan .

4. Building: real estate, properties, rent, residential, commercial building, industrial building, green building, sustainable building, apartment, housing, hvac, air condition, smart appliances, indoor, district heating.

5. Transportation: driving, car, bus, vehicle, passengers, mobility, transit, urban mobility, mode of transportation, travel, traffic, city transport, electric mobility, shared mobility, maas, micromobility, scooter, e-bike .

6. Constructions: construction, construction process, demolition, builder, architecture, construction & demolition, deconstruction, infrastructure, concrete, building materials, construction site.

7. Manufacturing: manufacturer, manufacture, fabrication, industry 4.0, manufacturing, production process, value chain.

8. Utilities (electricity, water, waste): water, electrical, waste, recycling, disposal, water management, waste management, electrical supply, power supply, public utility, electricity, wastewater.

9. Energy: solar, energy, renewable, wind, thermal, power, smart grid, energy storage, photovoltaic, kinetic, tidal, wave energy, hybrid, fuel cell, nuclear, offshore, onshore.

10. Mining: mining, mineral, excavation, metal, ores, extraction, quarry, coalmining, drilling, mineral exploitation.

11. Financial Services: asset, finance, invest, bank, credit, online banking, banking, insurance, crowdfunding, financial, cryptocurrency.

12. Agriculture: agribusiness, farm, farming, plants, cultivation, agronomy, tillage, horticulture, hydroponics, aeroponics, fishing, pest, soil, nutrient, precision farming, fertilizer, smart agriculture, algae cultivation.

13. Logistics: delivery, shipping, overhaul, freight, shipment, transporter, supply, supply chain, distribution network, logistics, distribution .

Some initial remarks can be made when analyzing the algorithm execution, extensive performance analysis is discussed in section 5.3. The algorithm execution proved to be computationally and memory intensive. The algorithm firsts gathered all embeddings

and computed the τ parameters to be equal to 0.76 using the seed words.
The bottleneck of the execution can be traced back to the clustering algorithm. The retrieval of the Bert embedding, combined with the execution of the K-Means algorithm, even on the small dataset of 36k document, takes almost 19 hours and the storage of the embedding of each word occurrence requires a disk memory of about 25 Gb. To speed up the Bert retrieval, a GPU, offered by Google Colab, was used, even though not much could be done to speed up the clustering process.
The K-Means needs to be executed for each word appearing in the documents using all the embeddings of all the occurrences of that word. Embeddings that in this case is a vector with dimension of 748. The algorithm version used is the one from the sklearn framework. Sklearn solves the k-means problem using either Lloyd's or Elkan's algorithm. The average complexity is given by $O(knT)$, where n is the number of samples and T is the number of iteration. The worst-case complexity is given by $O(n^(k + 2/p))$ with n = number of samples, p = number of features. The algorithm execution also depends on 2 parameters that stop the process: the number of maximum iterations (default at 300) and the relative tolerance value that declares if the convergence has been achieved (default at 0.0001). [Cou+20]
Even if the implemented version of the algorithm is quite fast, the algorithm suffers from the fact that it needs to compute the clusters for each unique word. In our dataset, this equals about 80 thousand words. A solution could be to leverage the parallel computation using cloud services like AWS or Azure. Unfortunately, this requires a not trivial amount of economic resources and research to set up the whole cloud environment. Over 80 thousand unique words 13 thousand were disambiguated.
The execution of the code halted multiple times due to an Out Of Memory error, the cause was traced back to an error in the implementation of the tf-idf function: to get the rankings of words, for each class, each document needs to be converted into a one-hot-encoding representation: consisting in a sparse vector of size equal to the total vocabulary size. Then, each component of the sparse vector is summed-up over all documents of the same class. This operation uses a huge amount of memory when the vocabulary is huge, as each vector of each document has to be stored in memory. Considering that in Python a list of 80 thousand integers uses 640056 bytes and some classes have more than 6 thousand this operation requires more than 3 Gigabites of RAM. The authors, for some reason, never encountered this problem but a simple

solution is offered by the Sklearn library. Representing a sparse vector can be done efficiently with a dictionary: instead of saving in memory all the components equal to 0, we can use a dictionary to only store elements different zero with (position:values) pairs. Then, the sum can be executed keeping this vector representation that requires insignificant amounts of memory. A pull request was asked on the ConWea git repository to offer a simple fix to this problem, also to the authors of the algorithm.

4.4 LOTCLASS, Label Name Only Classification

After encountering some limitations in the ConWea algorithm, that will be discussed in section 5.3, our focus shifted on a new research that was published while we were working on the ConWea algorithm. This new approach to tackle the weakly supervised text classification problem is proposed by Meng in the paper "Text Classification Using Label Names Only: A Language Model Self-Training Approach" [Men+20]. Without using any labeled documents, the possibility of training classification models on unlabeled data, using simply the label name of each class, is investigated. For category understanding and document classification, BERT pre-trained neural language model is exploited as a generic linguistic knowledge source and representation learning model. This method links, semantically, similar terms to label names, discovers category-indicative words and trains the model to predict their inferred categories. Finally, it generalizes the model by self-training. To categorize documents, a human expert just has to know the label name (i.e., a single or a few representative words) of each class, in contrast to existing supervised and semi-supervised algorithms that learn from labeled documents.

The LOTClass algorithm follows 3 steps. Firstly, a category vocabulary is constructed for each class. By using a pre-trained Language Model, semantically correlated words to each label are found and added to the vocabulary. Then, the Language Model collects category indicative words in the unlabeled corpus to train itself to detect category information. In the last step, the Language Model is generalized on the task via document level self-training.

The idea behind this approach relies on the following intuition: words that are interchangeable, most of the time, are likely to have similar meanings. By using the

pre-trained BERT masked language model (MLM), it is possible to predict what words can replace the label names in relation to most of the contexts. Specifically, for each occurrence of a class name in the documents, its contextualized embedding vector h produced by the BERT encoder is fed to the MLM head 3 that outputs a probability distribution over the entire vocabulary, indicating the probability of each word w appearing in that position:

$$\mathrm{p}(w|h) = \mathrm{Softmax}(W_2\delta(W_1h + b)) \tag{4.8}$$

where $\delta()$ is the activation function; W_1, W_2 and b are learnable parameters that were pre-trained with BERT. The author empirically observed how the top-50 predicted words usually have a similar meaning to the masked word. For this reason, the threshold of 50 words defines the number of valid replacement words for each occurrence of the label name in the corpus.
So for each label, identified by a word, all the occurrences in the document are retrieved, 50 valid replacements are obtained by masking each word occurrence. Finally, a category vocabulary is formed using the top 100 words ranked by how many times they can replace the label name in the documents. Stopwords and words appearing in multiple categories are discarded.

4.4.1 Masked Category Prediction

Similarly to Mekala [MS20], this algorithm tries to solve the word disambiguation problem using BERT contextualized embeddings. Word meanings depend on the context and, for this reason, not every occurrence of the category keywords retrieved is linked to the correct label. Therefore, directly highlighting every occurrence of the category vocabulary, is error-prone. Also, the category vocabulary only covers a small subset of all the words with similar meanings.
To address these 2 problems a new task, called "Masked Category Prediction", is introduced. In the new task, a dataset of labeled documents is created. Each word in a document is masked and a list of valid replacements is retrieved by using again the BERT pre-trained model. If 20 or more of the top-50 replacements match the class vocabulary, then a category label is assigned to that particular word. The threshold of 20 was empirically decided by the author. This task creates a word-level supervision

where category indicative words are paired with their category labels.
This dataset is then used to train a multiclass text classification model. The model structure gets the output of BERT adds a linear layer on top of each contextualized word embedding and returns a distribution over the K classes for each word in the sentence. During the training, the category indicative word w is masked by using the special token [MASK], and it is trained to predict w's indicating category c_w by using a cross-entropy loss function:

$$\begin{aligned} L_{MCP} &= -L \sum_{(w,c_w)\in S_{ind}} \log \mathrm{p}(c_w|h_w) \\ \mathrm{p}(c|h) &= \mathrm{Softmax}(W_c h + b_c) \end{aligned} \tag{4.9}$$

where h_w is the contextualized embedding of the word w and S_{ind} is the set of the labeled category indicative words.

The masking of the word it is crucial as it forces the model to base it's prediction on the word's context instead of memorizing the word-class relation.

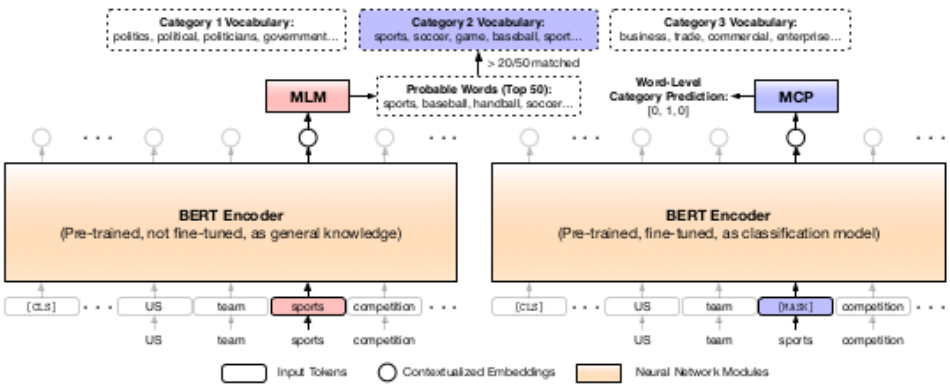


Figure 4.5: LOTClass

4.4.2 Self-Training

In the MCP task, the model was trained to output a class prediction for every single word. In order to be used for document prediction, it needs to be applied to a vector representation of the whole document. This can be done by using the [CLS] token,

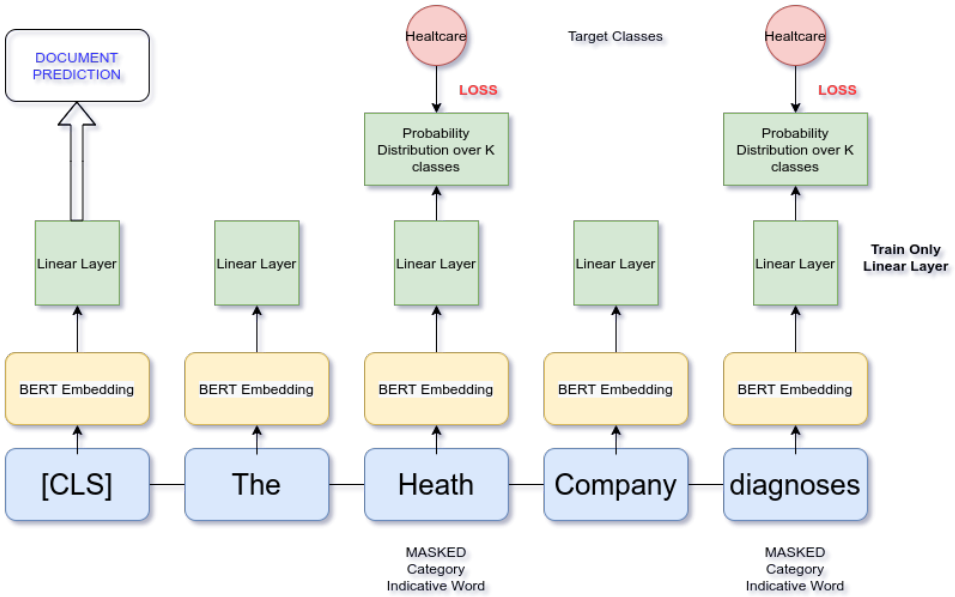


Figure 4.6: MPC training task

which allows the model to see the entire sequence to predict the category. The [CLS] token is added by the BERT tokenizer at the start of each document.

As only a small subset of data was used in the training process and the model was not trained on top of the [CLS] token, a new task is necessary to refine the model document prediction.

This task, called Self-Training, works by iteratively using the model current prediction P to compute a target distribution Q, that directs the model for refinement. The self-training function is defined as the Kullback–Leibler divergence loss that measures how one probability distribution is different from a second.

$$L_{ST} = KL(Q||P) = \sum_{i=1}^{N}\sum_{j=1}^{K} q_{ij} log \frac{q_{ij}}{p_{ij}} \tag{4.10}$$

where K is the number of classes and N the number of instances.

The target distribution Q can be computed in two ways: Hard labeling and Soft labeling. Hard labeling works by converting high confidence predictions over a threshold τ [Lee13] to one-hot labels. Soft labeling instead enhances high-confidence predictions and demotes low-confidence ones [JF16]. Meng observed how Soft labeling consistently

gives better results. The Soft labeling equation is the following:

$$q_{ij} = \frac{p_{ij}^2 / f_j}{\sum_{j'}(p_{ij}^2 / f_{j'})}, f_j = \sum_i p_{ij} \tag{4.11}$$

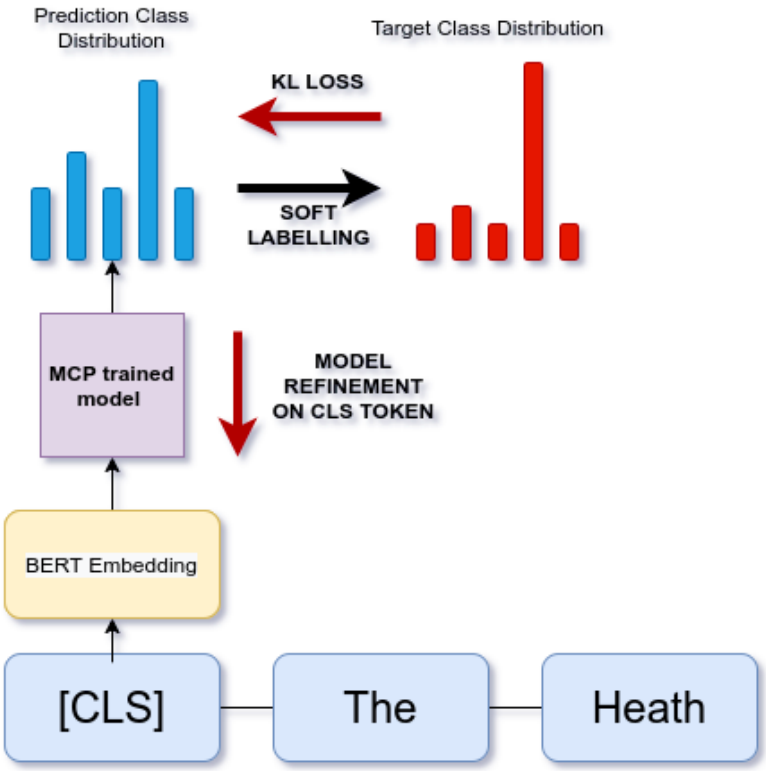


Figure 4.7: Self-training task

The model prediction is obtained by applying the classifier trained with the MCP task to the [CLS] token of each document.

$$\mathrm{p}_{ij} = \mathrm{p}(c_j | h_{d_i:[CLS]}) \tag{4.12}$$

4.4.3 Implementation

On the implementation side, some remarks can be made. The code is written in Python, all the tasks are executed using multiprocessing techniques to speed up the

computation. The library used is Pytorch that allows an easy usage of multiple cores for every computation. The step of finding the category indicative words, uses CPU multiprocessing. For each CPU core available, a task is executed and the data available is split evenly between each core. For the vocabulary construction and MCP training GPU, multiprocessing is executed. Batches of data are elaborated asynchronously between multiple GPUs to speed up the model training, or word embeddings retrieval. In our case, only one GPU was available so no GPU multiprocessing was used. In all experiments, the model is trained using the full length available for the Bert input (512), batch of size 32, and 4 accumulation steps. In this configuration, the loss is computed every 32*4=128 documents. The MCP task is executed over 3 epochs while the Self-Training task over just 1 epoch.

