DEPARTMENT OF INFORMATICS

PROGRAM OF POSTGRADUATE STUDIES IN DATA SCIENCE

Msc. Thesis

A data enrichment module based on NLP techniques for context analysis of web news

Fotini Kolokathi

DS3516007

Supervisor: Michalis Titsias

Corporate supervisor: Kostas Tsagkaris

Athens, October 2017

Acknowledgements

I would like to express my sincere gratidute to my supervisor Michali Titsia and my corporate supervisor Kosta Tsagkari from Incelligent for their continuing help and guidance throughout this thesis. Moreover, I would like to thank Aristoteli Margari, who works for Incelligent, because he made possible for me to carry out this thesis and paved the way with his acute remarks and clarifications. Finally, I would like to thank my family for their support.

Contents

Chapter 1

Introduction

1.1 Goal of the Thesis

The goal of this master thesis is to create a simple web service in which the client will give as input the name of a location and a time interval. Then, the server will give as response some possible texts in each of which an or more events are mentioned. These texts will have been published in Greek News websites in a date between the given time interval. The texts, that are expected to be extracted, will describe events in which there may be concentration of world and the use of devices that require network usage (e.g. mobile) may be possible by the people who will participate in them. Moreover, the server will return the possible dates each of the extracted events will happen. These dates will be extracted from the content of the texts.

The results of this assignment will be used from Incelligent company as input to an application that predicts possible signal problems given a location.

1.2 Software and Hardware Specifications

For the purpose of this assignment, the computer that was used has the following hardware specifications:

RAM (Random Access Memory): 8 GB

CPU (Central Processing Unit): Intel Core i7 6500U 2.5 GHz

OS (Operating System) : Windows 10

The programming language that was used is Python 3.5.2. Moreover, some packages from Anaconda 4.3.22 were used and the following libraries were installed too:

- nltk
- gensim
- newspaper
- dateparser

- datefinder
- imbalanced-learn
- python-yandex-translate

Some of these packages will be also mentioned in the following chapters.

1.3 Outline of the rest of this Thesis

The rest of this thesis is organized as follows:

- Chapter 2 presents the sources from which we made data crawling and the labels we used to annotate each text with one of them in order to deal with the problem by converting it to a classification task.
- Chapter 3 describes the different text representation models we tried.
- Chapter 4 presents the different classifiers we used, the parameters tuning we made for each of them and the experimental results we took from test data.
- Chapter 5 describes the web service, how we made test-data crawling given a particular location and how we extracted dates from texts that describe events.
- Chapter 6 concludes and proposes future directions.

Chapter 2

Data Crawling

2.1 Sources

In order to begin with our assignment, we had to find the sources from which we were going to extract data. For this purpose, we crawled 48 Greek News websites by using the *newspaper* Python library which was inspired by the requests HTTP library. *Newspaper* library enabled us to make News url gathering from websites. After that, we used these urls in order to extract texts from html by making a crawler for each website.

cnn.gr	*real.gr*	*naftemporiki.gr*	*in.gr*
newsbomb.gr	*news247.gr*	*thetoc.gr*	*gazzetta.gr*
protothema.gr	*newpost.gr*	*thestival.gr*	*multi-news.gr*
tovima.gr	*zougla.gr*	*alfavita.gr*	*iefimerida.gr*
documentonews.gr	*culturenow.gr*	*flashnews.gr*	*mononews.gr*
efsyn.gr	*altsantiri.gr*	*madata.gr*	*lifo.gr*
enikos.gr	*ert.gr*	*skai.gr*	*protagon.gr*

Figure 1.Some Greek news websites that were crawled

2.2 Data Format

Instead of extracting texts, we also needed to find the title of each text and the published date. So the format of the data that we extracted for each article is presented below:

{'article': 'Την προεδρία της Bundestag θα αναλάβει όπως όλα δείχνουν ο Βόλφγκανγκ Σόιμπλε σύμφωνα με τα γερμανικά ΜΜΕ. Την είδ
ηση μετέδωσε πρώτη η γερμανική εφημερίδα Bild και στη συνέχεια ο Γερμανικός Τύπος αλλά και οι Financial Times. Ο υπουργός Οικον
ομικών της Γερμανίας Βόλφγκανγκ Σόιμπλε είναι έτοιμος να αποχωρήσει από την θέση του υπουργού Οικονομικών της Γερμανίας και να
αναλάβει καθήκοντα προέδρου της Bundestag, της κάτω βουλής του γερμανικού κοινοβουλίου, επισήμανε ο Φόλκερ Κάουντερ, ο επικεφαλ
ής της κοινοβουλευτικής ομάδας του CDU της Άνγκελας Μέρκελ, σε ανακοίνωση που εξέδωσε. Σύμφωνα με την Bild, η κυρία Μέρκελ «παρ
ακάλεσε τον κ. Σόιμπλε να αναλάβει την προεδρία της Βουλή και ο κ. Σόιμπλε «έχει ήδη δείξει σημάδια προθυμίας» προς την ηγεσία
της Κοινοβουλευτικής Ομάδας της Χριστιανικής Ένωσης». Genau der Richtige in schweren Zeiten - #Schäuble soll #Bundestagspräside
nt werden https://t.co/a6p6aAaF1F — Sebastian Steineke (@SteinekeCDU) September 27, 2017 Σημειώνεται ότι ο σημερινός ΥΠΟΙΚ της
Γερμανίας έχει τη μακροβιότερη θητεία στο γερμανικό κοινοβούλιο, συμπληρώνοντας 45 χρόνια κοινοβουλευτικής παρουσίας. A Positiv
e sign for #Euro . #Schäuble to shift from German finance minister to Bundestag speaker https://t.co/6pIqdq3MJX — Falak Chattha
(@Chattha) September 27, 2017 Το παρασκήνιο Μετά τις εκλογές της Κυριακής, το συντηρητικό κόμμα της Μέρκελ παραμένει η μεγαλύτε
ρη δύναμη στο Κοινοβούλιο αλλά όχι τόσο ισχυρή αφού δεν μπορεί να συνεργαστεί με τους Σοσιαλδημοκράτες, και έχει πλέον ως μόνη
διέξοδο έναν συνασπισμό με τους FDP και τους Πράσινους. Οι τόνοι στη Βουλή αναμένεται πάντως να είναι υψηλοί κυρίως λόγω της πα
ρουσίας των ακροδεξιών της AfD οι οποίοι μπήκαν στο Κοινοβούλιο έπειτα από μισό και παραπάνω αιώνα. Η ανάληψη καθηκόντων στο υπ
ουργείο Οικονομικών από κάποιον εκλεκτό των FDP και όχι από τον Σόιμπλε θα δώσει την ευκαιρία στους Ελεύθερους Δημοκράτες να με
ιώσουν τους φόρους και επίσης να αντιταχθούν στα σενάρια της ολοκλήρωσης της Ευρωζώνης που πρότεινε ο Γάλλος Πρόεδρος. Η ηγεσία
του ΥΠΟΙΚ θα μπορούσε να είναι το αντάλλαγμα της συνεργασίας που θα προσφέρουν οι FDP στη Μέρκελ για την δημιουργία κυβερνητικο
ύ συνασπισμού. Δεν είναι τυχαίο ότι την περασμένη εβδομάδα, η Μέρκελ μιλώντας στη γιορτή για τα γενέθλια του Σόιμπλε που έγινε
75 ετών, απέτισε φόρο τιμής στα 45 έτη του ως μέλος του κοινοβουλίου, αλλά δεν έδωσε σαφές μήνυμα αν θα τον διατηρήσει στη θέση
του υπουργού Οικονομικών μετά τις εκλογές. Με πληροφορίες από Reuters/ ΑΠΕ',
 'publish_time': '2017-09-27',
 'title': 'Bild: Τέλος ο Σόιμπλε από το υπουργείο Οικονομικών | LiFO',
 'topic': ''}

Figure 2.Format of crawled data from Greek News websites

2.3 Label Annotation and Data Preprocessing

The approach that was followed in order to find articles that mention an or more events in which there may be great concentration of world, was to convert the problem to a classification task by annotating the extracted texts with labels.

The labels that were used are the following:

culture: It corresponds to cultural events such as concerts, festivals etc.

sport: It corresponds to sport events such as basketball matches, street races etc.

politics: It corresponds to political events such as pre-election talks etc.

other_event: It corresponds to other kind of events such as scientific events. The texts of this class are not 'homogeneous'.

no_event: It corresponds to articles that neither mention nor describe events.

The texts that were annotated with the first four labels are describing events with great popularity where the use of devices that require network usage is very possible, such as a concert. Also, most of these articles refer to the great presence of the world in the event.

