Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

An approach to document classification using verb-object pairs

Relatore:
Chiar.mo Prof.
Maurizio Gabbrielli

Correlatore:
Chiar.mo Dott.
Andrea Zanellati

Presentata da:
Filip Radović

Sessione I
Anno Accademico 2021/2022

Sommario

La classificazione dei documenti (DC) è un problema scientifico che mira ad assegnare un documento a una o più classi o categorie. I problemi che possono essere generalizzati in DC sono scontrati ogni giorno: dal filtraggio di spam alla classificazione di genere la gente risolve questi problemi manualmente o automaticamente. Informatica è principalmente focalizzata alla classificazione di documenti automatica (ADC) che cerca di fare l'assegnazione in modo automatico. La formulazione di strategia per risolvere questo problema richiede la scelta di tecniche adatte da una ampia gamma di scelte a ogni passo della strategia. Questo implica il numero enorme di possibili approcci che possono essere presi per questo problema.

Linguistica, psicologia e psicolinguistica ci forniscono informazioni delle associazioni di parole che possono essere usate nella scelta di caratteristiche (feature selection). Oggi, molteplici metodi sono usati per questo motivo tra cui TF-IDF, BoW, Word2Vec ecc. Quando si decide che caratteristiche scegliere da un documento, uno dovrebbe fare attenzione a farlo in modo che le classi sono effettivamente distinguibili usando quelle caratteristiche.

Lo scopo di questa tesi è di presentare un nuovo approccio alla classificazione del testo usando le coppie verbo-oggetto. Noi esploreremo una possibile strategia che usa la presenza delle coppie verbo-oggetto rilevanti in documenti come caratteristiche e il classificatore Naive Bayes come classificatore su cui il modello è allenato.

Introduction

Document classification (DC) is a scientific problem that aims to assign a document to one or more classes or categories. The problems that can be generalized in DC are faced every day: from spam filtering to genre classification people resolve these problems either manually or automatically. Computer science is mostly focused on Automatic document classification (ADC) which tries to do the assignment in automatic way. The strategy formulation for resolving this problem requires choosing adequate techniques from wide spectrum of choices at each step of the strategy. This implies the enormous number of possible approaches that can be made to this problem.
Linguistics, psychology and psycholinguistics provide us with information about word associations that can be used in feature selection. Today, multiple methods are used for this purpose including TF-IDF, BoW, Word2Vec etc. When deciding which features to select from a document, one should pay attention to make classes effectively distinguishable by them.
The aim of this thesis is to present a new approach to document classification using verb-object pairs. We will explore one possible strategy that uses the presence of relevant verb-object pairs in documents as features and a Naive Bayes classifier as a classifier on which the model is trained.

The Chapter 1 will provide reader with more information about Natural Language Processing and necessary knowledge about document classification for understanding the strategy. It will also give the basic information about psycholinguistics that lay behind the decision to use verb-object pairs in afterwards.
After that, in Chapter 2 we will go through some important results in theory that will

combine with knowledge from the previous chapter help us to eventually define our strategy.

In Chapter 3 the theoretical model will be tested in a case study. The case study is based on SDG Detector, a software based on the studied strategy that evaluates the presence of UN Sustainable Development Goal (SDG) indicators in documents.

We will then assess the accuracy of the classifier which will eventually allow us to make remarks and draw the conclusions in Chapter 4 which is also the final. The final chapter will also propose some further research that can be made.

Contents

Chapter 1

Background

In this chapter, we will introduce the reader to some important concepts that will be useful for the understanding of the approach proposed by the thesis. They are not independent of each other and all of them share the same core, computer science and linguistics.

1.1 Natural language processing

Natural language processing (henceforth, NLP) is an applied sub-field of computational linguistics that is concerned with the processing of natural language by computer. The idea of creating a machine being able to comprehend human language existed for centuries before the emergence of NLP in the 1950s. The first important result was Turing test [1] that aims to decide if a machine's intelligence is equivalent to human by evaluating the ability of the computer to communicate in human language. In 1966, an early NLP system that simulated a Rogerian psychotherapist called ELIZA [2] was developed successfully. ELIZA was, in fact, one of the first chatbots ever developed and it was also one of the first attempts to convince the user that it speaks with a human and thereby pass the Turing test. Since 1980s, machine learning algorithms became more and more used by developers. The victory of IBM's DeepBlue in chess against Gary Kasparov presented a motivation to continue the research in AI. In 2006, they eventually created Watson, a question-answering computer that was able to compete on

the quiz show *Jeopardy!* winning the first prize. In recent years, the surge of intelligent virtual assistants which use speech recognition (one of the important tasks of NLP) like Samsung's Bixby or Amazon Alexa is notable. Nowadays, NLP systems are widespread, we can use them for accomplishing the simple tasks like making a web search through virtual assistants or some more complex job like analysis of text collections in academic work.

1.1.1 NLP tasks

The state-of-the-art NLP systems are capable of solving different language processing tasks. We will address some of them in this subsection.

Tokenization

Tokenization is the process during which a string is being broken up in parts (called tokens) following a certain rule. The most significant type of tokenization is word segmentation, which represents the process of dividing a text in its words and punctuation. This task is usually performed before any other because it results necessary for further text manipulation.
Example: "I am a student at the University of Bologna." → ['I', 'am', 'a', 'student', 'at', 'the', 'University', 'of', 'Bologna', '.']

Part-of-speech tagging

Part-of-speech tagging (henceforth, POS tagging) is the process of labeling words in text with corresponding part of speech. Of course, before performing this task, we need to tokenize the text we want to tag. At first sight, this can seem easier than it is in reality because one could think that it is possible to map every word of a dictionary to only one POS tag. In fact, there are many words in English and other natural languages that are ambiguous in their POS. There are many approaches to this task, from simple rules [3] to hidden Markov chains [4] and Trigrams'n'Tags [5].
Example 1: "$\underbrace{\text{I}}_{\text{PRP}}\ \underbrace{\text{study}}_{\text{VBP}}\ \underbrace{\text{at}}_{\text{IN}}\ \underbrace{\text{the}}_{\text{DT}}\ \underbrace{\text{University}}_{\text{NNP}}\ \underbrace{\text{of}}_{\text{IN}}\ \underbrace{\text{Bologna}}_{\text{NNP}}\ \underbrace{\text{.}}_{\text{.}}$"

Example 2: "$\underbrace{\text{I}}_{\text{PRP}}\ \underbrace{\text{did}}_{\text{VBD}}\ \underbrace{\text{my}}_{\text{PRP\$}}\ \underbrace{\text{study}}_{\text{NN}}\ \underbrace{\text{at}}_{\text{IN}}\ \underbrace{\text{the}}_{\text{DT}}\ \underbrace{\text{University}}_{\text{NNP}}\ \underbrace{\text{of}}_{\text{IN}}\ \underbrace{\text{Bologna}}_{\text{NNP}}\ \underbrace{\text{.}}_{\text{.}}$"
As we can see, in the first example the word *study* is tagged with *VBP* which represents a verb that is in non-3rd person singular form and in the second example it is marked as *NN* which stands for a singular noun. The tags used in these examples are more famous as Penn POS tags [6].