4.5 Joint Objective Training

While analyzing the individual limitations of the Meng and Mekala's algorithms, Mekala came up with the innovative idea of contextualizing words by appending a suffix ($0,$1..) in order to partially disambiguate the different meanings of words. However, it was observed experimentally that, on the dataset in question, this pre-processing step did not particularly affect the results: disambiguated word are just a small subset of the dictionary of seed words at the end of the last iteration. Moreover, this process is extremely computationally expensive. Executing the K-Means algorithm has an average complexity of $O(k * n * T * M)$ where k is the number of clusters, n the number of samples for each word, T is the number of iterations and M the total number of unique words). Even if the complexity is linearly dependent on each variable, with a dataset of only 36 thousand samples the pre-processing execution is not trivial and requires high-end hardware resources and a lot of storage space to save every word occurrence embedding. In addition to this, Mekala's algorithm has a major flaw. By contextualizing words, Mekala creates a new vocabulary and therefore cannot use the knowledge power of Bert's pretrained version. It, therefore, loses some of the information obtainable from a model trained on millions of documents and has to train new word embeddings using a standard Word2Vec approach on the small amount of data available.

On the other hand, Meng's algorithm takes full advantage of Bert's potential. However, it merely trains the model using only the class name. Experimentally, it has been

observed that some labels are too conceptually complex to be defined by a single word. For example, the class "public administration" cannot be defined by a single word and contains within it a number of subclasses that are difficult to generalize. The algorithm, therefore, fails at the moment of predicting certain classes obtaining a very low recall score. One the other side, it has the advantage of being able to find multiple category indicative words for each document that contribute all in the training process In his paper, Meng suggested how his algorithm could be maybe improved by training the model on a joint objective composed of both document-level and word-level tasks. The initial experiments, with the LOTClass algorithm on this dataset, show how the self-training approach is not able to enhance the prediction with the same effectiveness as other datasets: the label names, in this case, help the model in understanding the category's characteristics but not in fully describing them.
For this reason, leveraging also the available dictionary of seed words during the model training could improve the overall model performance.
We propose this new training process leveraging both keywords and label names and using a joint objective to refine the model predictions. The LOTClass algorithm is limited by the fact that refines the model prediction on the token [CLS] by only using the initial knowledge gained from the label names. Instead, we propose to train jointly the model on the [CLS] token, representing the full document and the category indicative words. The idea is to enrich the MCP created dataset with new documents where each [CLS] token is assigned the most suitable category. There are several ways to assign pseudo-labels to the [CLS] token using the seed words. As a proof-of-concept, we employ the same simple but effective method based on counting used in ConWea. Each document is assigned a label whose aggregated term frequency of seed words is maximum. Other more advanced heuristics can be used, for example, WestClass [Yu+18].

The loss function is computed in the same way as before. The only difference is that it is also calculated on the [CLS] token prediction. Once the model is trained, it gets further enhanced using the Self-Training technique. This approach proves to achieve comparable results to the Mekala algorithms in much less time. By implementing a few precautions to deal with the word disambiguation problem, it achieves results even better than the one achievable by using the best keywords possible with Mekala.

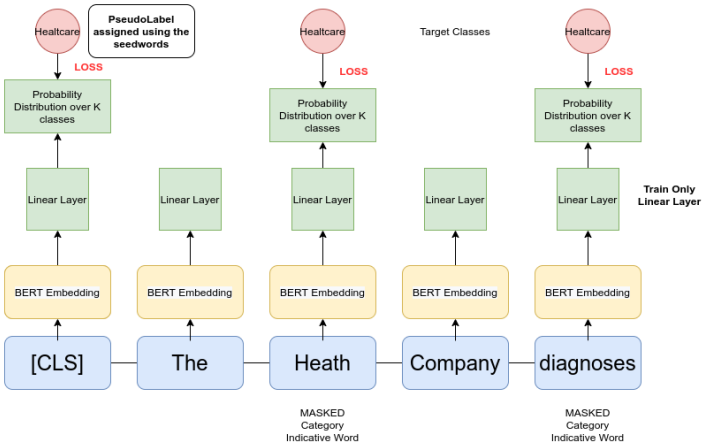


Figure 4.8: Joint Objective LOTClass Version

Chapter 5

Experimental Results

5.1 Introduction

In this chapter we analyse the experiments carried out. The experiments aim to analyse the weaknesses and strenght of ConWea (chapter 4.2) and LOTClass (chapter 4.4) algorithms. In addition we show the results obtained using the approach introduced in chapter 4.5 called "Joint Task" that integrates information from different sources. The experiments are as follows:

- 4 experiment on ConWea, one testing the basic capabilities, one testing a possible improvement by using a non decreasing number of seedwords, one using all the seedwords available and one testing the theorical best results possible. As the algorithm requires the execution of multiple iterations, performance is recorded individually using the F1-score for each class at each iteration.
- 3 experiment on LOTCLass, one testing the basic capabilities, one where the ConWea seedwords are used as label names, one to test how the threshold affects the performance of the algorithm. As the algorithm consists of only one iteration, it was decided to use the confusion matrix to analyse the results as it is the method of data analysis that provides the most information.
- 2 experiments are carried out the proposed Joint Task training: the first one combines the Masked Category Prediction task of LOTCLass with an heuristic assigning a pseudolabel by taking the class matching most seedwords, the second experiment instead uses an heuristic taking the most matched class that has found

at least 2 different seedwords with the intention of reducing uncertainty during pseudolabel assignment. Each experiment is paired with a comparison with the model using only keyword heuristics during training to show the improvements achieved.

5.2 Evaluation of a Classifier

Since it is impossible to prove mathematically the optimality of any text classifier, the evaluation of the performance is done in an experimental way, measuring its effectiveness i.e. its ability to make correct decisions during the classification process. At the end of the classification process, for each class c, it is possible to group the texts in the following way:

- TP (true positive) number of documents/texts correctly inserted in class c
- FP (false positive) number of documents/texts wrongly inserted in class c
- FN (false negative) number of documents/texts incorrectly placed under class c
- TN (true negative)number of documents/texts correctly not placed under class c

The fundamental metrics for the evaluation of any system for text classification are: accuracy, precision, recall and F-score. Each metric highlights different aspects of classification and are the following:

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{5.1}$$

Accuracy measures the percentage of documents/texts correctly classified

$$precision = \frac{TP}{TP + FP} \tag{5.2}$$

Precision measures the correctness of the classifier in terms of percentage of documents/texts correctly labelled in a given class compared with respect to the total number of documents labelled in that class.

$$recall = \frac{TP}{TP + FN} \tag{5.3}$$

Recall measures the completeness of the classifier in terms of the percentage of documents/texts correctly labelled in a class compared to the total number of documents it should have classified.

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{5.4}$$

F1-score is the geometric mean between precision and recall and gives a good measure of correctness and total completeness of the classifier.

To handle the unbalance between the different classes only considering the accuracy score to evaluate the overall model is not expressive enough. If a class is underrepresented in the testing dataset (that in our case correspond to the full dataset as no training set was used), its contribution to the overall model performance is not relevant. A model can completely fail to predict a class but still obtain very high accuracy. For this reason, we pair each model evaluation with a macro F1-score and weighted F1-score. The macro F1-score gives equal importance to all classes as it calculates the F1 metric for each label, and finds their unweighted mean. The weighted F1-score instead weights each F1 metric by the number of samples found in the testing set for that particular class. In all the experiments we have done, the weighted metric is always greater than the macro metric. This is a predictable phenomenon due to the fact that a larger number of samples corresponds to better model training: classes with a large number of samples always perform better than classes with few samples contributing more to the calculation of the F1-score. [Der16]

In the following sections, we will describe the results obtained from the different algorithms implemented in this thesis using the metrics just described.

5.3 ConWea Experiments

We decided to train the Mekala algorithm (ConWea) using different dictionaries and observe how the model performance is influenced. The contextualization step was executed only once and is the same for each experiment [MS20]. Considering the long processing time the Colab service could not be used as it limits the maximum amount of coninuous usage for a single user. The execution of the contextualization so was executed on an AWS EC2 instance with 2 vCPU, 8 GB RAM and 64 Gb storage. It takes about 14 hours.

5.3.1 Using Three Seedwords Dictionaries

In this experiment we test the model accuracy by using 3 keywords for each class as stated in 4.3. The dictionary expansion improve the model macro and weighted F1-score at each iteration and converges to a stable value. The macro F1-score starts at 59% and gets to 69% at the final iteration, meanwhile the weighted F1-score starts higher at 63% and improves at 74% as can be observed in figure 5.1.

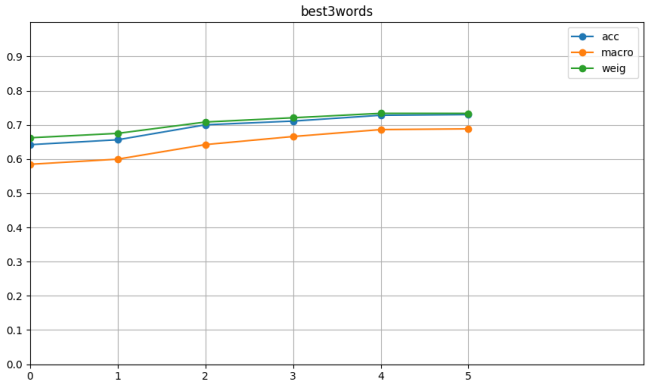


Figure 5.1: Performance metrics for ConWea over 6 iterations using 3 words dictionaries

Individual class performance is more variable and is subject to noise caused by the expansion of seedwords, here we comment one class achieving good results and one class whose results worsen, all other graphs can be found in chapter A. The analysis of individual class performance is not straightforward: the deterioration of the performance of one class may be indirectly due to the reduction of the accuracy of another class and not from the expansion of its own dictionary. Here we comment results where it is clear the causes of the model's behaviour.

The "healthcare" class achieved good results by improving the F1-score from 72% to 82%. In figure 5.2 it can be observed that the model decides that the initial word "pharmaceutical" is not meaningful enough and removes it even though it was added by the user. In this case the removal does not influence negatively the model F1-score but can be a source of problems.

The "manufacturing" class instead proved to be hard to handle. As can be seen in figure 5.3, at iteration number 3 the model adds the words "products" and "equipment" to the seedwords, these words introduce a lot of noise into the model as they appear

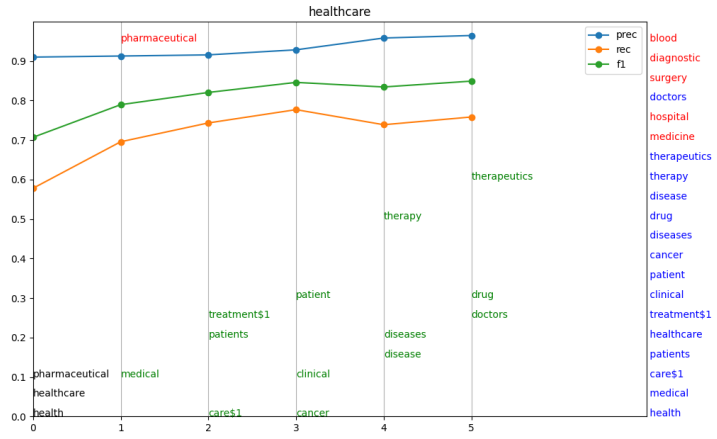


Figure 5.2: Perfomance metrics of class "healtcare" over 6 iterations, in black are the initial 3 words disambiguated (the first results are of the model without word disambiguation), new keywords added ad each iteration but contribute to the following training. On the side is the best keywords that the tf-idf model could choose, blue are the one the model manages to find and in red the one that the model misses.

also in a lot of documents not belonging to the "manufacturing" class, consequently it greatly improves the recall but reduces the precision by the same amount. The increase in recall and simultaneous reduction in precision keep the F1-score at a stable level but still is not satisfactory to have a model with only 50% precision. A disambiguation of the word "products" and "equipment" could have improved the classification but the contextualization step 4.2.1 found the best number of cluster for each word to be only one.

Remarks have to be made on the "public administration" class. As it can be observed in 5.4 the training starts with a low precision that gradually increases and high recall that gradually decreases. This trend improves the F1-score but shows how the model is trying to learn to classify correctly only a subset of all documents of this class, it progressively deletes the words "administration$1" and "education" suggesting the model is converging on classifying correctly only documents related to the word "government" ignoring the other subtopic of the "public administration" class.

These results show some limits of these models and hint at some changes that can improve the performances. ConWea's algorithm has some issues in handling

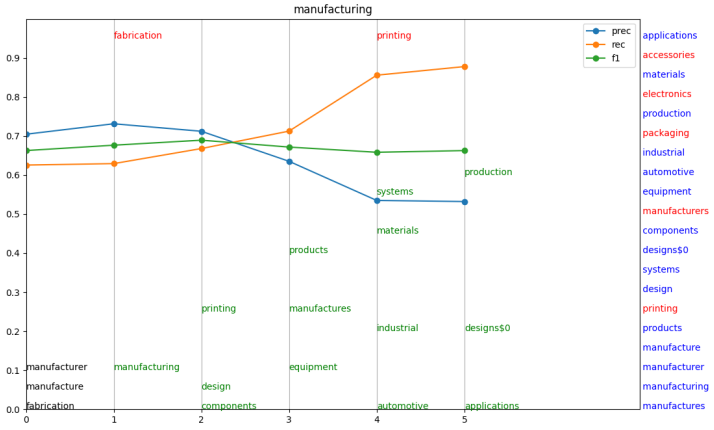


Figure 5.3: Perfomance metrics of class "manufacturing" over 6 iterations, see figure 5.2 for chart legend.

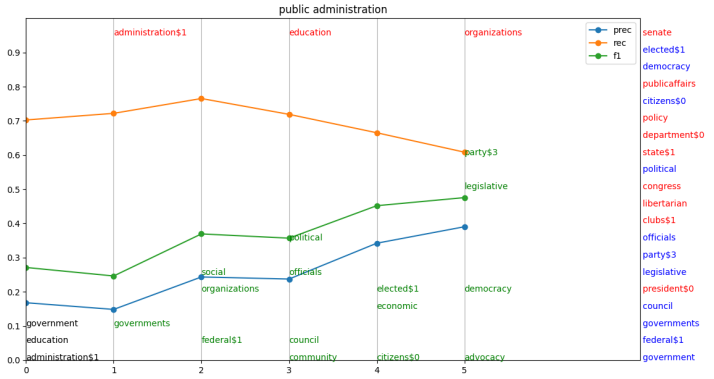


Figure 5.4: Perfomance metrics of class "public administration" over 6 iterations, see figure 5.2 for chart legend.

heterogeneous classes like "public administration". Public administration companies work in many different subsectors and it's much harder to find some ideal keywords that summarize the concept. Because different keywords are needed for each topic the algorithm tends to converge the classification to only one keyword. An easy improvement can be made to mitigate this problem. Starting from the first dictionary expansion, the algorithm chooses the top-3 best keywords that the tf-idf algorithm can find. These words do not necessarily match the ones initially selected by the user, so in the following iterations, the model is allowed to delete the first words added

by the user and continues the dictionary expansion using its own knowledge. As the user knowledge is much more reliable than the tf-idf it is advisable to keep the initial seedwords by making the dictionary expansion strictly non-decreasing.

5.3.2 Model with Three Seedwords and Non-Decreasing Dictionaries

By allowing the model to keep the initial keywords the performance slightly increases. The last iteration achieves 71% macro F1-score and 75%. The model improves the prediction on the "public administration" achieving 63% F1-score. Keeping the initial seedwords pushes up the precision improving consequently also the F1-score. See figure 5.5

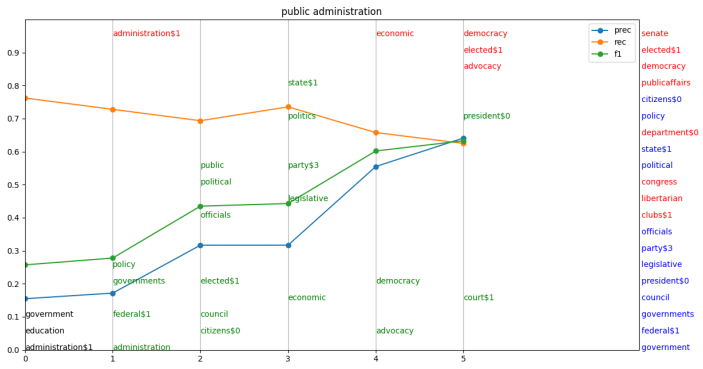


Figure 5.5: Perfomance metrics of class "public administration" over 6 iterations, see figure 5.2 for chart legend.

The same effect can be observed in the "mining" class where by keeping the words "excavation" and "ores" the model slightly improves the F1-score by 3% being able to find 2 more words from the top-20 most category indicative words (listed in blue on the side of chart 5.6.

5.3.3 Full Seedwords

Until now only 3 seedwords were used for each category, in the following experiment, the full seedwords available are used. All the available seedwords chosen in collaboration with my tutor are used. Also, n-gram words are utilized. To allow the model to match

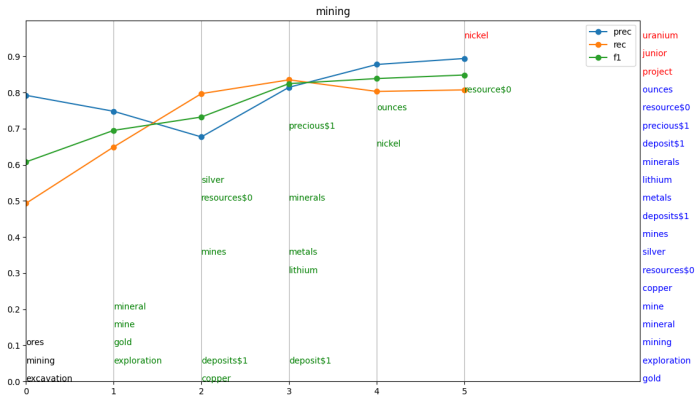


Figure 5.6: Perfomance metrics of class "mining" over 6 iterations, see figure 5.2 for chart legend.

n-gram [1] words a preprocessing step is necessary: the words n-grams are encoded by removing all whitespaces; all occurrence of the n-gram in the corpus is encoded in the same way (for example "public administration" becomes "publicadministration"). This preprocessing step was suggested by Mekala after asking him how to handle n-grams words in private communication. The performance slightly improve with these settings achieving the same macro F1-score of 71% of 5.3.2 and weighted one of 76%. Considering the large number of words added to each dictionary, the results obtained are quite disappointing. Using a richer initial dictionary does not seem to improve the model significantly.

5.3.4 Perfect Dictionary Execution

To test the limits of the Mekala approach we designed an experiment to check the best possible results that the dictionary expansion is able to obtain if the model classified all document in the correct class starting from the first iteration. The tf-idf algorithm that computes the dictionary expansion is limited by how accurate the class predictions are. If a document is incorrectly classified it won't be useful for the tf-idf computation of its own class and will introduce noise to the dictionary expansion of another class. The experiment consists of calculating the best list of keywords by applying the tf-idf

[1] In the fields of computational linguistics, an n-gram is a contiguous sequence of n items from a given sample of text or speech. For example the word "food" is a unigram while "food production" is a bigram and "food and beverages" a trigram.

algorithm assuming that we know the gold class of each document. Considering the top-20 keywords obtained, ranked by relevance, we start training a model using the first best 4 keywords as initial seedwords and we record the precision, recall, and F1-score. At each subsequent iteration, a new keyword is added from the top-20 list and a new model is trained. The model using the top-20 keywords for the training will represent the best possible model obtainable using a dictionary of keywords created by the tf-idf algorithm. It can be observed that the model is not capable of achieving an F1-score higher than 65% for some specific classes like: "mining", "logistics", "construction", "public administration", (see charts at chapter A.3). The F1-score, as shown in figure 5.13 slowly increases during the first 6 executions and stays stable in the next 14 executions, suggesting that each class can be described completely by just 6 words.

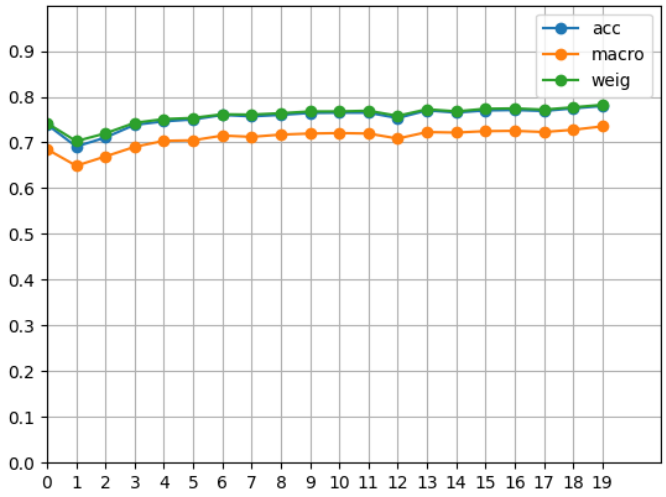


Figure 5.7: Perfomance metrics of ConWea adding one new seedword at each execution and retraining a new model, in blue is the list of all the words used at the last execution.

It can also be observed how the precision and recall values are inversely correlated, an increase in the precision is paired with a reduction of the recall. This is mainly caused by the pseudolabel assigning function 4.3. A model that bases its knowledge only on keywords is affected by a significant level of noise: if a new keyword improves the precision, most of the times it also makes the model converge the classification to

a small subset of all the possible class instances, reducing the recall.
Overall the best macro F1-score achievable (by using the entire list of the top-20 keywords) is 73%, while the weighted F1-score is 77%.
This will be useful as a baseline for future model comparisons.

5.4 LOTClass Experiments

The experiment with the Meng algorithm requires some initial set up and have the goal of assessing the importance of the name of the category when calculating the category vocabulary. The author stated that not more than 3 words were used for each label name and made some changes to the category names to make them useful for the model. Some modification to the code are required: in some configurations the category vocabulary of some classes created contains less than 50 words, this is caused by the removal of stopwords and words shared by multiple categories. For this reason using a fixed threshold of 20 matching words needs to be change to a variable threshold. The threshold has been converted into a percentage of the vocabulary size of each category. The percentage of word required to be matched is set to 40% (20/50*100). The execution of each experiment takes approximately 40 minutes, using a machine offered by the Colab Pro subscription: 2 vCPU, 24 GB RAM, NVIDIA P100 GPU.

5.4.1 Simple Label Names

A simple approach is to use the original label names with some specific changes. Setting the name of our classes is not trivial: some names are composed of multiple words and cannot be used directly by the BERT language model.

Here is how the names have been converted:

- Agriculture: agriculture
- Buildings: buildings
- Construction: construction
- Energy: energy
- Food & Beverages: food, beverages
- Healthcare: healthcare

- Mining: mining
- Financial Services: finance
- Manufacturing: manufacturing
- Public Administration: government
- Transportation: transportation
- Utilities (electricity, water, waste): electricity, water, waste

Only 2 classes need their name changed. Considering that the language model has only knowledge of unigram words, multi-word labels need to be converted appropriately. The words "public" and "administration" do not singularly give enough information on the category it was decided to replace them with a single word bringing much more information: "government". Food and beverages instead are pretty self-explanatory so they can be used singularly to build the category vocabulary. The class “utilities” since it’s not a name it is converted using the subcategories names inside the parentheses. The final macro F1-score is 59% while the weighted F1-score is 64%. The Self-Training task improved the initial accuracy of the model being trained only on the MCP task by 7%. The confusion matrix is shown at 5.8. In the confusion matrix each row represents the instances in an actual class while each column represents the instances in a predicted class. The sum of all the values in each row gives the total amount of samples for that specific class, the sum of all the values in each columns gives the total predictions for the corresponding column class. The results of a good model would be distributed only on the diagonal axis, meaning that the true class correspond to the predicted class. If the predictions of a class are scattered over all the row it means that the recall for that class is very low (a lot of False Negatives), if the predictions are scattered over a column it means that the model has low precision (a lot of False Positives). In this matrix we can observe that “manufacturing” instances have very low recall (the values are scattered on the 8-row) with most predictions being as “mining” (row 8, column 9). At the same time most of “mining” instances are predicted as “manufacturing” (row 9, column 8).
“financial services” and “healthcare” achieve great results, they appear as white cells because they are the most numerous classes, consisting in more than 5 thousand correctly classified samples each.

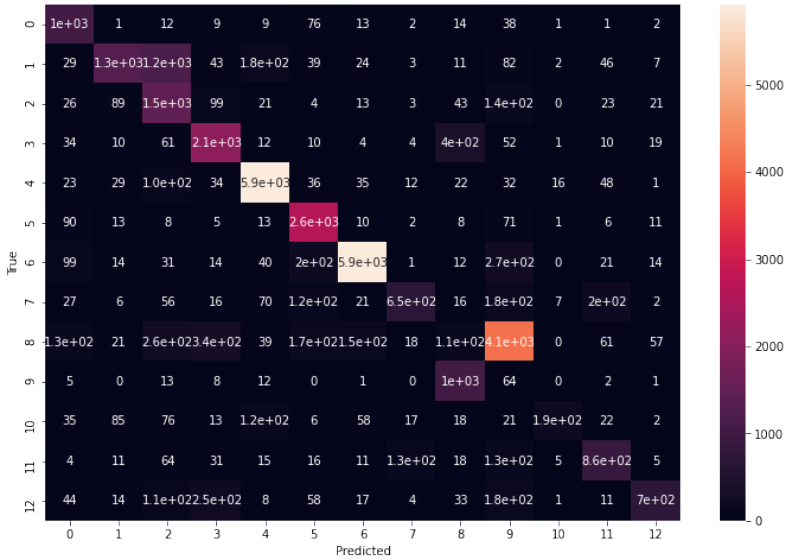


Figure 5.8: Confusion Matrix using Simple label names, each class is linked to a number in alphabetic order: 0: Agriculture, 1: Buildings, 2: Constructions, 3: Energy, 4: Financial services, 5: Food & Beverages, 6: Healthcare, 7: Logistics, 8: Manufacturing, 9: Mining, 10: Public Administration, 11: Transportation, 12: Utilities (electricity, water, waste)

An additional observation can be made. The model is not able to distinguish between "buildings" and "construction", this is obvious as both words are synonyms, even a human interpreter cannot understand what each category identifies. In this context "buildings" identifies economic activities related to the use of buildings (rentals, sales) while "construction" deals with the construction of buildings per se. These two classes have relative small vocabularies because most of the words are shared between the two vocabularies and deleted from both.

This analysis is only possible thanks to the confusion matrix, which makes visible where the classifications converge; looking just at the precision and recall values of individual classes is not helpful in understanding what the problems in a given model are.

The problem related to class ambiguity has to be dealt with by giving more information to the model. A solution is proposed in the next chapter.

5.4.2 Multiple Label Names

While doing some experiments on the label names, it has been hypothesised how having multiple words as label names could improve the model accuracy. So it was decided to use instead of the label names, the dictionary made of 3 keywords as a label name. This approach also skips the label name refactoring required in 5.4.1 making the implementation more straightforward. Similarly to the original algorithm if a list of words is used as label name each word singularly contributes to the creation of the category vocabulary.
In the case of the "public Administration" category, using three words ("government", "administration", "education") for the MCP task (see 4.4.1), creates a faulty category vocabulary that is not able to match any category indicative word in the corpus. The algorithm throws a warning when this problem happens and prompts the user to fix this problem. For this reason, only the word "government" is used as a label name for the class "public administration". This solution anyway does not help the model and it completely fails to predict "government" with most predictions converging into the "energy" class. The MCP task finds 9886 documents each containing at least one category indicative term. The final macro F1-score is 59% while the final weighted F1-score is 65%. This approach using the 3 seedwords as label names proves to be equal to LOTClass using only standard label names, some classes that previously were hard to classify like "buildings" and "construction" now have a more clear distinction.

The followings are the vocabularies for these two classes:

- "buildings": [building, property, properties, house, housing, estate, residence, lot, title, floor, dwelling, plan, personal, ground, rent, equity, asset, flat, condo, hotel, park, let, tower, cost, territory, roof, luxury, accommodation, facility, value, landlord, lease]

- "construction": [construction, constructions, completion, erected, completed, renovation, steel, concrete, bridge, repair, restoration, construct, architecture, masonry, designed, erection, structures, finished, stone, contractor, structural, finish, additional, demolition, added, architects, planned, plumbing, contracting, excavation, architect, temporary, installation, seismic]

The length of the vocabulary is quite short (32 and 34 words respectively) compared

to the length of 50 suggested by Meng as a lot of words are shared and for this reason deleted, but it can be observed how the vocabularies are descriptive of their own class. On the other side the new label name for “manufacturing” becomes too generic and a lot of predictions converge into it drastically reducing the precision. Also the class Transportation does not get predicted correctly (row 8, column 4)

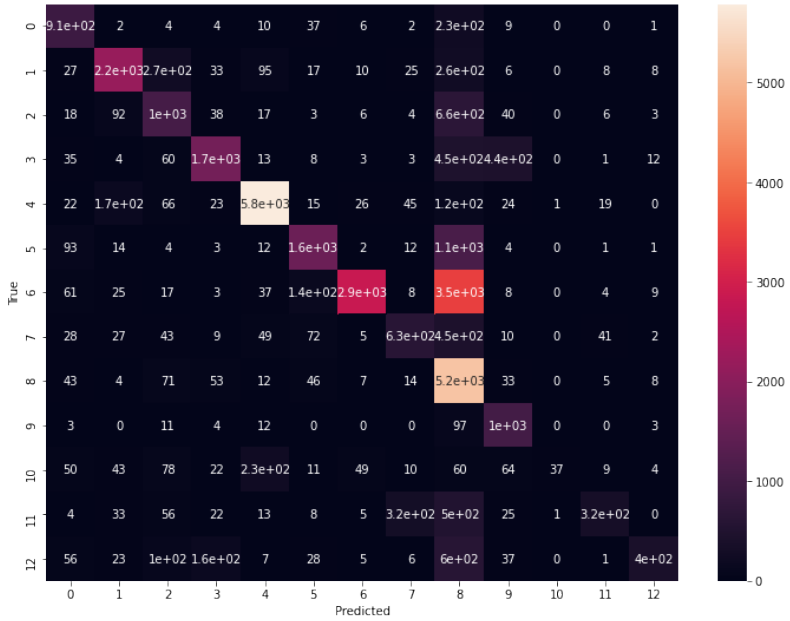


Figure 5.9: Confusion Matrix of LOTClass results using 3 words as label names, check 5.8 for chart legend.

5.4.3 Using Stricter Threshold for "Manufacturing" Class

The last experiment results show how “manufacturing” is hard to deal with. This problem also appeared when trying to use ConWea: the word "products" considered by the tf-idf algorithm as an important keyword for the "manufacturing" class adds a considerable noise to the pseudolabel generation that makes the model converge into classifying all documents into that specific class. This problem of the classification converging onto a single label also appears here.