Stemming

Stemming is the process of transforming words into their base form also called word stems. It has many applications, for example we can use it to count the occurrences of a term independently of the form it takes.
Example: Words *study*, *studies* and *studying* have the same stem: *studi*.

Lemmatization

Lemmatization is the process of reducing a word in lemma, i.e., its dictionary form. It is closely related to stemming, but in the case of stemming we do not have any knowledge about the context in which the word appears, but when it comes to lemmatization, we take in consideration its POS tag and the context. These additional checks obviously make lemmatization slower because it does not use the hard-coded rules but more complex procedures. This is a tradeoff because the results from the lemmatization are usually more useful because of the better insight on a single word during the reduction.
Example: The word *studying* if considered a noun will remain the same after lemmatization, but if lemmatized as a verb it will become *study*.

Parse trees

Formal grammar of a language is used to form strings composed of the language's alphabet that respect the language's syntax. Parse trees are the tree representations of the syntactic structure of a sentence or string according to some formal grammar. They are widely used in computational linguistics, and their construction is one of the tasks that are performed by modern NLP systems. There are two types of formal grammars they can be built on: phrase structure grammars and dependency grammars. The former

are concerned with how words and sequences of words combine to form constituents, and the latter are instead focused on how words relate to other words [7]. If a sentence can be interpreted in more than one way by a grammar it is called ambiguous and for each interpretation one parsing tree can be built. The set of these trees is called parse forest.

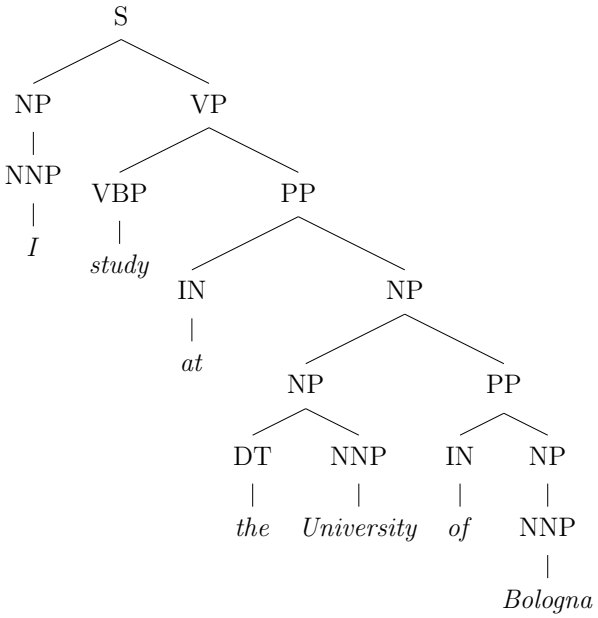


Figure 1.1: Example of constituency-based parse tree

Constituency-based parse trees which are based on phrase structure grammars have terminal categories as leaf nodes and non-terminal categories as interior node. In Figure 1.1 we can see an example of constituency-based tree that is built from the sentence "I study at the University of Bologna". Leaf nodes (we do not consider words) are terminal categories: *NNP* represents singular proper noun, *VBP* verb of non-3rd person singular present, *IN* preposition or subordinating conjunction and *DT* determiner. We have also interior nodes: *S* represents sentence, *VP* verb phrase, *NP* noun phrase and *PP* prepositional phrase. Leaf nodes can be seen as result of simple POS tagging and abbreviations used in this example are part of Penn POS tags [6].

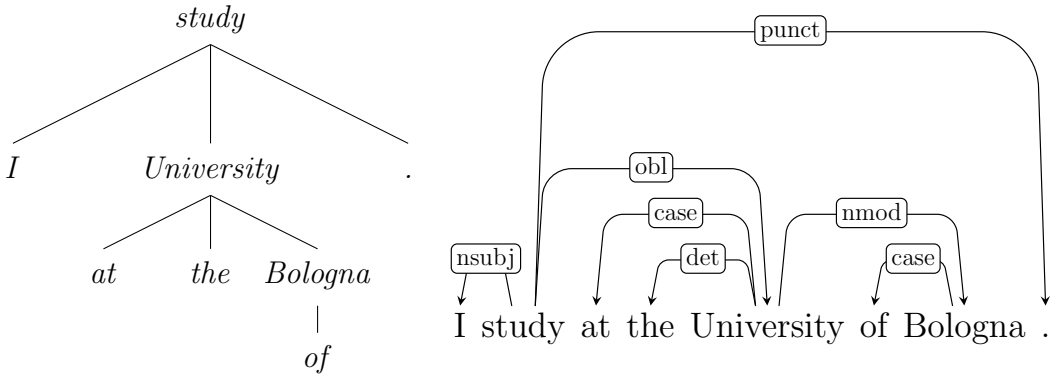


Figure 1.2: Example of dependency-based parse tree (left) and dependency graph (right)

Dependency-based parse trees are based on dependency grammars. In such a tree all nodes contain terminal elements and they can be represented using a dependency graph too. In Figure 1.2 both representations are presented for the same example from Figure 1.1. The parse tree visualization is simple, it contains only terminals, i.e. words and punctuation. On the other hand, in dependency graph we named the relations between the words using Universal Dependencies (UD) relations abbreviations [8]. UD is "a framework for morphosyntactic annotation of human language" [9]. UD v2 recognizes 37 universal syntactic relations and is widely used due to its cross-linguistic consistency and availability. Some of those relations are present in this graph: *nsubj* stands for nominal subject, *punct* for punctuation, *obl* for oblique nominal, *case* for case marking, *det* for determiner and *nmod* for nominal modifier.

1.1.2 NLP toolkits

Nowadays, many NLP toolkits are available to use. The vast majority of them is made for Python programming language. Python is a programming language known for its simplicity but also capability of performing the most complex tasks in spheres of industry and scientific research. Some of the most popular NLP toolkits are: NLTK [7], spaCy [10] and Stanza (more famous as Python interface to Stanford CoreNLP) [11]. All of them can perform tasks we have previously seen with many others. It is difficult to make any comparison at the level of toolkit because their performances differ depending

on tasks. There are also some more "exterior" elements in which we can compare toolkits, like cross-linguistic support.

1.2 Document classification

In the introduction, we defined document classification as a scientific problem that aims to assign a document to one or more classes or categories. In this section, we will describe the problem in more detailed way and present possible strategies for the resolution.