For this reason, a final experiment is designed to check how changing the threshold could improve the classification for specific classes. The LOTClass algorithm is modified so that it uses a stricter threshold. To consider a word category indicative for the class "manufacturing" it needs to match 55% of the vocabulary words instead of 40%. The

MCP task manages to find only 78 category indicative words for "manufacturing" and the final trained model completely fails to classify any document with this class. The macro F1-score is 57% and the weighted F1-score is 61%.

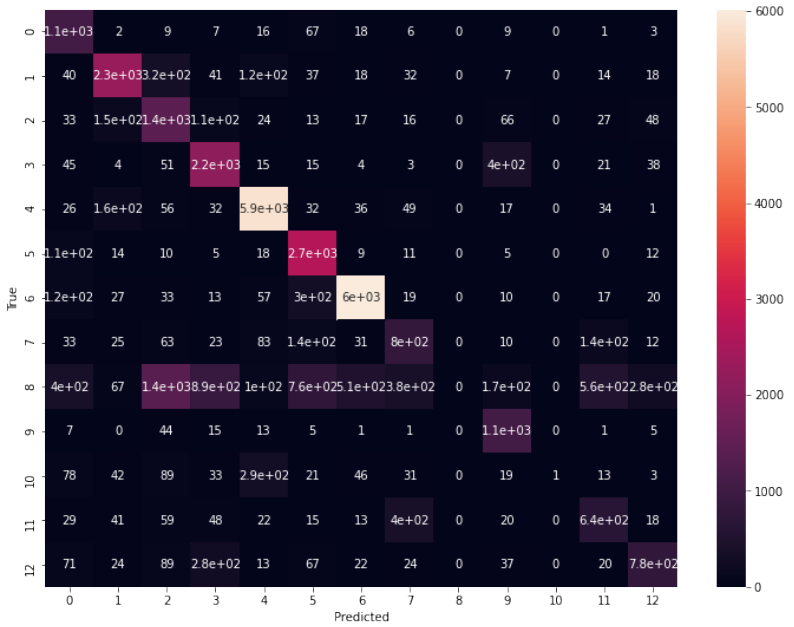


Figure 5.10: Results using a 55% thresh for Manufacturing, check figure 5.8 for chart legend

This experiment shows the limits of the LOTClass approach. Small changes in the threshold completely change the model output, handling complex classes is not feasible by only using label names information. Overall the LOTClass model has lower performance than ConWea (10% lower F1-score) however, it introduces an innovative method of classifying data. The training happens at word level and then gets propagated to the document level via Self-Training. This configuration allows the Self-Training process to be easily converted into a task that can be trained together with the MCP task leveraging both document level and word level information. Different information sources can be used to train each task and this is the idea behind the Joint Task approach proposed in 4.5 and that will be analized in the next chapter.

5.5 Joint Task Experiments

5.5.1 Joint Task Standard

The join task is trained using the 3 keywords dictionary as label names, (the same as in experiment 5.4.2) and the full dictionary for word count [CLS] labeling (see section 5.5.1. This task requires specific modifications to the pipeline and data handling. As the Bert tokenizer splits unknown words into subwords it is necessary to create 2 versions of each document one where the text is encoded with the Bert tokenizer and one with a classic tokenizer. The text encoded with a classic tokenizer is also preprocessed so that n-grams appearing in the seedwords are encoded by removing the whitespaces (e.g. "real estate" becomes "realestate"). This allows also to match n-grams seedwords when labeling the [CLS] token.
The Masked Category Prediction Task is able to find 9886 documents containing category indicative words, the word count task finds instead 26127 documents. After removing 5212 thousand documents that share both MCP and word count labelling from the word count task we obtain a dataset of 30801 documents, corresponding to 85% of all available documents. The MCP task is preferred over the word count task because it focus the model attention on specific relevant parts of the document while the word count task is not able to do so.

In this configuration the Self-training step is not able to further improve the model performance (it does not reduce it either) and can be removed from the pipeline. The improvement over the standard LOTClass is considerable. It achieves 67% macro F1-score and 74% weighted F1-score, improving by 8% and 0,34% respectively, with a relative error reduction of 27% and 13,8% each. Compared to the standard ConWea this model performs 2% worse in the macro F1-score but same in the weighted F1-score, achieving overall similar results in less training time. The results shows that the model further improved over ConWea in the classification of classes with a lot of samples like "healthcare" (row 6, column 6) and "financial services" (row 4, column 4) but perform worse in classifying low samples classes. As can be seen in row 8 of matrix 5.11, the precision of most classes is compromised by the elements of the "manufacturing" class failing to be classified correctly and being distributed among the various classes. The precision of the "mining" (column 10) class is pretty low while the recall is quite high (row 10).

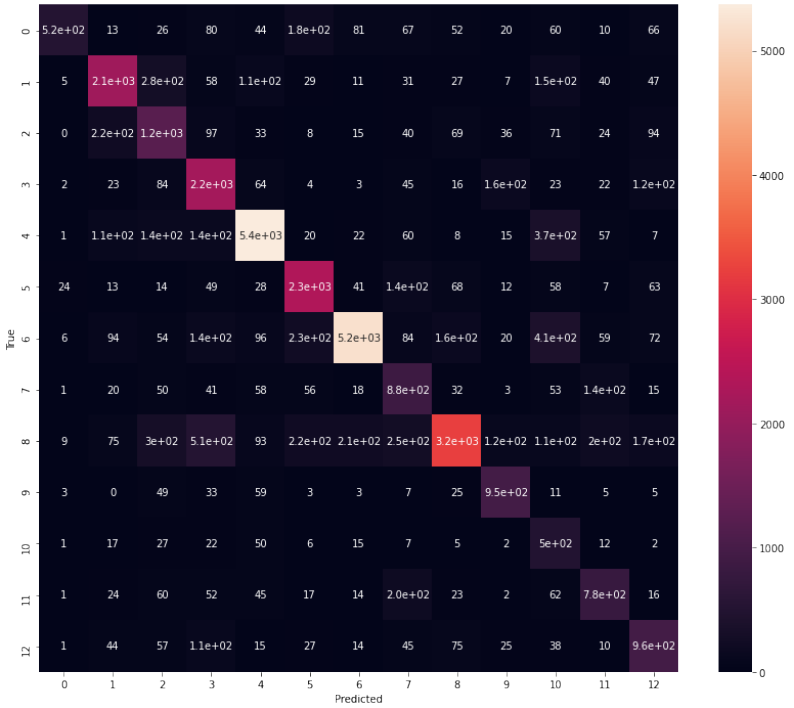


Figure 5.11: Confusion Matrix of the joint Task, check figure 5.8 for chart legend

5.5.2 Training Only on [CLS] Token

To be sure that the combination of the 2 tasks is helping to improve the model performance we need to assess that both tasks (training on CLS and MCP) are useful. We have seen that the joint task performs better than the standard LOTClass model; now we need to assess that it also performs better than just training on the [CLS] token. As we can see in the confusion matrix 5.12 and by comparing the F1-scores this model performs worse.

By Comparing confusion matrix 5.12 and 5.11 we can observe how the joint task improves the performance of the model based only on the [CLS] task. The [CLS] task completely fails to predict the Agriculture class; instead the joint task achieves 91% precision and 43% recall. Small improvements are register in most precision and recall values. The model struggles to classify the "manufacturing" (row 8) and transportation, having very low recall but discrete precision. This configuration achieves 60% macro F1-score score and 69% weighted F1-score score. Meaning that the Joint task improves the macro value by 7% and the weighted value by 5%.

TODO add comparison precision recall of all classes.

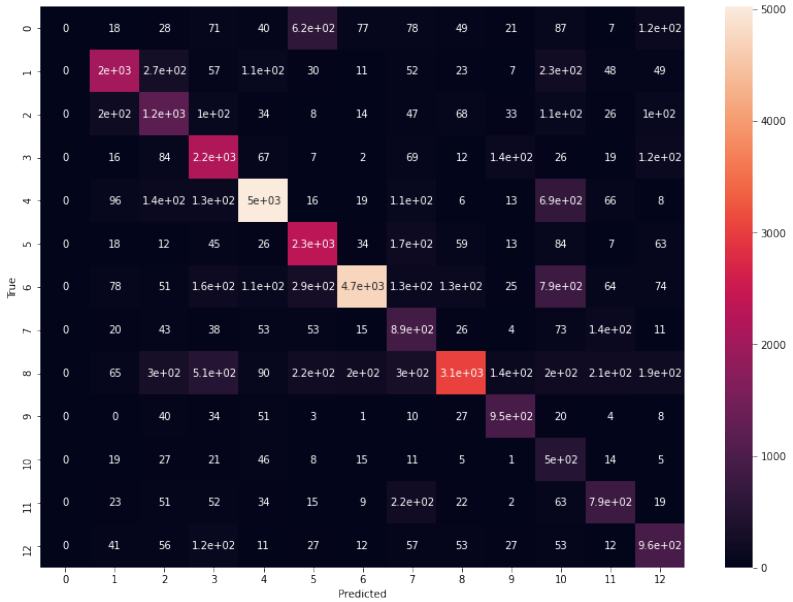


Figure 5.12: Confusion Matrix of the CLS based task, check figure 5.8 for chart legend

5.5.3 Joint Task Using Less Noisy Data

The function 4.3 used to assign the pseudolabels to the [CLS] token is very simple and is subjected to noise in the documents. For example, sometimes the business company's description talks about other business sectors not related to the company itself. This behavior is hard to control and is not easy to identify the documents responsible for this problem. The argmax function simply takes the class matching the highest number of seedwords: if only one seedword is found the heuristic assigns that class to the [CLS] token and this is prone to error. This heuristic can assign a class to the [CLS] tokens of more than 26 thousand documents over the 36 thousand available. Considering that a lot of seedwords were made available by my supervisor, a better approach could be in adding a threshold so that only documents matching at least 2 seedwords of the same class are taken into consideration during the training. This method creates a smaller and less noisy dataset that achieves remarkable results combined with the MCP task. While the MCP task finds 9886 documents containing category indicative words, the word count task finds 9231 labeled documents. A subset of documents shares double labeling of both the MCP task and the word count task and are treated by removing the [CLS] labeling (only keeping the MCP assigned labels). Figure 5.13 shows the confusion matrix. The class "manufacturing" (row 8, column 8)

and "mining" (row 10, column 10) that previously failed during the classification now get better results.

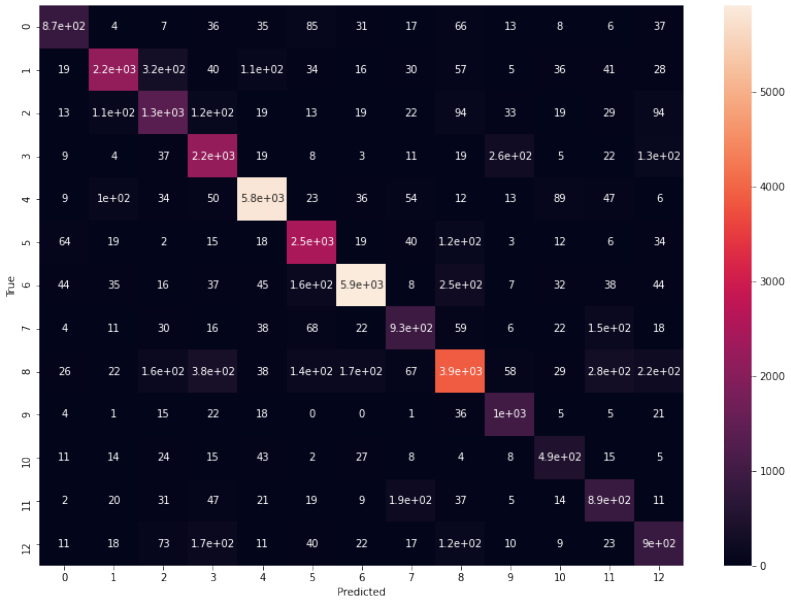


Figure 5.13: Confusion Matrix of the Joint Training Task

The results are better than the one achievable using the best possible dictionary with ConWea 5.3.4; the macro F1-score is 75% and the weighted F1-score: 81%.
The model trained only on the [CLS] token using this denoising approach achieves 67 % macro F1-score and 75 % weighted F1-score. The joint task is able to improve by 8% the macro F1-score and by 6% the weighted F1-score, corresponding to a relative error reduction of 24% for both measures.
The results of the model trained only on the [CLS] token are equal to the one achieved by the joint task of 5.5.1 and hint that maybe the problem with the data available is an abundance of seedwords appearing in the wrong document class. The evaluations show that the proposed variant of LOTClass, trained on a joint task using an external dictionary and label names, improves the performance of weak supervised text classification models when dealing with complex data classes, while not introducing additional costs to the inference.

The final results of the experiments are shown in table 5.14.

Model	Version	macro F1-score	weighted F1-score
ConWea	3seed	69%	74%
	3seedNonDescending	71%	75%
	FullSeedwords	71%	76%
	PerfectDict	73%	77%
LOTClass	standard	59%	64%
	seedwordsAsLabels	59%	65%
	55%thresh	57%	61%
Only Cls	argmax-word-count	60%	69%
	double-match	67%	75%
Joint Task	argmax-word-count	67%	74%
	double-match	**75**%	**81**%

Figure 5.14: Table comparing results of all experiments done.

Chapter 6

Conclusions and Future Work

In the thesis that is about to be concluded, we have set out the research, design, and implementation operations that were necessary for the implementation of a system capable of classifying companies into business sectors. The aim of this thesis was twofold. On the one hand, it served to present some existing literature concerning the field of Natural Language Processing and how Deep Learning models can be useful for solving word embedding and text classification tasks in the absence of categorized data. On the other hand, it showed how the models presented can be applied in solving a real task using a specially constructed dataset. Initially, the theoretical arguments at the basis of the project were reviewed, starting from the fundamentals of Machine Learning such as classification, Naive Bayes, and other algorithms, through to more advanced topics such as Natural Language Processing and numerical representation, with particular attention to the techniques of Word Embedding. We introduced some Weak Supervision techniques and we analyzed the performance on the available dataset of the internship company. The focus was on the weaknesses and strengths of each algorithm. Starting from ConWea, suffering from the intensive computations required and lack of stability on more complex tasks but achieving state-of-the-art performance we shift our focus on LOTClass, innovative algorithm that require less initial information but suffer from poor performances on complex datasets and classes. In the end, a new training technique is proposed that exploits different sources of weakly supervised to jointly train a model with enhanced performance. The multiple sources of weak supervision are both an heuristic using seed words to assign pseudo-labels and the "category vocabulary" introduced by LOTClass. The Joint training shows better

results than LOTClass on this type of dataset. The category works well when each class features are similar but struggle in handling macro-categories containing a lot of different instances. This new approach can leverage multiple sources of information starting from the same set of keywords, further work could show how this method can be applied also on different datasets. The new Joint Training approach requires further investigations. First, it must be assessed if the performance is comparably good even on different datasets. These experiments are not easy to compare as a lot of variables can influence the performance of the model, the initial keyword chosen by the human agent can affect unpredictably the model performances. In this implementation of the Joint Task training only the BERT pretrained version, dated 2018, was used. In the previous years, new improved versions have emerged like Roberta [Zhi+19] and XLNet [Yin+19], it can be interesting to test if these new models can further improve the Language Model understanding of each class and enhance the classification performance of the model. Further work could test if the seed expansion of ConWea could improve the Joint Task Training. The joint training proposed does not use the word contextualization like ConWea but it is possible to add the K-Means clustering step at the start of the process. Care must be taken to use two different versions of the documents, one contextualized that is processed by the seed words heuristics and the original one being processed by the MCP task.

The project was carried out with ambition and passion for the covered topics. Despite of all the various difficulties that new approaches usually demand, I am proud about the final result. It is the outcome of a long journey, not just metaphorically speaking. While the embryonic idea was developed in Italy, it was further implemented by working and studying in Portugal and Finland. I thereby declare myself satisfied from the overall enriching experience of writing this experimental thesis.

Appendix A

Appendix

A.1 3 Words Dictionaries

Here we report the result using the simple dictionaries composed of 3 keywords for each class, available as cited in chapter 4. As usual we report in blue the precision, in orange the recall and in green the F1-score, on the right we see the best keywords that the model could find: in red the one it misses and in blue the one it finds. In black at the start are shown the initial dictionaries and in green/red the words added/deleted.

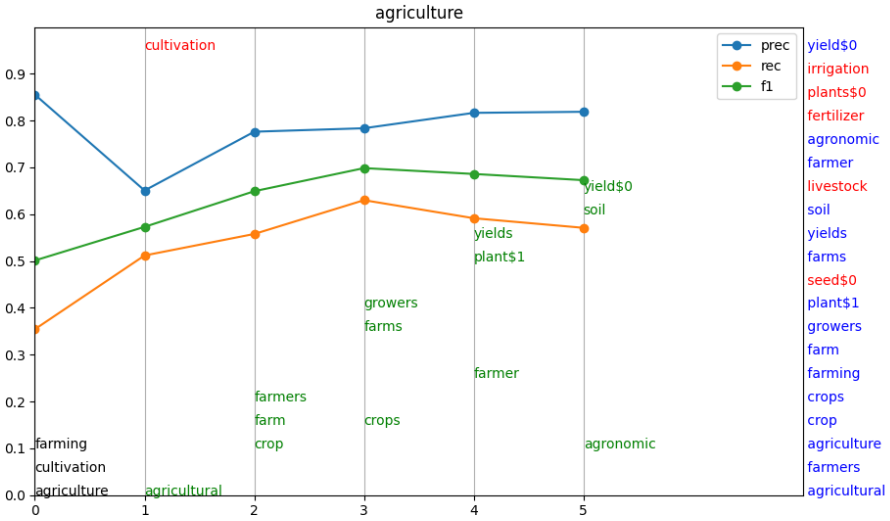


Figure A.1: Precision, Recall and F1-score values over 6 iterations of the Agriculture class.

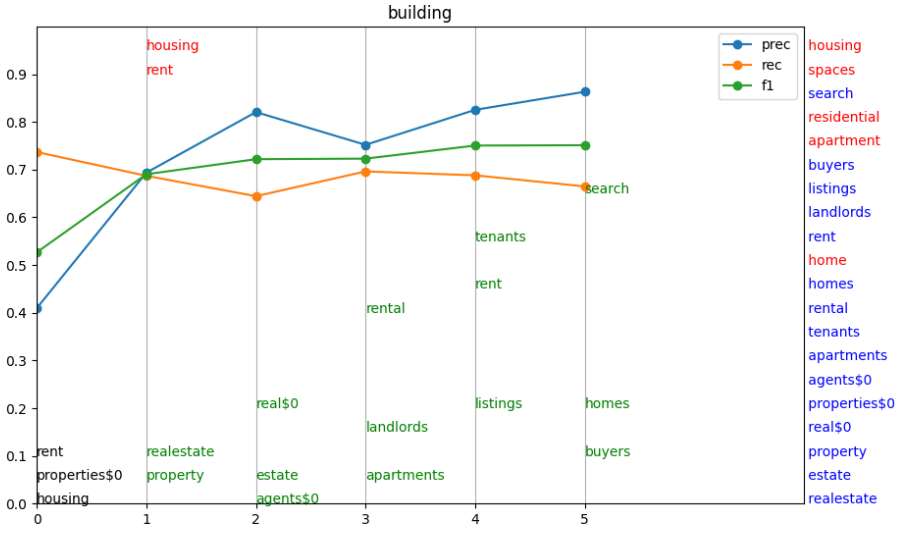


Figure A.2: Precision, Recall and F1-score values over 6 iterations of the Building class.

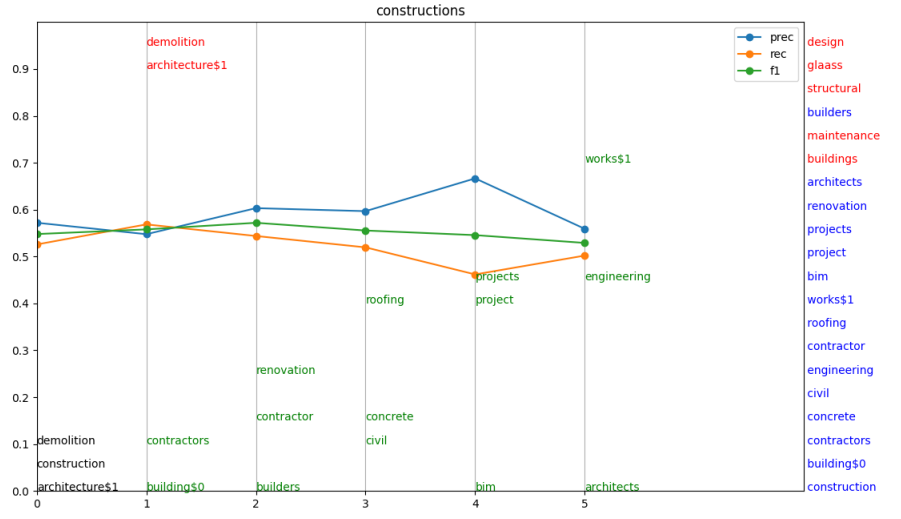


Figure A.3: Precision, Recall and F1-score values over 6 iterations of the Constructions class.

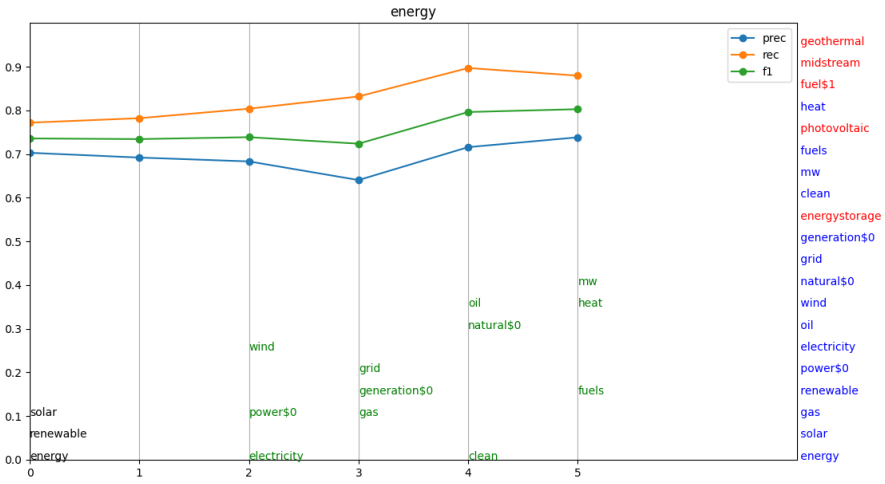


Figure A.4: Precision, Recall and F1-score values over 6 iterations of the Energy class.

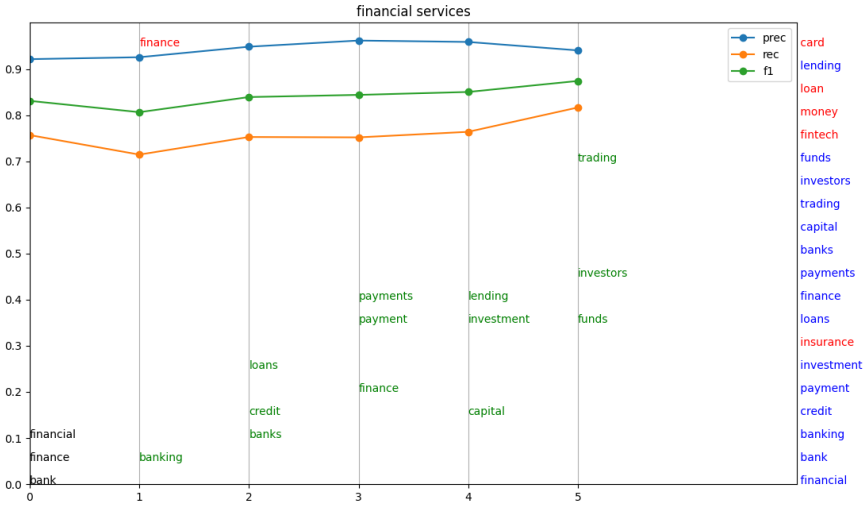


Figure A.5: Precision, Recall and F1-score values over 6 iterations of the Financial Services class.

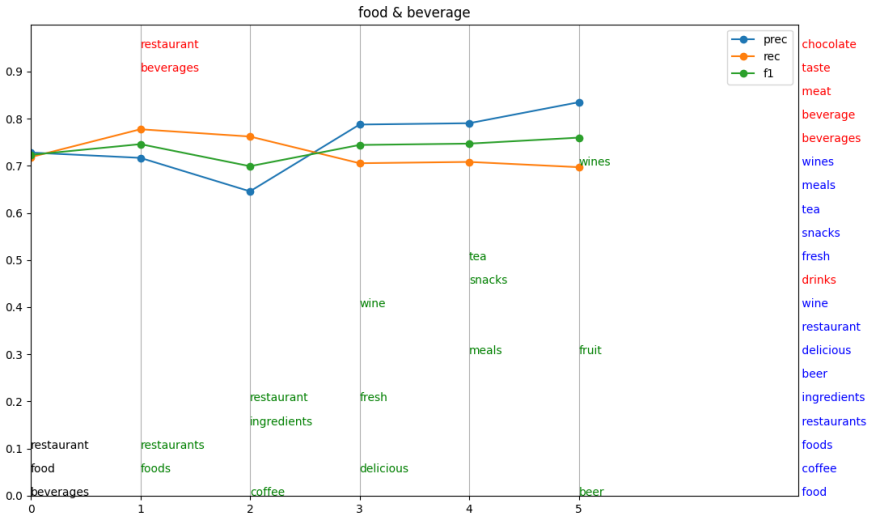


Figure A.6: Precision, Recall and F1-score values over 6 iterations of the Food & Beverages class.

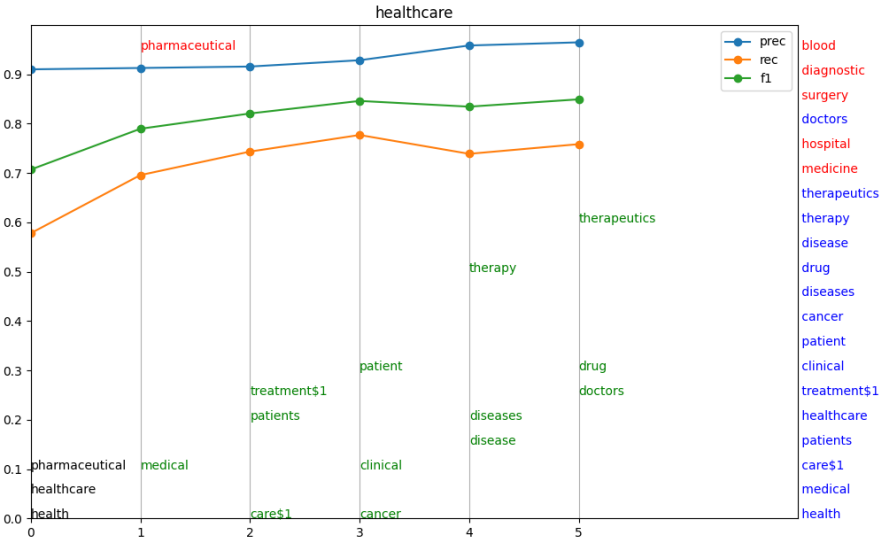


Figure A.7: Precision, Recall and F1-score values over 6 iterations of the Healthcare class.

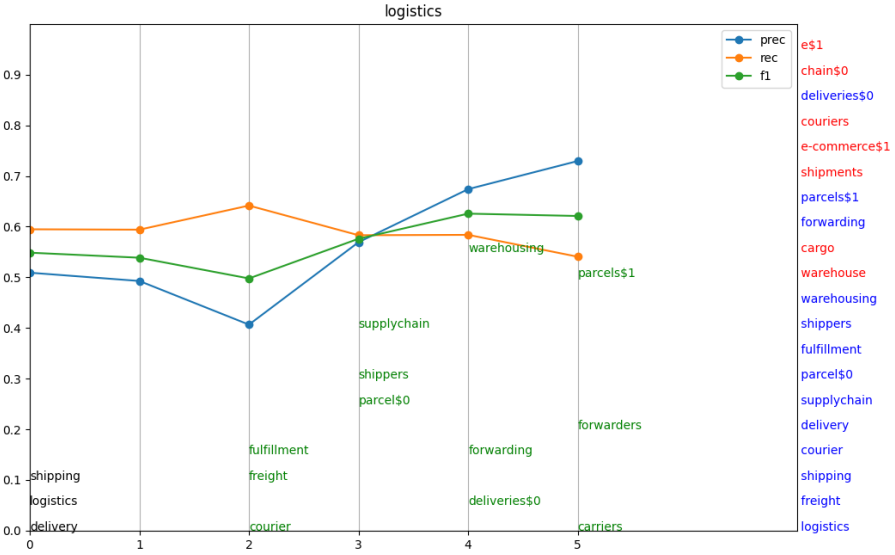


Figure A.8: Precision, Recall and F1-score values over 6 iterations of the Logistics class.

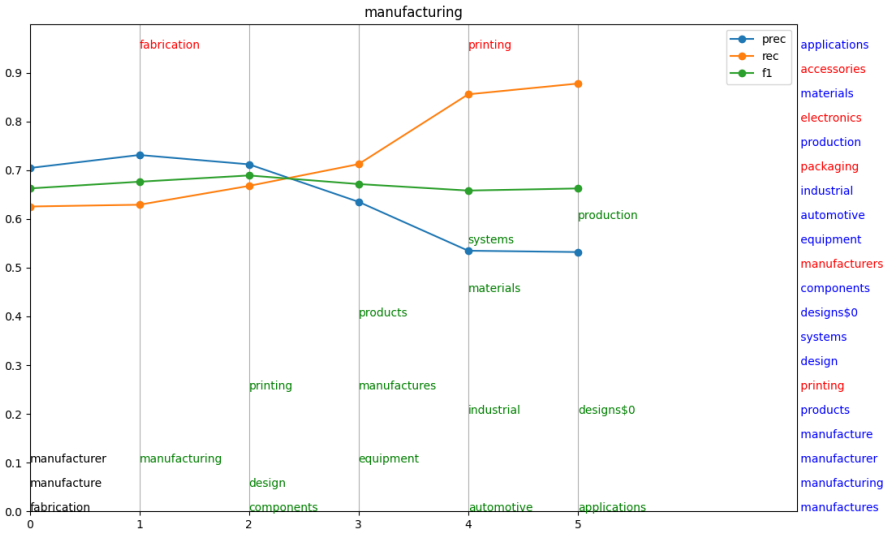


Figure A.9: Precision, Recall and F1-score values over 6 iterations of the Manufacturing class.