1.2.1 Description of the problem

People encounter DC problems on a daily basis, but most of them are resolved automatically by software that uses NLP mechanisms. Whether it is to filter spam mails or to label books by genre, one should generally go through the same process.
Let's first see how a human would resolve this problem manually. We will take the example of book labeling to make it more illustrative without losing the generality of the problem. First, we need a finite set G of genres which we will use to label the books. Another requirement is that the person that is responsible for classification needs to be well-informed about the characteristics of each of the genres, otherwise it would not be able to recognize it. There are two ways how one can get informed about it: by reading already labeled books from each genre or through reading the descriptors of each of genres. Here we meet first limitations, reading the books from a genre can make us aware of key differences in contents between the books of different genres and the efficiency of the descriptor is limited by its precision. In practice, one usually undertakes some kind of hybrid approach that combines these two ways of informing. In DC, content characteristics that provide us with information useful for the problem are called features. For example we can treat the presence of aliens and fictional worlds in a novel as a feature that tells us it may be labeled as science fiction novel. The sole presence of fictional worlds could also make us consider other genres like ordinary fiction or fantasy, but in any case it would still be a restriction of the initial set of genres. Some problems have peculiarities like the one we are examining, for example some books usually have summaries which can be

sufficient for content analysis. This can be of course a tradeoff because it would increase uncertainty of the result comparing to the case in which analyze the whole book. After we have extracted the features from the content, we can analyze them and conclude to which genre we want to assign the book.
We can go through another example in order to convince the reader in the generality of the problem. Email filtering has been of great interest in recent years, and it is another very illustrative DC problem. Suppose we want to filter out fraudulent mails. Similarly as in the case of book labeling, we can learn about them from some concise descriptions or learn about them by reading the examples. Many deceptive mails have similar forms, a famous example is lottery scam. Most of them would have a variation of the sentence "You won!" as the title, so we could use it as one of the features. Nowadays, the recognition of this kind of spam is trivial for spam filters.
As the reader may have noted, the process of classification is pretty straightforward if we do not take peculiarities into account. Because of this, the process can be automatized.

1.2.2 Automatic document classification strategy

As the name suggests, automatic document classification (ADC) is concerned with resolving DC problems automatically. Today some problems like spam filtering are almost completely delegated to computer thanks to the progress of NLP mechanisms and scientific research in the areas of computer science and computational linguistics.
ADC solutions rely mostly on classifier models that are used to predict the label of the document which is given in input. If we need previously some information in the form of data for constructing classifier model, we will call such document classification supervised. In the recent years, other two types of classification were the subject of interest for researchers: unsupervised and semi-supervised document classification. Unsupervised classification can construct classifier model without any data and semi-supervised need few already labeled data.
We can use a generic strategy [12] for ADC illustrated by a process diagram in Figure 1.3. The generic strategy is composed of six steps and it shows the process of the derivation of classifier from a training set of documents. The strategy is obviously made for supervised document classification. The first step is the preparation of a training set of

documents. If we want to train a supervised classifier we need a lot of already labeled documents, so this step is very important. One part of that documents will also be used as testing data. The second step is preprocessing and its task is to transform the training set of documents in dataset. After that, we have to extract the useful features of documents. After the selecting of appropriate machine learning model for classification, we can train it using the features and labels to train classifier. Eventually, we test the classifier to assess its accuracy.

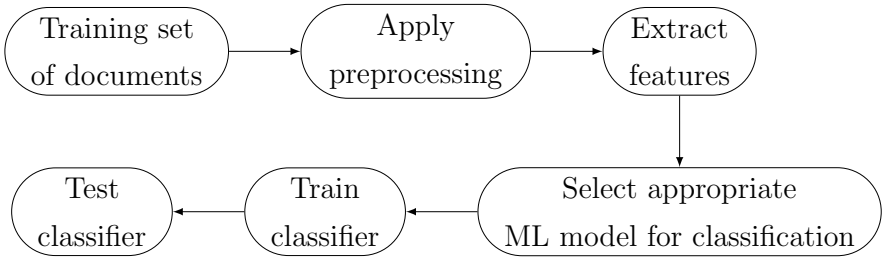


Figure 1.3: Generic ADC strategy

Now we will see more technical details about these steps.

Preparation of the set of documents

At the beginning of the process, we need to prepare a set of documents we will eventually use to train and test the classifier we want to train. We will denote with the term corpus a structured set of texts that usually have already passed some kind of preprocessing. Many available toolkits come with their own corpus package that can be very useful. For example, NLTK has its *nltk.corpus* package that contains dozens of different corpora. If the available corpora are not sufficient to the user it usually resorts to search of alternative datasets online, for example using DBpedia. The last and the most complex way of the preparation of the set is requiring the user to create it by itself. This can be done in many ways and its difficulty depends on the approach that is undertaken. The user can create it using some kind of data scraping like web scraping. In that case, the user has to write the program that will collect the useful documents from the internet. At the end, in case that there are no useful resources on the internet the user could overcome the problem by doing data collection which can also be a very

difficult job.

Preprocessing of documents

In many experiments already preprocessed corpora can suffice the user's needs, but in the case it wants to work with scraped data it is necessary. If the data contains superfluous information, it can attenuate the efficiency of the classifier. We can think of an example where we get an HTML document using web scraping methods. After the scraping, we cannot perform instantly NLP tasks on such a document because it contains tags and other elements irrelevant for further analysis. If we do not remove it, the text could be misinterpreted in for example, feature selection.
Many Python packages appear to be useful for this task. If we want to do web scraping, we can use *Beautiful Soup*. It successfully parses HTML and XML files. A user can also get text cleaned from tags using a simple function, which makes that part of preprocessing simple. Another useful library in Python is *re* which allows us to manipulate text using regular expressions. Regular expressions are simple way of extracting sequences from text that match a pattern required by the user. Sometimes, the preprocessing involves the elimination of stop-words, commonly used function words, like definite and indefinite articles or auxiliary verbs. These operations allow us to work with cleaned instead of raw content. If the cleaned text has a canonical form, we can speak also about the normalization of text.
The following operations usually depend on what the feature extraction process looks like. If we need to check the presence of words in text, one of the standard procedures would be text tokenization. If our features depend on syntactic relations in sentences we will need to do parse dependency-based trees which requires previously tokenization. This part of preprocessing could also require some other NLP tasks, and most of toolkits come with functions that allow their performing.
The preprocessing can be really complex, and sometimes it is rather convenient to store preprocessed information if it is likely to be reused commonly. Say our feature extractor checks if words of a certain text are present in one which is evaluated, it will be computationally more efficient to store the list of words from that text and load it when demanded than to extract them on every evaluation.

Feature extraction

Feature extraction is a very important part of the strategy. Features represent characteristics of text that help the classifier to differentiate various types of text. Therefore, the features have to be informative.

One of the traditional models for feature extraction is Bag of Words (BoW). So far, in our examples, we commonly used a simplified version of this model without naming it. The idea of BoW is to present the frequencies of certain words as the feature of a text. We can keep track of the occurrences or presence of each word in two different ways, depending on what is our goal. If the number of occurrences does not matter, each word can be represented by a Boolean, true if present in text, otherwise false. If the number does matter, we can simply represent words as integers that denote how many times the word appears in text.

Example: For text: "I study at the University of Bologna. I am currently enrolled in the third year." and BoW ['I', 'study', 'Bologna', 'work'] we will have the following result array: [2, 1, 1, 0] (Word *I* occurs 2 times, *study* and *Bologna* occur one time each and *work* is not present in the text). In the case of the array of presences, we will have [True, True, True, False] because all words but *work* appear in the text.