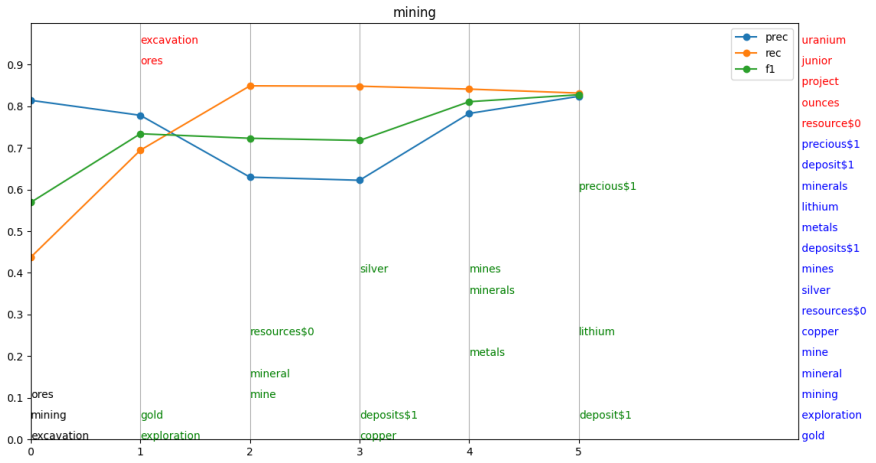


Figure A.10: Precision, Recall and F1-score values over 6 iterations of the Mining class.

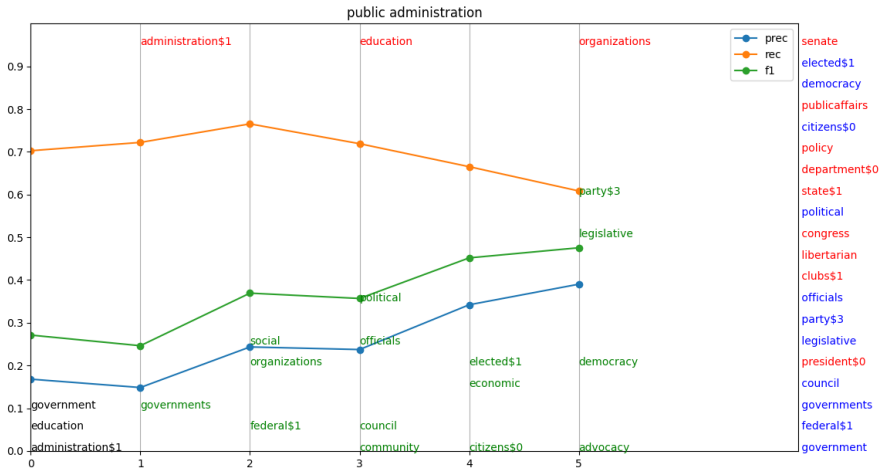


Figure A.11: Precision, Recall and F1-score values over 6 iterations of the Public Administration class.

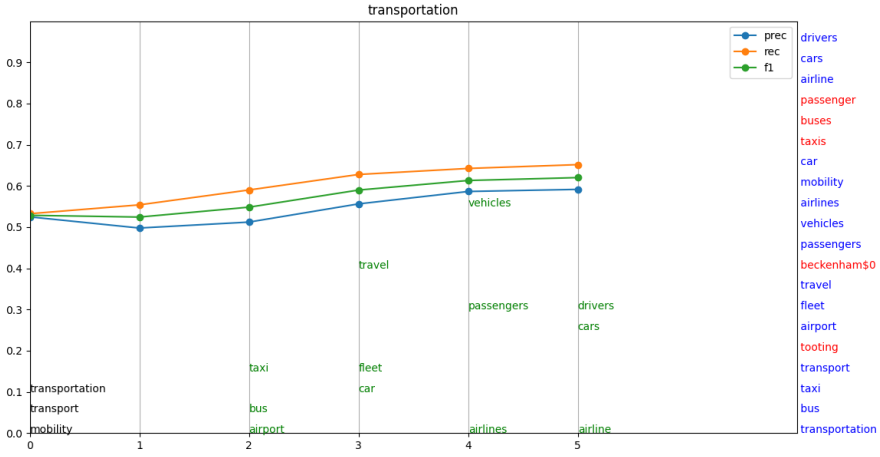


Figure A.12: Precision, Recall and F1-score values over 6 iterations of the Transportation class.

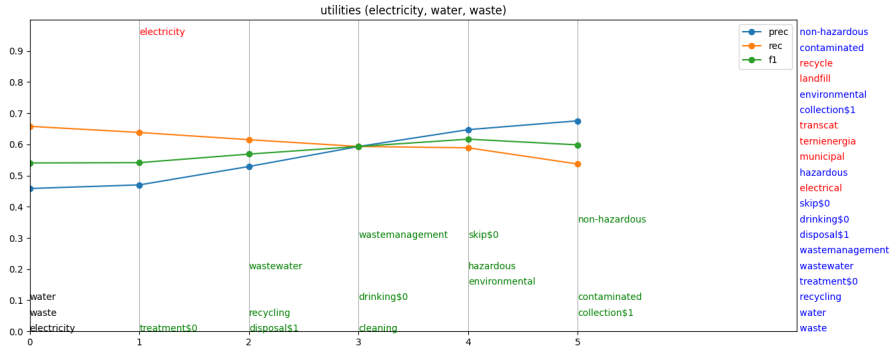


Figure A.13: Precision, Recall and F1-score values over 6 iterations of the Utilities class.

A.2 Full seedwords dictionaries

Here we report the result using the full dictionaries available as cited in chapter 4. As usual we report in blue the precision, in orange the recall and in green the F1-score, on the right we see the best keywords that the model could find: in red the one it misses and in blue the one it finds. In black at the start are shown the initial dictionaries and in green/red the words added/deleted.

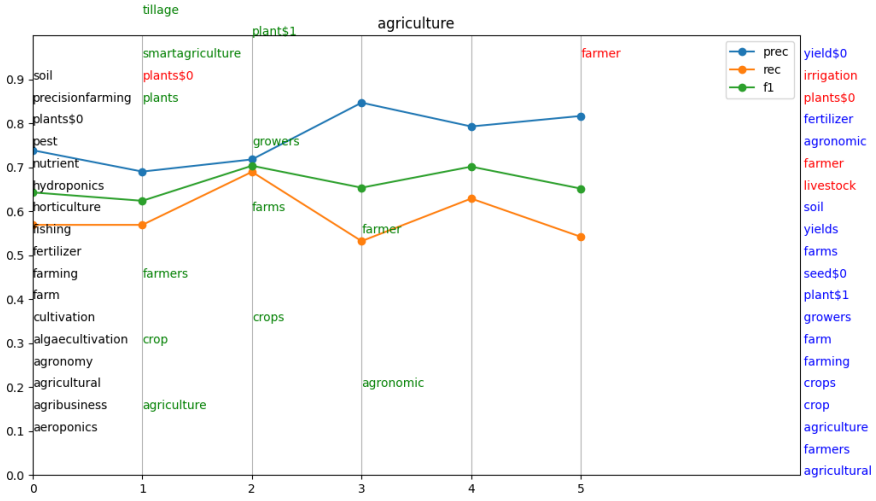


Figure A.14: Precision, Recall and F1-score values over 6 iterations of the Agriculture class.

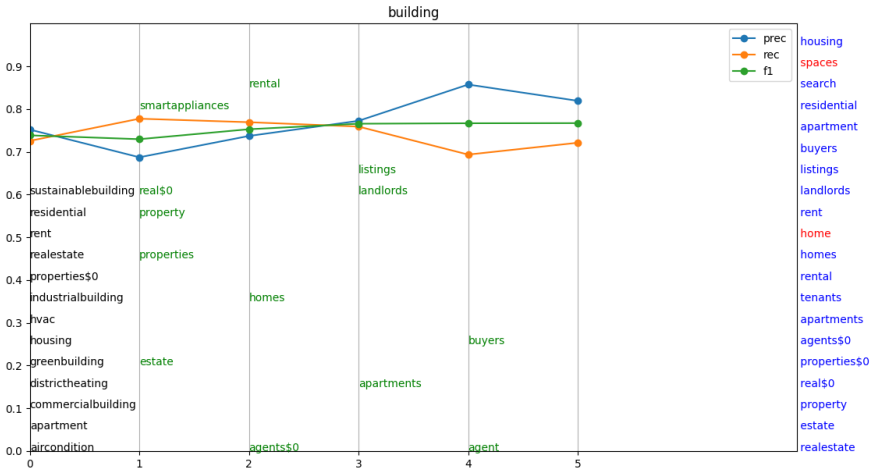


Figure A.15: Precision, Recall and F1-score values over 6 iterations of the Building class.

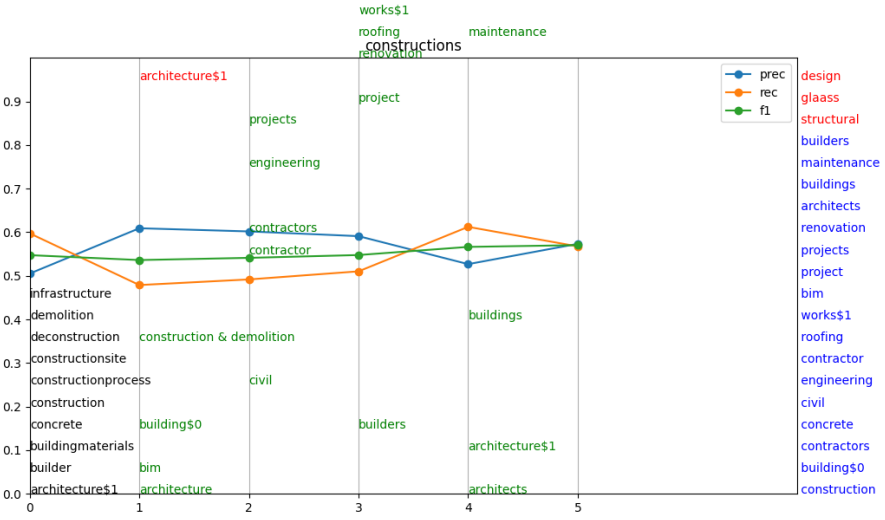


Figure A.16: Precision, Recall and F1-score values over 6 iterations of the Constructions class.

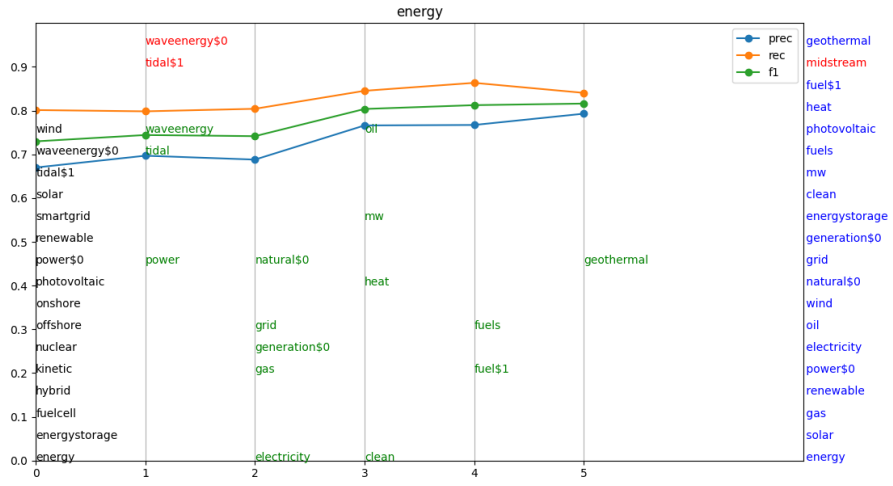


Figure A.17: Precision, Recall and F1-score values over 6 iterations of the Energy class.

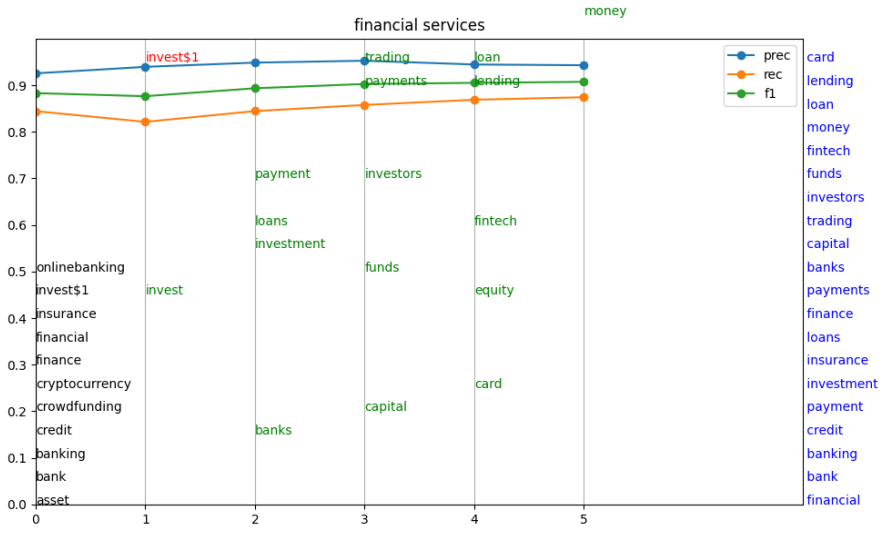


Figure A.18: Precision, Recall and F1-score values over 6 iterations of the Financial Services class.

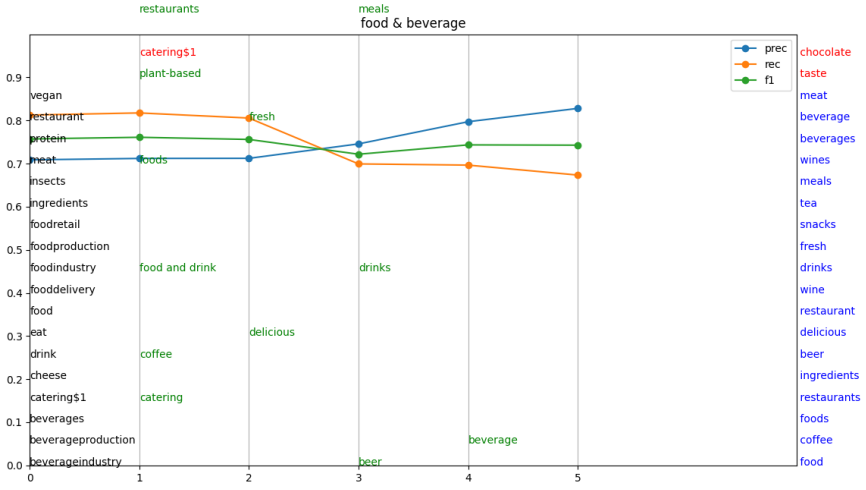


Figure A.19: Precision, Recall and F1-score values over 6 iterations of the Food & Beverages class.

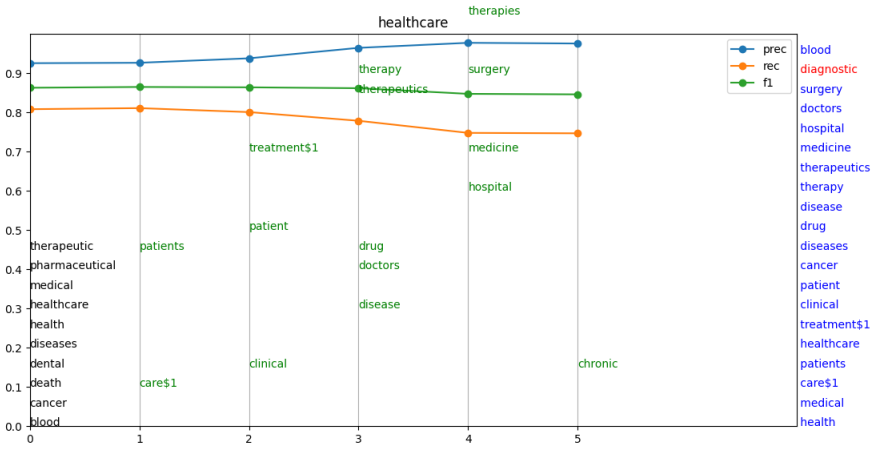


Figure A.20: Precision, Recall and F1-score values over 6 iterations of the Healthcare class.