Another commonly used method is term frequency-inverse document frequency (TF-IDF) [13] which is similar to the previous method because of the term frequency part. TF-IDF tends to reduce the significance of common words in documents. For example, the presence of stop words is not very informative feature, and they appear in most of the documents so by using the TF-IDF we ignore them. We can calculate the TF-IDF weight of term using the following formula:

$$W_d = f(w, d)\ log(\frac{|D|}{f(w, D)})$$

where W_d is the weight we are looking for, f is occurrence function that in the case when the second argument is a single document d returns the number of the occurrences of word w in d and in the case when the second argument is corpora D it returns the number of documents in which word w is present. We use $|\cdot|$ to denote the cardinality of a set.

Even we can exploit syntactic relations from the preprocessing phase, it does not imply that we necessarily catch the semantics, so in recent years modes to do so emerged. One

of the best examples is *Word2Vec* that is based on the work of Mikolov et al. [14].

Machine learning models

Now we will see some machine learning techniques used in text classification. They are important because they define how the classifier will be constructed and how it will predict the label of input text.
One of the most simple classifiers is Naive Bayes classifier (NBC) which is based on Bayes theorem. Bayes theorem can be given through the following equation

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

where $P(A|B)$ is probability that event A will occur given that B is true. We can now adapt this formula to our needs, so we can get the probability that a document d is of a class c that belongs to the predefined set of classes C:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} = \frac{P(d_1, ..., d_n|c)P(c)}{P(d)}$$

where $\{d_1, ..., d_n\}$ is the set of document features of d. Now we can define the result of classification as follows:

$$\underset{c \in C}{argmax}\ P(c|d) = \underset{c \in C}{argmax}\ P(d_1, ..., d_n|c)P(c) = \underset{c \in C}{argmax}\ P(c) \prod_{i=1}^{n} P(d_i|c)$$

where in the first equation we left out $\frac{1}{P(d)}$ because we can assume that it is constant and does not have any effect on the result and in the second equation $P(d_1, ..., d_n|c)$ became a product of a sequence because of *naive* assumption that the features are mutually independent. There are also some variations of NBC like Multinomial NBC which can be used when we want to take for example the number of occurrences of words in account. Another famous model is Support Vector Machine (SVM) [15] which seeks to find the maximum margin hyperplane that divides two convex hulls of points (feature vectors) belonging to one class. Because it works on the sets of classes with only two classes, this was traditionally a binary classifier. Through years, a multi-class version evolved using Multiple SVM (MSVM). There is no standard implementation of MSVM, but an

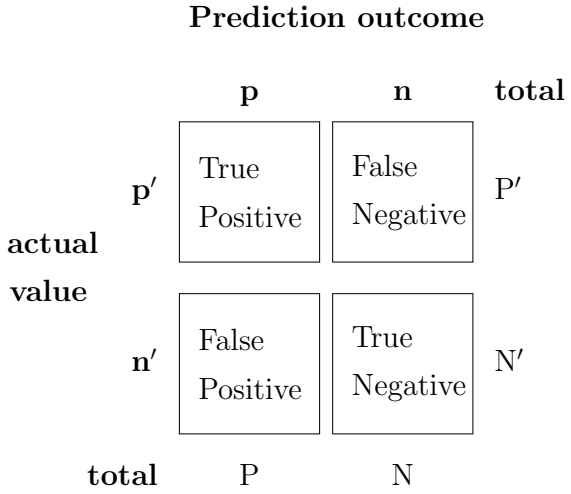


Figure 1.4: Example of confusion matrix

empirical study presented the superiority of pairwise coupling approach to MSVM [16]. Modern classifiers are using deep learning [17], but the traditional ones are still used widely.

Training and testing

Training and testing are the final part of ADC, and they use feature set obtained previously. The feature set can be viewed as a vector of couples $\langle d, l \rangle$ where d is the vector of document features and l is the label assigned to that document. At this point, we have to decide how to do the cross validation of data. Cross-validation methods are used to assess learning algorithm performances by dividing data in two parts: one used for learning (training set) and the other to validate (testing set) [18]. We will mention two cross-validation methods: holdout method and k-fold cross-validation method. Holdout method consists of splitting the feature set in two distinct sets: training set and testing set. In k-fold cross-validation, we divide feature set into k equally sized segments. The division is followed by k iterations of training and testing where at each iteration a different segment is chosen for testing set and the remaining segments are used as training set. It is known that when we have small datasets, it is better to use k-fold cross-validation in order to achieve more accurate results. There is also a special case of k-fold

cross-validation called leave-one-out cross-validation (LOOCV). LOOCV is considered a special case because its configuration parameter k is equal to the length of the evaluated feature set. It is especially used when the data is rare [19] and its pros is that there is almost no bias in evaluation phase, but it can produce very large variance [18]. Bias is an error caused by wrong assumptions in the used algorithm, while variance is an error caused by small variations in the training set. The high variance in LOOCV is expected due to famous bias-variance tradeoff [20] which states that the increase of bias implies the reduction of variance. It is also time-consuming which can be a problem when we use larger dataset. We also do not need to repeat the evaluation as it will not produce any better results [21].

The training part depends on which machine learning model we have chosen. In the case of NBC it is the determination of conditional and class probabilities, while in the case of SVM it is the approximation of hyperplane. In each case we are using the data from the training set to calculate the values and there are multiple ways to do so. Because of this, NBC and SVM can be seen as families of classifiers rather than single classifying methods. For example, in some cases class probabilities in NBC are not equal and cannot be obtained by simple formula $P(c) = \frac{1}{|C|}$ where c is a certain class in the set of classes C.

One of the most used ways of testing is using the confusion matrix. The confusion matrix is constructed in the following way: each row of the matrix represents the actual label of tested instances, and each column represents the predicted label of tested instances. If we use k-fold cross-validation method, then the final confusion matrix is the sum of confusion matrices constructed at each iteration of the method. We can see an example of confusion matrix for a binary classifier with two classes, positive and negative, in Figure 1.4. We can say that True Positives (TP) and True Negatives (TN) are correctly predicted, and our goal is to maximize their share in total tests. We can express that using formula:

$$ACC = \frac{TP + TN}{P + N}$$

where P is the total number of Positives and N is the total number of Negatives. The value calculated by this formula is also called the accuracy of classifier. We are also interested to see balanced accuracy in cases when we have unequal numbers of tests per

class. The balanced accuracy is given by the following formula:

$$BA = \frac{TPR + TNR}{2}$$

where TPR is True Negative Rate ($\frac{TP}{P}$) and TNR is True Negative Rate ($\frac{TN}{N}$). Another interesting coefficient for measuring binary classifier quality is Matthews correlation coefficient (MCC) [22]. We can calculate MCC as follows:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

where FP is the number of False Positives and FN is the number of False Negatives. Recent research showed that MCC maintains consistency through comparison with some other metrics [23]. MCC values range from -1 to 1 and the coefficients close to 1 show almost perfect prediction, the coefficients close to -1 show almost worst possible prediction and the coefficients around 0 show that the model behaves randomly [24].
We will now introduce Cohen's kappa which is another important coefficient in classification statistics and is calculated in the following way [25]:

$$\kappa = \frac{2 \times (TP \times TN - FP \times FN)}{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}$$

There is discussion what magnitudes of kappa show "good agreement". We will for simplicity use Landis and Koch's guidelines [26] that consider magnitude over 0.81 as almost perfect, 0.61 to 0.80 as substantial, 0.41 to 0.60 as moderate, 0.21 to 0.40 as fair, 0.00 to 0.20 as slight and below 0.00 as poor.
Many other evaluation metrics exist, but they suffer from different problems. Some of them leave out important parameters like False Negatives, or it is difficult to decide what values of coefficients can be considered acceptable (which is also the case with Cohen's kappa sometimes).

1.3 Psycholinguistics

In this section, we will introduce some important results from psycholinguistics that will be useful for the following part of the thesis. Psycholinguistics is the study of the

relationship between linguistic behavior and psychological processes. It originates with Chomsky's critical review of B.F. Skinner's work *Verbal Behavior* (1957) [27][28]. This discipline has a wide spectrum of study, and we will focus on language comprehension.

1.3.1 Word associations

Word associations can be useful to DC problems for several reasons. The first reason would be the possibility to measure the connection of each word in text to each class. Methods for quantifying those word associations already exist [29] and they were the subject of study in the past. Another reason is the result that some syntactic relations like verb-object contain significant mutual information (MI). MI is defined as the measure of mutual dependence between two random variables. Existence of pairs of words that have MI can affect the accuracy of some classifiers. For example, NBC assumes that features are mutually independent. However, an empirical study showed that existence of such connections is not directly correlated with the accuracy [30], but still the assumption of mutual independence is questioned. This limitation of word was noted, and some approaches do not use them. One possible alternative is the use of n-grams. n-grams are n-character slices of a longer string. The character n is usually replaced with some number that would represent the dimension of slices, for example a 2-gram (or bi-gram) would be a two character long slice of a string [31]. The problem with n-grams are that they do not have to capture sensible slices, but rather a random group of words. Syntactic n-grams (sn-grams) [32] tend to overcome this problem by creating n-grams over syntactic relations instead of slicing sentences directly. For better understanding, say we want to obtain all the possible sn-grams from the example we illustrated in Figure 1.2. We can identify four distinct sn-grams (we will not count path with punctuation): *I study*, *study at University*, *study the University*, *study University of Bologna*. The value of n would determine the size of these slices.
Most of the toolkits allow to user to extract n-grams efficiently. Extraction of sn-grams requires from the user to create a parse tree before executing n-gram extraction.

1.3.2 WordNet

Another important practical result of this discipline is WordNet [33]. WordNet is a lexical database of semantic relations between words created in Cognitive Science Laboratory of Princeton University. It was first created only for English language, but soon it was expanded to other languages. In this large database, words are grouped into synsets (sets of cognitive synonyms). For example, words *predominate* and *prevail* can be found in the same synset. This can be very useful in text classification because we can group terms by their meaning.

WordNet initially introduced six semantic relations: synonymy, antonymy, hyponymy, meronymy, troponymy and entailment. Synonymy is the most important semantic relation between two words that share at least one sense in common because synsets are based on it. Antonyms are words of opposite meaning and their relation is, just like synonymy, symmetric. Hyponymy is used to represent a word as subordinate to some other word, called hypernym. The existence of hyponymy allows us to organize nouns in hierarchical structures because they usually have only one hypernym. Meronyms denote nouns that are part of some more complex noun called holonym. We can think of word *lip* as meronym and its holonym would be *mouth*. In troponymy one verb is a manner of another. An example would be verb *march* which is a manner of verb *walk*. Entailment is a relation between two verbs that necessarily and unidirectionally entail one another. WordNet also contains cross-POS relations that include morphosemantic links. An example of such relation would be *RESULT* relation between words *slicing* and *slice*.

To this date, WordNet 3.0 has 155287 unique strings, 117659 distinct synsets and 206941 word-sense pairs [34]. Some words are polysemous (have more than one meaning) and as a consequence belong to more than one synset. The most polysemous POS are verbs, with 2.17 average polysemy when we include monosemous words and 3.57 when we exclude them. Polysemous nouns have greater average polysemy than adjectives and adverbs. but still have smaller average polysemy than these two POS when we include monosemous nouns in the calculation too. Totally we have 128391 monosemous and 26896 polysemous words with 79450 senses.

NLTK comes with WordNet package, so it can be used together with other tools we have seen so far without any problems.

Chapter 2

Proposed Strategy

After introducing some important concepts, we can now proceed to the presentation of the strategy that will use verb-object pairs as features. We will do it step by step, following the generic strategy we defined in subsection 1.2.2. We will not go in technical details, but we will rather develop the strategy theoretically. The tasks we have seen until now can be performed by tools provided by NLP toolkits, so by basing the strategy on them, we can assume that the strategy is implementable.

2.1 Motivation

One could ask why verb-object pairs? As we have seen in the section about psycholinguistics, verb-object pairs can contain significant mutual information. By capturing these pairs, we would enclose that information and get a much more informative feature than a single word. We can make an example of sentence: *Tomorrow I am going to drink some coffee.* We can extract the pair *drink-coffee* which is proved to have significant mutual information [29] and use it as feature. The pairs also seem to have much larger degree of mutual independence from single words. Also, the pairs could provide us with more information about classes we want to use for classifying because they represent complete actions while words can frequently mislead when taken from context and be also polysemous.

2.2 Strategy step-by-step

2.2.1 Dataset

The first step is defining how dataset needed for the strategy should look like. The first and only obligatory requirement for the dataset is, like in any other ADC strategy, the set of already labeled documents which will be afterwards transformed into feature set.
Other dataset requirement could be task-related. For example, let's assume that each class from a set of classes has a list of verb-object pairs that are highly correlated to it. Having in dataset those lists would be useful in feature selection because knowing the relevant words a priori would save the classification process from some extra work of approximating them through various methods (like TF-IDF). One could make similar list for words in the case of BoW model, so addition to dataset is not approach related.

2.2.2 Preprocessing

At the beginning of the preprocessing part, we will normalize the content of documents. The canonical form for this purpose will be the text cleaned from any tags or elements that do not belong to human language. This implies that data will be cleaned if any data scraping has been done. No additional work is needed for the preprocessing of texts from corpora.
After that, the only operation we will need to do is tokenization. Obviously for extracting verb-object pairs we will need to do tree parsing, but for efficiency we will leave that work for later to be done on each evaluated sentence during feature extraction phase. We will do this way because we do not want to fill the memory with parse trees that will be used only once, during the passage through the sentence from which tree was derived. No other tasks need to be performed at this point.