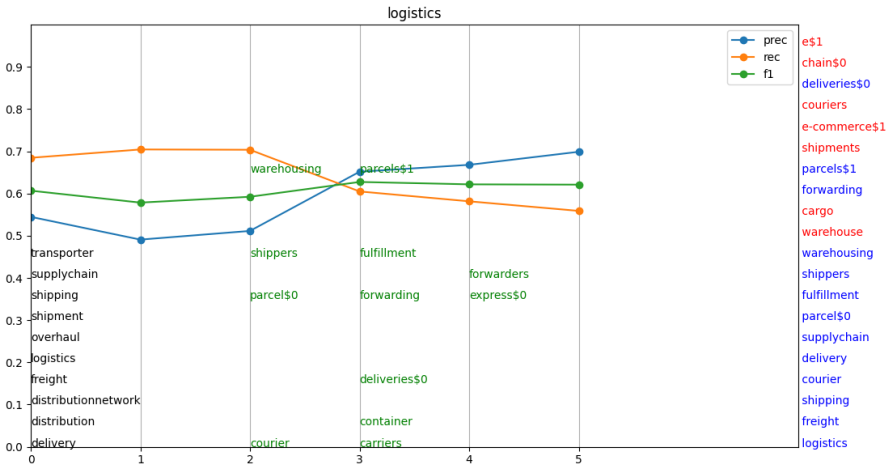


Figure A.21: Precision, Recall and F1-score values over 6 iterations of the Logistics class.

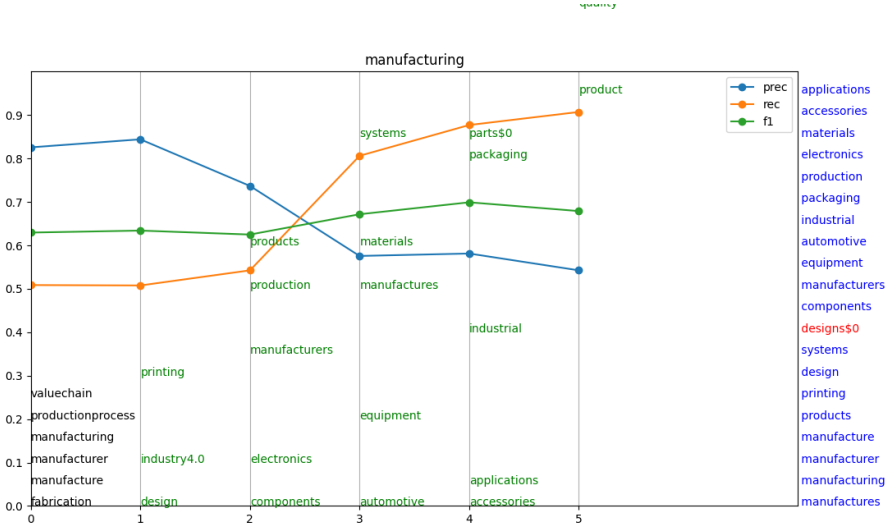


Figure A.22: Precision, Recall and F1-score values over 6 iterations of the Manufacturing class.

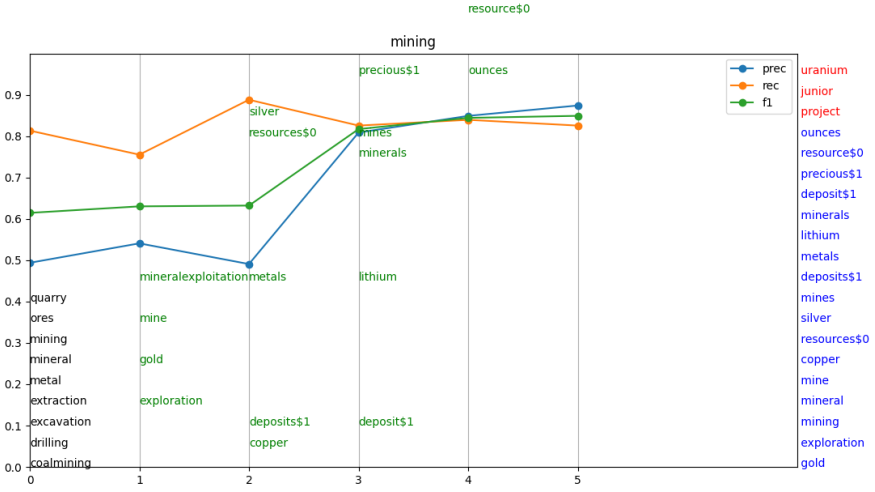


Figure A.23: Precision, Recall and F1-score values over 6 iterations of the Mining class.

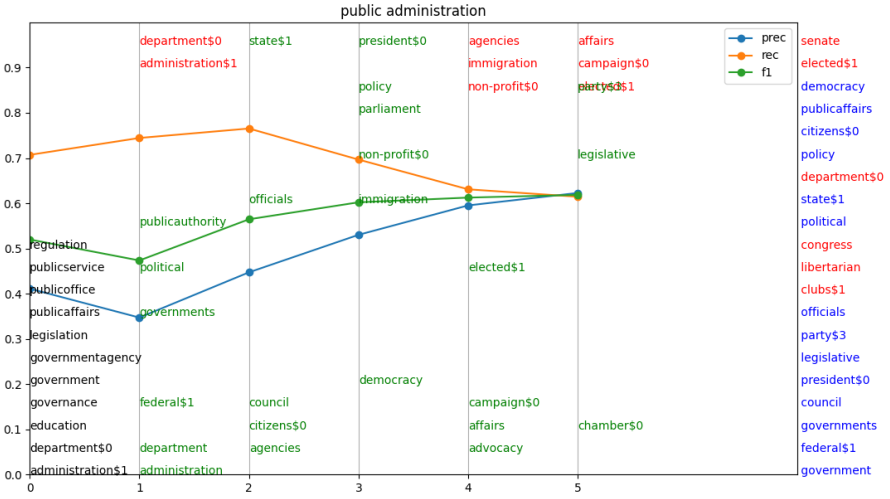


Figure A.24: Precision, Recall and F1-score values over 6 iterations of the Public Administration class.

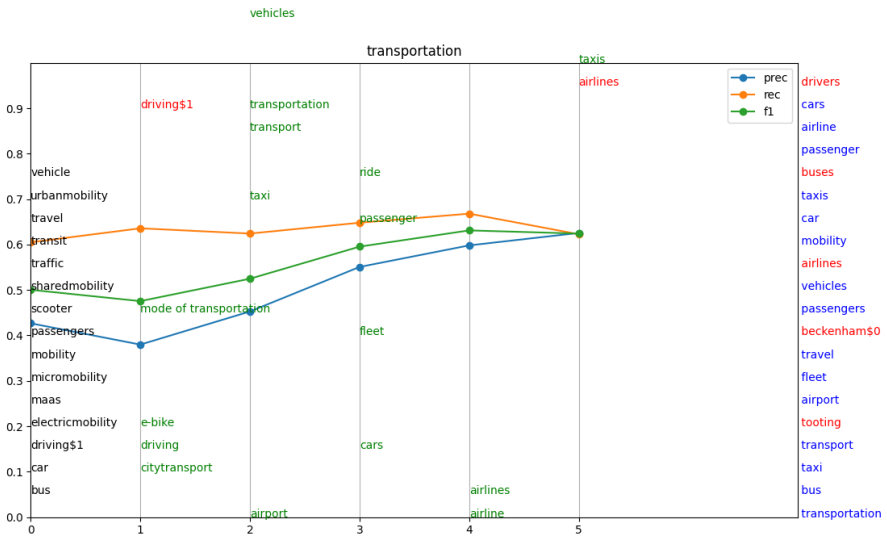


Figure A.25: Precision, Recall and F1-score values over 6 iterations of the Transportation class.

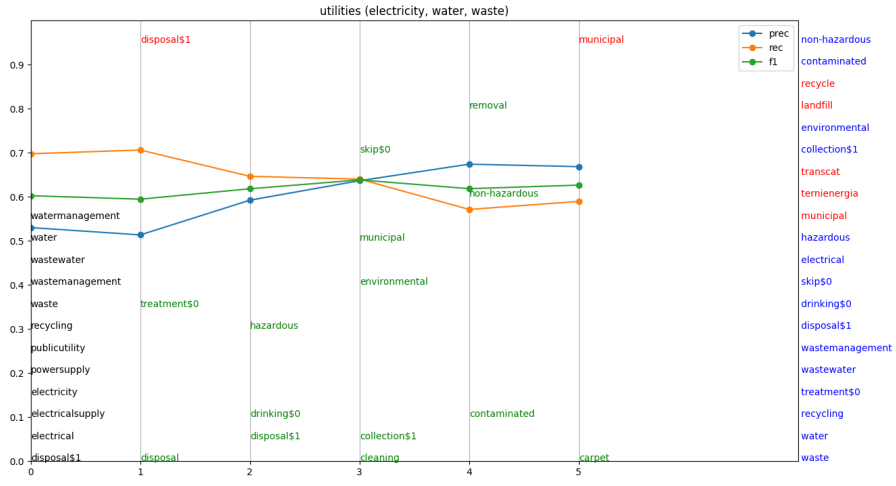


Figure A.26: Precision, Recall and F1-score values over 6 iterations of the Utilities class.

A.3 Perfect dictionary Expansion

These graphs are obtained by training 20 different models, adding a new seedword from the best top-20 each time. The blue line is the precision, the orange one the recall and the green one the f1-score. One the side in blue there is the list of the 20 best seedowords (best at the bottom, worst at the top) using the tf-idf euristich. In green are the words added at each iteration.

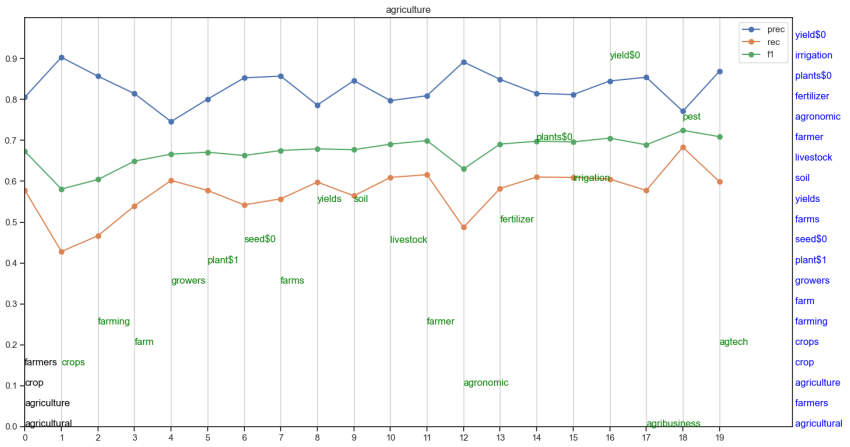


Figure A.27: Precision, Recall and F1-score values of 20 models of the Utilities class.

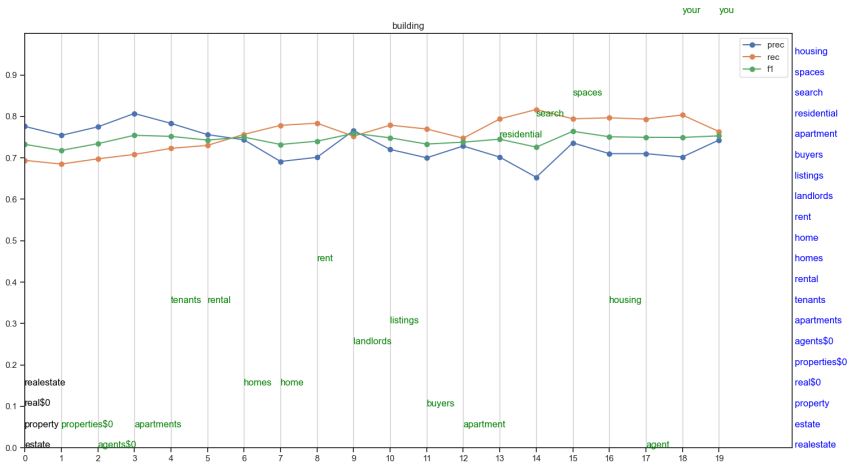


Figure A.28: Precision, Recall and F1-score values of 20 models of the Agriculture class.

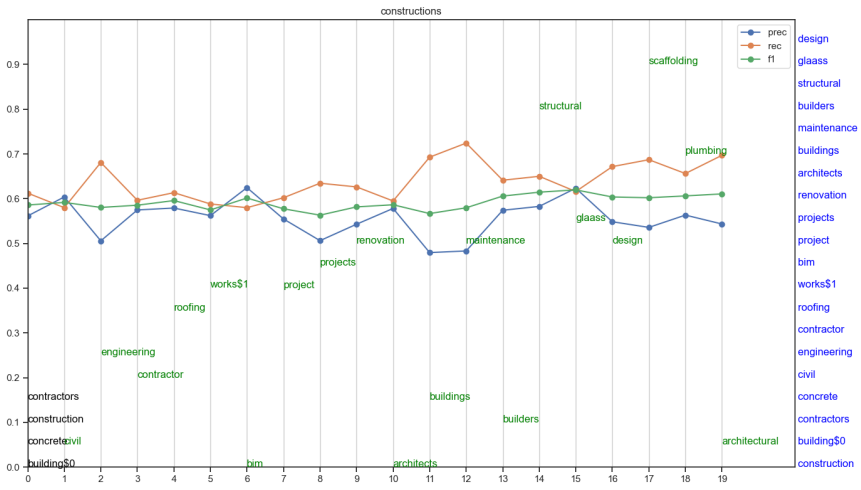


Figure A.29: Precision, Recall and F1-score values of 20 models of the Building class.

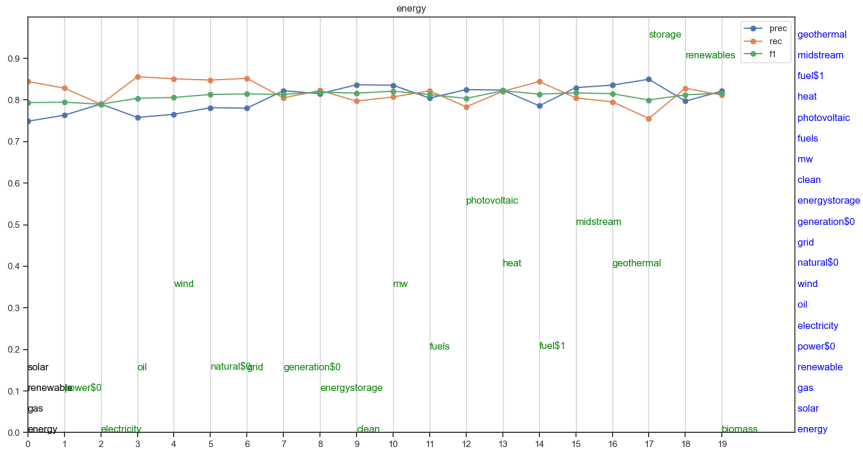


Figure A.30: Precision, Recall and F1-score values of 20 models of the Constructions class.

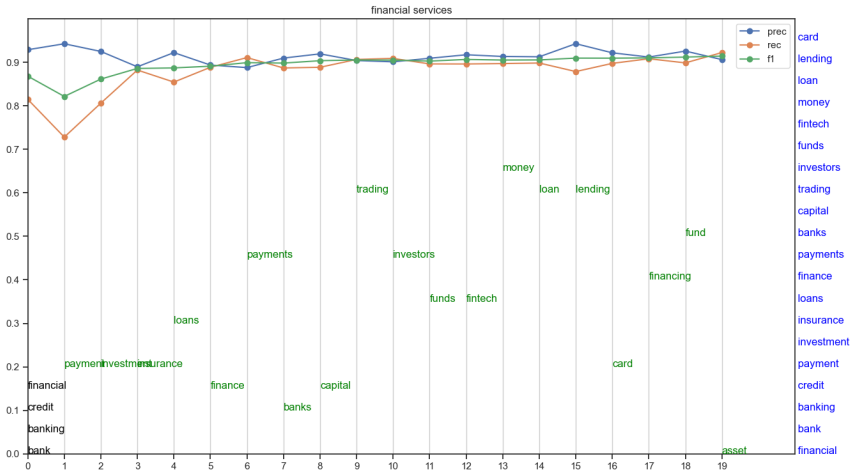


Figure A.31: Precision, Recall and F1-score values of 20 models of the Energy class.