Algorithm 1 Feature extraction

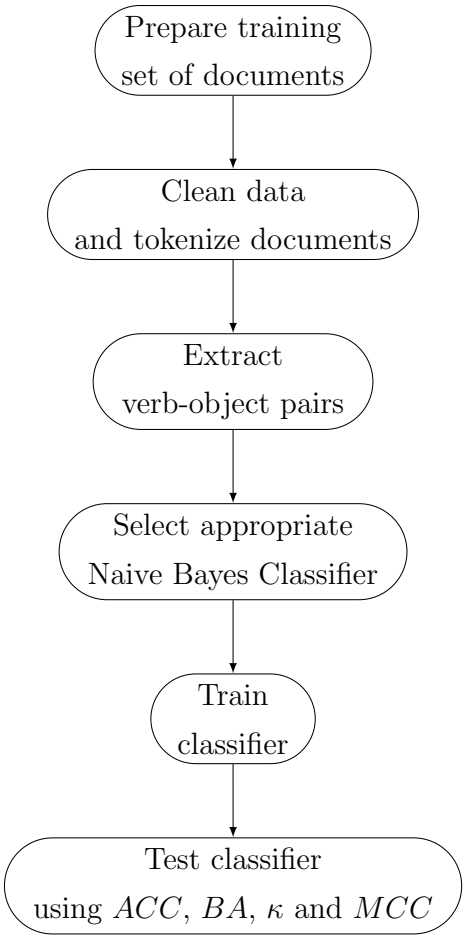
```
function FExtract(doc)
    features ← [ ]
    for all s ∈ doc.sentences() do
        pt ← parseTree(s)
        features.push(lemmatize(pt.getPairs()))
    end for
    return features
end function
```

2.2.3 Feature extraction

Now we are ready to define our feature extraction function. The function will take document text as parameter and will return the array of features where every feature will be a pair where the first element will be verb-object pair and the second element will be Boolean that will be True if the pair is present in the text, otherwise False.
To make this function more efficient, we first create an empty vector where we store the features, and we visit each sentence using a loop. Each sentence should be parsed into a dependency-based or constituency-based tree. If we decide for the former we will simply look if there is direct or indirect object relation going from some verb, and then we will simply extract these two words. This could also be seen also as some special case of s2-gram. The extraction of the pair from constituency-based tree is based on looking up the tree for every noun in the sentence if there is a verb and in that case that verb and noun are extracted [35][11]. After the lemmatization of the extracted pairs we can push them into the features vector. Algorithm 1 shows the pseudocode of feature extraction function. The *getPairs*() function depends on the type of parse tree we have chosen.
Our next goal is to keep only relevant pairs for the training, so we should eliminate those superfluous. There are several ways to do so, and we could use some techniques we used on words like TF-IDF where our term would be the pair. Another approach would use the additional data we described at the first step. Thanks to the additional data, we could eliminate irrelevant pairs efficiently.
At the end of this phase, we have a feature set of relevant verb-object pairs.

2.2.4 Model training and testing

The next decision to be made is to choose a machine learning model that suits the problem. We could use Naive Bayes Classifier because we suppose that we have greater mutual independence of features than in the case of BoW model. What NBC classifier should be used depends on the task. If the number of occurrences of each pair is important, we should use a Multinomial Naive Bayes Classifier.
After the model is chosen we can proceed to training of model - the process completely independent of the strategy and finally to testing. The testing could be done using the four coefficients we have seen previously in 1.2.2.
The strategy can be visualized using the following diagram:



Chapter 3

Case Study

In this case, we will cover the case study of the previously proposed strategy. Our real-world context will be an ADC problem that we will resolve using the implementation of the strategy. The results from testing phase will be afterwards used for making conclusions.

3.1 Description of task

The task we will resolve is to verify the presence of Sustainable Development Goal (SDG) indicators in documents.
SDGs are defined by United Nations (UN) in 2015, and they are intended to be achieved by 2030. The UN published 17 SDGs together with their targets and indicators. Each SDG contains several targets, and each target is monitored by from one up to four indicators. The goals are listed in the following order:

1. No Poverty
2. Zero Hunger
3. Good Health and Well-being,
4. Quality Education
5. Gender Equality

6. Clean Water and Sanitation
7. Affordable and Clean Energy
8. Decent Work and Economic Growth
9. Industry, Innovation and Infrastructure
10. Reduced Inequality
11. Sustainable Cities and Communities
12. Responsible Consumption and Production
13. Climate Action
14. Life Below Water
15. Life On Land
16. Peace, Justice, and Strong Institutions
17. Partnerships for the Goals.

2030 Development Agenda was adopted by 193 countries in 2015. The commitment to the SDGs by countries was passed to all institutions and companies in society. This motivates the need for software that would be able to analyze documents and check their compliance with SDGs. Sustainable development has become the important part of Corporate social responsibility (CSR) [36]. Through the years many guidelines emerged, and we can single out environmental, social, and corporate governance (ESG) approach as of special interest for sustainable development.

We will not use any of the already present guidelines like ESG for resolving our task, but we will use only the information available from 2030 Development Agenda that includes SDG definitions and their targets. Our only task will be to check which from 17 goals are promoted in a queried document. We will assume that the analyzed documents are trustworthy, and no fact checking will be implemented. In reality, this would be needed sometimes because of various spins like greenwashing. Greenwashing can be defined as

the behavior in which a corporation with poor environmental performance communicates positively about its environmental performance [37].

3.2 Description of software

In this section, we will describe SDG Detector, a software used for the resolution of the previous task. The software was developed by my colleague Stefano Colamonaco and me during an internship at the University of Bologna.
Before we commence with the description of code, we will mention what information we were able to use for its implementation. The first useful resource was 2030 Development Agenda that contained the list of all SDGs and their targets explained. Another useful resource was a dataset of verb-object verb lists for each SDG. As we have seen in the previous chapter, this reduces work relative to term relevance check because we have already such a list. We also had positively and negatively labeled documents for each SDG that we used for training and testing. You can see the numbers relative to the dataset in Table 3.1. It is important to mention that verb-object lists were constructed using an experimental method, which is the argument of Colamonaco's thesis. Firstly, using web scraping techniques and internet as the source of information 130 labeled documents were scraped. After that for every goal a list was created and verb-object pairs from documents were put in them. Certain pairs were considered irrelevant, so a blacklist of pairs that should be ignored was made. At the end, the number of documents was elevated to 572. The share of documents from which verb-object pairs were extracted in final documents is 22.73% so this can affect the results negatively because of bias. The bias can be a consequence of the fact that these two knowledge bases were not constructed independently, so we should have that in mind when analyzing the results.
Now we can proceed to the description of the code, which was written in Python for reasons we explained in the first chapter. The documents are in canonical form, so we do not need any kind of normalization. The only preprocessing required is the tokenization, as explained in the proposal of the strategy. We used NLTK for performing the task of tokenization.
The next step was the implementation of an algorithm for feature extraction. Constituency-

Dataset			
SDG number	verb-object pairs	positively labeled documents	negatively labeled documents
1	335	16	16
2	107	18	18
3	344	16	16
4	197	18	18
5	212	17	17
6	189	18	18
7	79	18	18
8	174	16	16
9	141	18	18
10	135	18	18
11	327	18	18
12	220	15	15
13	54	18	18
14	179	17	17
15	72	18	18
16	166	17	17
17	80	10	10

Table 3.1: Tabular view of numbers relative to used dataset

based tree parsing approach was chosen for tree parsing, and it was done using Stanza's parser. The main difference from the algorithm presented as Algorithm 1 and the implementation was another parameter g which is an integer that represents the SDG for which we are doing feature extraction. Before the pushing of lemmatized pairs obtained from the parse tree, it was controlled if they are present in the verb-object list of the g-th goal. In the negative case, another control was made if there was any verb-object pair in the list whose verb's synset contained the evaluated verb and noun's synset contained the evaluated noun. If any of these two controls was passed positively, the pair was pushed, otherwise it was rejected and the algorithm would have proceeded to the evaluation of the next sentence. Lemmatization was performed using `WordNetLemmatizer` from `nltk.stem` package that contains NLTK stemmers. Synset checks were made using WordNet Interface from `nltk.corpus`.

Feature sets are a vector containing 17 another vectors which represented a feature set for each goal. Before the training phase, they were filled with features extracted from labeled documents. The goals were used to decide where to push obtained features to pair with the labels, which are in Boolean form. The labels denote if SDG indicators are present in the document.

There are 17 binary classifiers in total, one for each SDG. Each classifier was trained as a NBC model using the correlated feature set. The model is trained using NLTK's Naive Bayes Classifier implementation. Python's `pickle` module was useful for storing the trained models so if we want to run the program we do not have to wait for new training. Also, the software is available as notebook interface written for Jupyter Notebook. This version increases interactivity.

Figure 3.1 contains an example of document classification using the software we have just described. After the passing of document and goal number to the feature extraction function, we extract verb-object pairs and put them in feature set vector if they are present in verb-object list for the requested goal (green background) or we discard them (red label). Finally, we pass the feature set alongside with the goal number to the classifier and get the result.

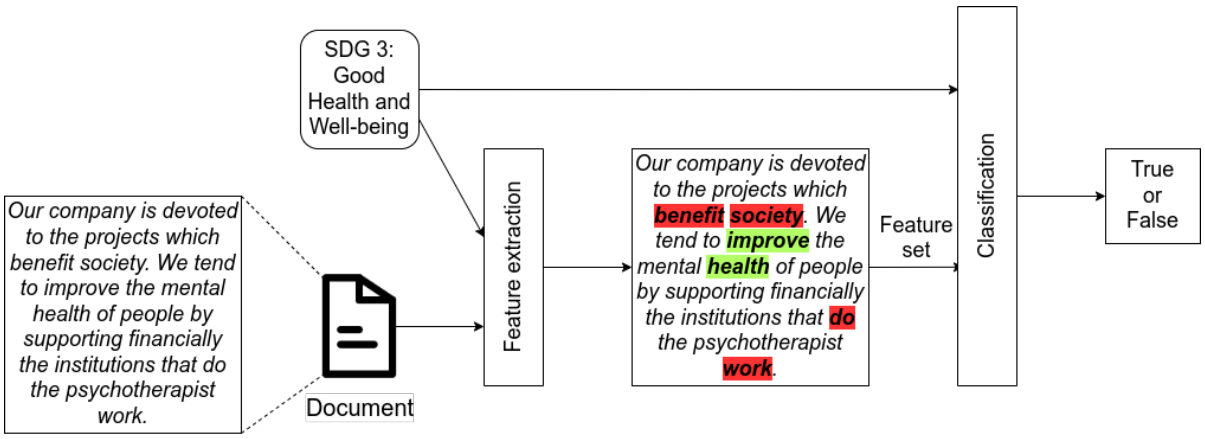


Figure 3.1: An example of document classification using SDG Detector

3.3 Testing

In this section, we will evaluate the models using the methods we have seen in 1.2.2. As the reader may have noted, the dataset used for training and testing is very small, and each goal has on average 33.65 documents that can be used for testing. Because of this, we will use leave-one-out cross-validation (LOOCV). It is time-consuming, but in our case it is not a problem because of the dimension of dataset.

Testing was done using `KFold` from scikit-learn's `sklearn.model_selection` package. In order to use it, the transformation of feature set into a `numpy.array` was needed, so NumPy was imported. Scikit-learn and NumPy are Python libraries. The former contains machine learning algorithm and the latter is used usually for array and matrix manipulation.

In Table 3.2 are reported the values of the resulting confusion matrix. First thing we can note is that we have False Positives only in two classifiers, for SDGs 8 and 14. Also, it is notable that the model for SDG 14 is the only one that fails to satisfy $TP + TN > FP + FN$ inequation. The first observation justifies the need for prediction bias check. Prediction bias is the quantity that describes how different average predictions are from average actual labels. We can see that in some cases like of the classifier for the 14th goal it labels 35 out of 36 documents as negative which is 97.22% and in reality only 50% of them are negative, so the calculated prediction bias for that classifier is

Confusion matrix values				
SDG number	True Positives	False Negatives	True Negatives	False Positives
1	4	12	16	0
2	2	16	18	0
3	3	13	16	0
4	1	17	18	0
5	15	2	17	0
6	3	15	18	0
7	3	15	18	0
8	11	5	6	10
9	4	14	18	0
10	6	12	12	0
11	17	1	18	0
12	12	3	15	0
13	6	12	18	0
14	14	3	0	17
15	1	17	18	0
16	13	4	17	0
17	9	1	10	0

Table 3.2: Distribution of test results for each goal

$97.22\% - 50\% = 47.22\%$. It is also notable that the prediction bias is always towards negative labels, except in the case of the classifiers for the 8th and the 14th goal. For the following goals we have the classifiers with prediction biases greater than 40%: 2, 3, 4, 6, 7, 14 and 15. Prediction bias is commonly a consequence of biased sample. If we take in consideration that the dataset was created using an experimental method, we can say that this kind of behavior was expected.

We should now decide what coefficients to calculate using this table in order to assess the classifiers. We will calculate accuracy, Cohen's kappa and Matthews correlation coefficient. The values will be interpreted in the following way: accuracy values over 0.50 are acceptable (because that means it performs better than random guessing) and those closer to 1.00 are almost perfect, for κ values we will use Landis and Koch's guidelines and MCC values over 0.50 are good as they are closer to 1.00 which represent perfect performance than to 0.00 which represent randomness. We will not calculate balanced accuracy because it would have the same value as accuracy because the numbers of the positively labeled documents and the negatively labeled documents are equal:

$$BA = \frac{\frac{TP}{P} + \frac{TN}{N}}{2} \overset{P=N}{=} \frac{TP + TN}{2P} = \frac{TP + TN}{P + N} = ACC$$

The calculated coefficient values can be found in Table 3.3.

As we could observe before the calculation, the classifiers for all goals, except for the 14th, have acceptable performances when it comes to accuracy measurements. Seven of them scored in the range of values from 0.51 to 0.60 which can be seen as acceptable but low confident performance. Four of them scored between 0.61 and 0.85, so we can call these classifiers fair to moderate. At the end, five of them had scores over 0.85 which is very good. Average accuracy is 67.8%. Note that there is no standard evaluating method for accuracy values and that this one presented here is arbitrary.

Using Landis and Koch's guidelines we can assert that there is one classifier with poor, seven classifiers with slight, four with fair, two with substantial and three with almost perfect performances. Average value of κ is 0.356, so we can say that classifiers have fair performances.