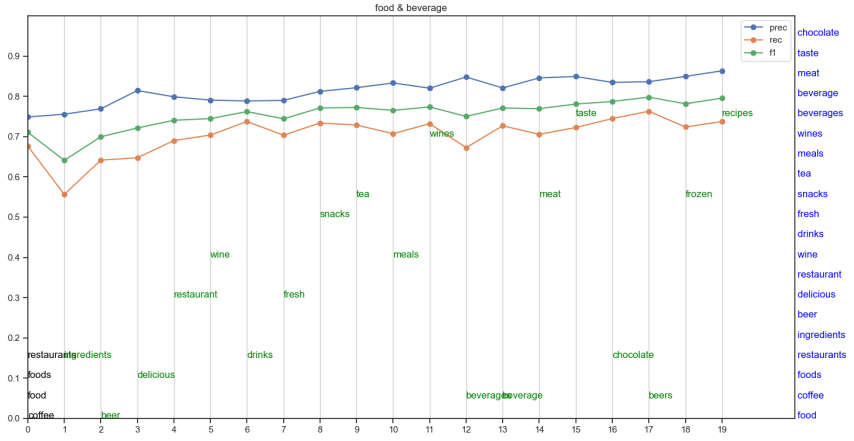


Figure A.32: Precision, Recall and F1-score values of 20 models of the Financial Services class.

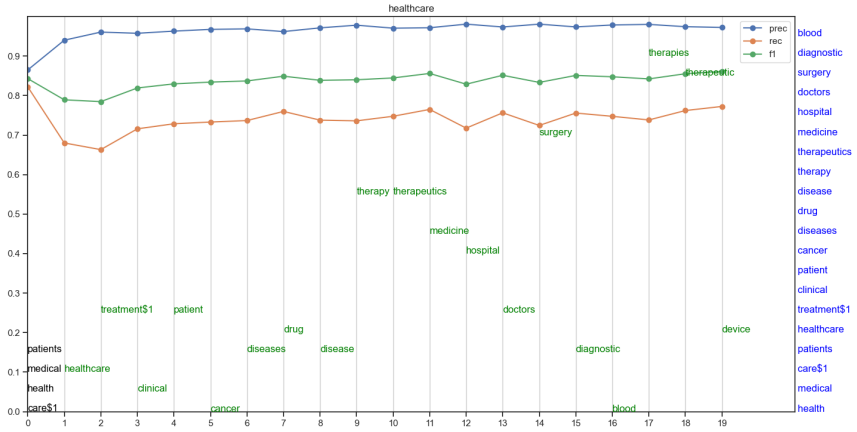


Figure A.33: Precision, Recall and F1-score values of 20 models of the Food & Beverages class.

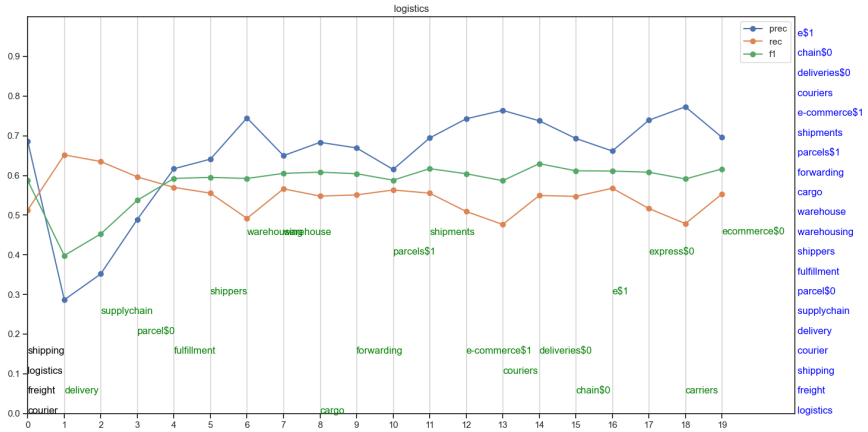


Figure A.34: Precision, Recall and F1-score values of 20 models of the Healthcare class.

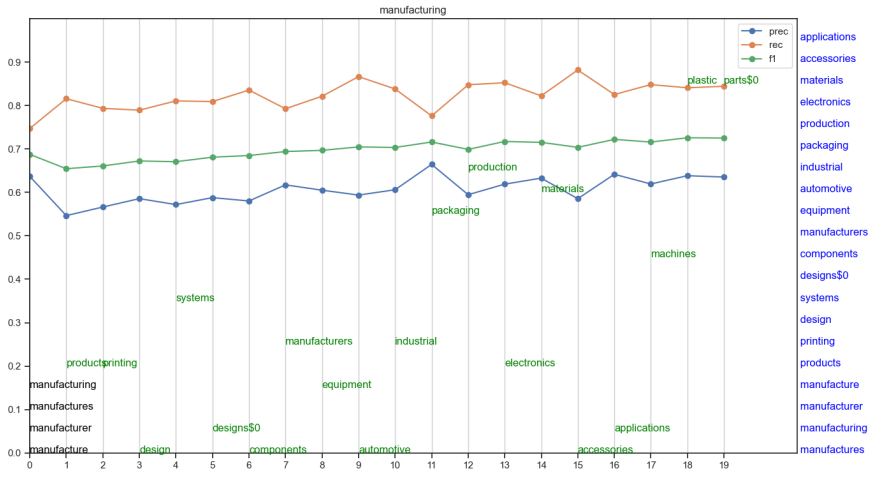


Figure A.35: Precision, Recall and F1-score values of 20 models of the Logistics class.

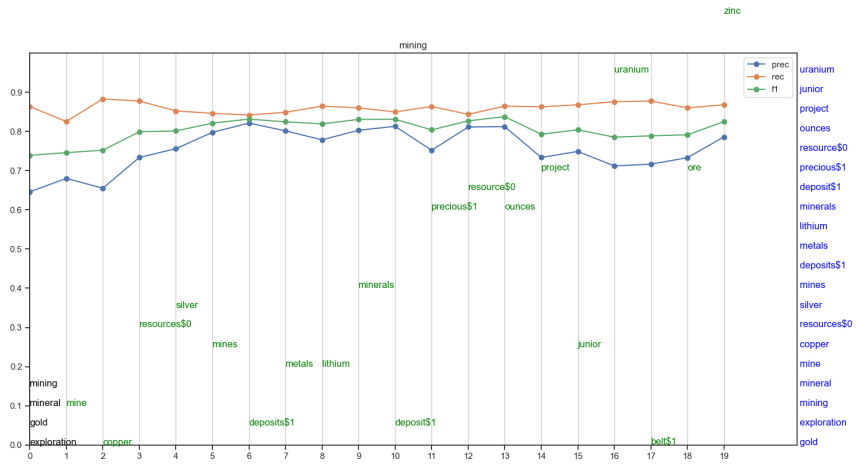


Figure A.36: Precision, Recall and F1-score values of 20 models of the Manufacturing class.

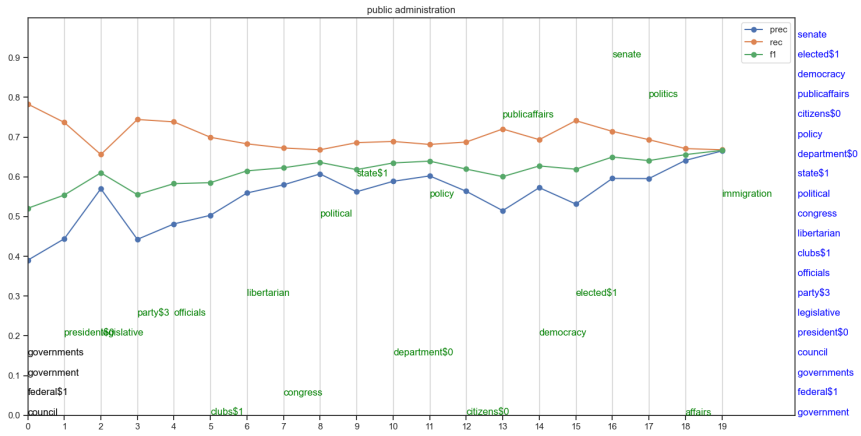


Figure A.37: Precision, Recall and F1-score values of 20 models of the Mining class.

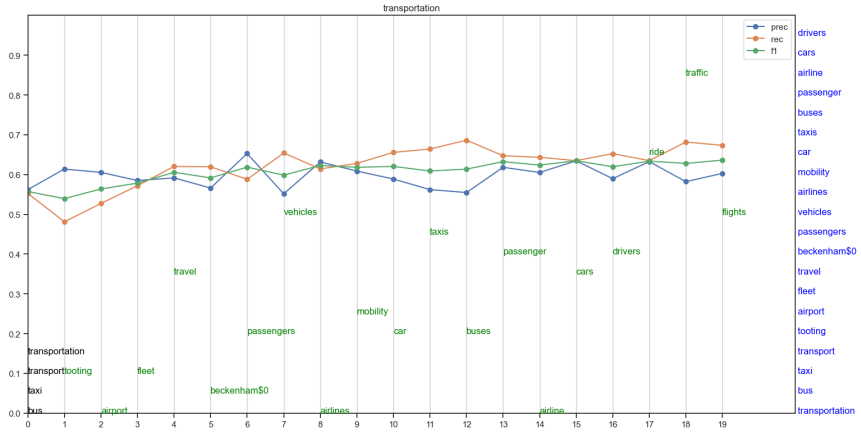


Figure A.38: Precision, Recall and F1-score values of 20 models of the Public Administration class.

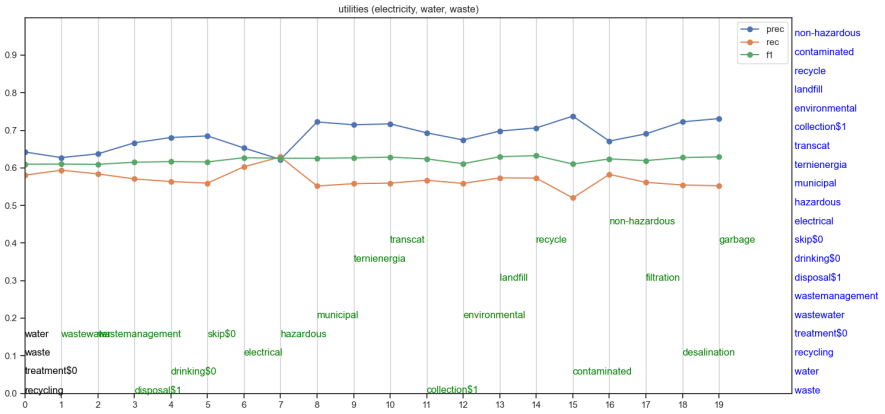


Figure A.39: Precision, Recall and F1-score values of 20 models of the Transportation class.

Bibliography

[al14] Kyunghyun Cho et al. “Learning phrase representations using RNN encoderdecoder for statistical machine translation”. In: (2014) (cit. on p. 28).

[ANN17] Vaswani Ashish, Shazeer Noam, and Parmar Niki. “Attention Is All You Need”. In: (2017) (cit. on pp. 27, 28, 30, 31, 41).

[Arm+17a] Joulin Armand et al. “Bag of Tricks for Efficient Text Classification”. In: (2017) (cit. on pp. 22, 32).

[Arm+17b] Joulin Armand et al. “Enriching Word Vectors with Subword Information”. In: (2017) (cit. on pp. 22, 24).

[Cou+20] David Cournapeau et al. *Sklearn documentation*. 2020. URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html (cit. on p. 46).

[Der16] L. Derczynski. “Complementarity, F-score, and NLP Evaluation. Proceedings of the International Conference on Language Resources and Evaluation.” In: (2016) (cit. on p. 57).

[DJ03] Andrew Y. Ng David M. Blei and Michael I. Jordan. “Latent Dirichlet Allocation”. In: (2003) (cit. on p. 3).

[DKY16] Bahdanau Dzmitry, Cho Kyunghyun, and Bengio Yoshua. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: (2016) (cit. on p. 27).

[Har54] Zellig S. Harris. *Distributional Structure*. 1954 (cit. on p. 18).

[HS97] Sepp Hochreiter and J¨urgen Schmidhuber. “Long short-term memory”. In: (1997) (cit. on pp. 25, 26).

[Jac18] Eisenstein Jacob. *Natural Language Processing*. MIT Press, 2018 (cit. on pp. 5, 8).

[JF16] Ross B. Girshick Junyuan Xie and Ali Farhadi. "Unsupervised deep embedding for clustering analysis". In: (2016) (cit. on p. 50).

[Jor09] Michael I Jordan. "Attractor dynamics and parallelism in a connectionist sequential machine". In: (2009) (cit. on p. 24).

[JT18] Kenton Lee Jacob Devlin Ming-Wei Chang and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: (Oct. 2018). URL: https://arxiv.org/abs/1810.04805v2 (cit. on p. 31).

[Kil+09] Weinberger Kilian et al. "Feature hashing for large scale multitask learning". In: (2009) (cit. on p. 23).

[L R86] B. Juang L. Rabiner. "An introduction to hidden Markov models". In: (1986) (cit. on p. 24).

[Lac17] Peter Lacko. "From Perceptrons to Deep Neural Networks". In: (2017) (cit. on p. 14).

[Lee13] Dong-Hyun Lee. "Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks". In: (2013) (cit. on p. 50).

[Men+20] Yu Meng et al. "Text Classification Using Label Names Only: A Language Model Self-Training Approach". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. 2020 (cit. on pp. 2, 35, 47).

[MS20] Dheeraj Mekala and Jingbo Shang. "Contextualized Weak Supervision for Text Classification". In: (July 2020), pp. 323–333. URL: https://www.aclweb.org/anthology/2020.acl-main.30 (cit. on pp. 2, 35, 39, 48, 57).

[MS69] Minsky Marvin and Papert Seymour A. *Perceptrons, An Introduction to Computational Geometry*. MIT Press, 1969 (cit. on p. 10).

[RB09] Tomas Mikolov Razvan Pascanu and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: (2009) (cit. on p. 25).

[Rog19] Adam Rogers. *unstructured data*. 2019. URL: https://www.forbes.com/sites/forbestechcouncil/2019/01/29/the-80-blind-spot-are-you-ignoring-unstructured-organizational-data/?sh=2b828588211c (cit. on p. 2).

[She+20] Minaee Shervin et al. "Deep Learning Based Text Classification: A Comprehensive Review". In: (2020) (cit. on p. 18).

[Tom+13] Mikolov Tomas et al. "Efficient Estimation of Word Representations in Vector Space". In: (2013) (cit. on p. 19).

[Voi20] Lena Voita. *NLP Course*. 2020. URL: https://lena-voita.github.io/nlp_course.html.

[Yin+19] Liu Yinhan et al. "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: (2019) (cit. on p. 76).

[Yu+18] Meng Yu et al. "Weakly-supervised neural text classification". In: (2018) (cit. on p. 53).

[Zhi+19] Yang Zhilin et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: (2019) (cit. on p. 76).

[Zic+16] Yang Zichao et al. "Hierarchi-cal attention networks for document classification." In: (2016) (cit. on p. 41).