Assessment of Matthews correlation coefficient show that most of the classifiers fall into a domain that goes from -0.5 to 0.5 which expresses more random behavior (because

Coefficient values			
SDG number	ACC	κ	MCC
1	0.625	0.250	0.378
2	0.556	0.111	0.242
3	0.594	0.187	0.322
4	0.528	0.056	0.169
5	0.941	0.882	0.888
6	0.583	0.167	0.301
7	0.583	0.167	0.301
8	0.531	0.062	0.066
9	0.611	0.222	0.353
10	0.667	0.333	0.447
11	0.972	0.944	0.946
12	0.900	0.800	0.816
13	0.667	0.333	0.447
14	0.412	-0.176	-0.311
15	0.528	0.056	0.169
16	0.882	0.765	0.787
17	0.950	0.900	0.904
Mean	0.678	0.356	0.425

Table 3.3: Coefficient values calculated from test results

those coefficients are closer to 0 than to extremes -1 and 1). Even twelve of them belong to this category, but only one of them is in the range that goes from -0.5 to 0 which is the sign of poor performance. On average, MCC equals to 0.425, slightly below 0.5 threshold.

Although on average our strategy did not succeed in going over the 0.5 threshold for MCC, we can say it was quite successful. If we leave out goals that have prediction bias greater than 40% we have on average the following values: 0.77 for accuracy, 0.55 for κ and 0.6 for MCC. Obviously, these results are much better than before. MCC is greater than 0.5, we have almost substantial performance according to κ and accuracy is much closer to the perfect one than previously.

If we leave out these goals, we are also much closer to the symmetric distribution of coefficients. This is implied by the comparison of the absolute differences of mean and median values. The calculated mean and median values of coefficients are reported together with their absolute differences in Table 3.4.

Coefficient values			
Case	ACC	κ	MCC
Mean - all goals	0.6782	0.3564	0.4250
Median - all goals	0.6110	0.2220	0.3530
Difference - all goals	0.0672	0.1344	0.0720
Mean - biased left out	0.7746	0.5491	0.6032
Median - biased left out	0.7745	0.5490	0.6170
Difference - biased left out	0.0001	0.0001	0.0138

Table 3.4: Mean, median and absolute difference between mean and median coefficient values

Chapter 4

Conclusions

In this thesis, we discussed a possible approach to document classification using verb-object pairs. We started by giving some important theoretical foundations and afterwards we built the strategy proposed by thesis step-by step. At the end, we evaluated the strategy using a case study where we resolved the problem of document classification according to the presence of SDG indicators.

The case study, which is the focal point of this thesis, presented some interesting results. The coefficients we obtained during the testing phase can be considered acceptable. If we leave out data that causes prediction bias, we get the coefficient distribution that is almost normal. In that case we got great accuracy equal to 77%. Cohen's κ value was equal to 0.55 which means the classifier on average had a moderate performance. Matthews correlation coefficient was equal to 0.6 which clearly showed that the classifiers do not perform randomly and has very positive correlation by being closer to 1 which represents total positive correlation than to 0 which represents randomness.

The results are encouraging, not only for this specific approach, but also for the testing of document classification approaches using some of many other informative syntactic relations that could potentially have excellent results. Further study of this approach could include its testing on a dataset of much bigger dimension and its comparison to the other approaches with independently constructed knowledge bases. It could also compare the use of different machine learning algorithms for classification.

Bibliography

[1] Alan M Turing. Computing machinery and intelligence. In *Parsing the Turing test*, pages 23–65. Springer, 2009.

[2] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

[3] Eric Brill. A simple rule-based part of speech tagger. In *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*, 1992.

[4] Douglass Cutting, Julian Kupiec, Jan Pedersen, and Penelope Sibun. A practical part-of-speech tagger. In *Third conference on applied natural language processing*, pages 133–140, 1992.

[5] Thorsten Brants. Tnt-a statistical part-of-speech tagger. *arXiv preprint cs/0003055*, 2000.

[6] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[7] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[8] Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 4585–4592, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA).

[9] Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre, and Daniel Zeman. Universal Dependencies. *Computational Linguistics*, 47(2):255–308, 07 2021.

[10] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python, 2020.

[11] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2020.

[12] Mita Dalal and Mukesh Zaveri. Automatic text classification: A technical review. *International Journal of Computer Applications*, 28, 08 2011.

[13] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 29–48. New Jersey, USA, 2003.

[14] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[15] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[16] Kai-Bo Duan and S. Sathiya Keerthi. Which is the best multiclass svm method? an empirical study. In Nikunj C. Oza, Robi Polikar, Josef Kittler, and Fabio Roli, editors, *Multiple Classifier Systems*, pages 278–285, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[17] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information*, 10(4), 2019.

[18] Payam Refaeilzadeh, Lei Tang, and Huan Liu. *Cross-Validation*, pages 1–7. Springer New York, New York, NY, 2016.

[19] Andrew J Stephenson, Alex Smith, Michael W Kattan, Jaya Satagopan, Victor E Reuter, Peter T Scardino, and William L Gerald. Integration of gene expression profiling and clinical variables to predict prostate carcinoma recurrence after radical prostatectomy. *Cancer: Interdisciplinary International Journal of the American Cancer Society*, 104(2):290–298, 2005.

[20] Ron Kohavi, David H Wolpert, et al. Bias plus variance decomposition for zero-one loss functions. In *ICML*, volume 96, pages 275–83, 1996.

[21] Tzu-Tsung Wong. Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation. *Pattern Recognition*, 48(9):2839–2846, 2015.

[22] B.W. Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, 405(2):442–451, 1975.

[23] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21(1):1–13, 2020.

[24] Pierre Baldi, Søren Brunak, Yves Chauvin, Claus AF Andersen, and Henrik Nielsen. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, 16(5):412–424, 2000.

[25] Davide Chicco, Matthijs J. Warrens, and Giuseppe Jurman. The matthews correlation coefficient (mcc) is more informative than cohen's kappa and brier score in binary classification assessment. *IEEE Access*, 9:78368–78381, 2021.

[26] J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.

[27] B.F. Skinner. *Verbal Behavior*. Appleton-Century-Crofts, 1957.

[28] Noam Chomsky. Review of b. f. skinner's verbal behavior. *Language*, 35(1):26–58, 1959.

[29] Kenneth Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

[30] Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, pages 41–46, 2001.

[31] William B Cavnar, John M Trenkle, et al. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*, volume 161175. Citeseer, 1994.

[32] Grigori Sidorov, Francisco Velasquez, Efstathios Stamatatos, Alexander Gelbukh, and Liliana Chanona-Hernández. Syntactic dependency-based n-grams as classification features. In *Mexican International Conference on Artificial Intelligence*, pages 1–11. Springer, 2012.

[33] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[34] Princeton University. About wordnet. `https://wordnet.princeton.edu/`, 2010. Accessed: 2022-06-24.

[35] Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 252–259, 2003.

[36] Donna J Wood. Corporate social performance revisited. *Academy of management review*, 16(4):691–718, 1991.

[37] Magali A Delmas and Vanessa Cuerel Burbano. The drivers of greenwashing. *California management review*, 54(1):64–87, 2011.