The total number of articles that were annotated with the labels is equal to 3890 and the numbers of annotated articles per class are shown in the following table:

Class	Number of articles
culture	453
sport	187
politics	268
other_event	93
no_event	2889

The dataset has strong between-class imbalance with the class *no_event* to correspond to most of the articles in contrast with the other classes and especially class *other_event.*

The next step was to preprocess the data by converting the uppercase letters of words to lowercase and by removing stopwords[1], punctuations and numbers.

Then, 80% of the dataset was used as training data and the remaining 20% was used as test data.

[1] https://github.com/stopwords-iso/stopwords-en/blob/master/stopwords-en.txt
https://github.com/xtsimpouris/gr-nlp-law/blob/master/Greek%20Stopwords/stopwords.txt

Chapter 3

Text Representation Models

3.1 Visualizing data using t-SNE

The first step around any data related challenge is to start by exploring the data itself. This could be by looking at, for example, the distributions of certain variables or looking at potential correlations between variables.

The problem nowadays is that most datasets have a large number of variables. In other words, they have a high number of dimensions along which the data is distributed. Visually exploring the data can then become challenging and most of the time even practically impossible to do manually. However, such visual exploration is incredibly important in any data-related problem. Therefore it is important to understand how to visualise high-dimensional datasets. This can be achieved using techniques known as dimensionality reduction.

Traditional dimensionality reduction techniques such as Principal Components Analysis [1] and classical multidimensional scaling [2] are linear techniques that focus on keeping the low-dimensional representations of dissimilar datapoints far apart. For high-dimensional data that lies on or near a low-dimensional, non-linear manifold it is usually more important to keep the low-dimensional representations of very similar datapoints close together, which is typically not possible with a linear mapping.

T-Distributed Stochastic Neighbour Embedding(t-SNE) [3] is another technique for dimensionality reduction, which converts a high-dimensional data set into a matrix of pairwise similarities and is well suited for visualizing the resulting similar data. t-SNE is capable of capturing much of the local structure of the high-dimensional data very well, while also revealing global structure such as the presence of clusters at several scales.

t-SNE tries to match distributions and the way it does is computationally heavy. Since it scales quadratically in the number of objects N, its applicability is limited to data sets with only a few thousand input objects; beyond that, learning becomes too slow to be practical. So in case of very high dimensional data, it is essential to apply another dimensionality reduction technique before using t-SNE.

In our dataset, each sample corresponds to a text which belongs to a class. In order to examine the separability of the texts of different classes, we use t-SNE in order to reduce the dimensions of each sample to 2-dimensions so as to be able to visualize them by creating a

scatter plot of the two dimensions and colouring each sample by a colour that corresponds to its label.

So, for each of the following text representation techniques we create a scatter plot of two dimensions and colour each sample in order to see how separable the texts of different classes are before applying Machine Learning.

3.2 TF-IDF

In language processing, we often model documents using the ***bag-of-words*** model. In this model, we use a ***bag***, a.k.a a ***multiset***, to represent each document. The bag contains, for the document, the counts of words that occur in the document, disregarding order, syntax, and grammar.

We can understand the bag-of-words representation as a matrix in which the rows correspond to documents and the columns correspond to vocabulary words of the training data. For each document i, we count the number of times each vocabulary word w appears in the text and store it in a matrix X[i, j] as the value of feature j where j is the index of word w. Most values in X will be zeros since for a given document less than a couple thousands distinct words will be used, so the matrix will be very sparse.

Terms in large documents will display higher frequency counts, even if the documents talk about the same topics. To get around this problem we can go beyond frequency counts and use the tf-idf metric.

The tf metric stands for term frequency. The simplest way to calculate tf is as the raw frequency of a term in a document, i.e., the number of times the term t occurs in document d.

The idf metric stands for inverse document frequency. It measures how much information a term provides, that is, how rare a word w is in the language. We want frequent words of the text that are infrequent in the language to have large values (they are important):

$$idf(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

where D is the document corpus, N is the total number of documents in the corpus i.e N=$|D|$ and $|\{d \in D : t \in d\}|$ is the number of documents where the term t appears.

With idf we downscale weights for words that occur in many documents in the corpus and are therefore less informative than those that occur only in a smaller portion of the corpus.

So for each vocabulary term t, the vector contains its $tf - idf(t, d, D)$ score:

$$tf - idf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

In fact, scikit-learn (which is the library we use for calculating tf and idf scores) uses a slightly different variant:

$$tf - idf(t, d, D) = tf(t, d) \cdot (idf(t, D) + 1)$$

where we add one so that terms with zero idf i.e. that occur in all documents of a training set, will not be entirely ignored.

We also need to mention that during training we learn the vocabulary from the training dataset. The terms of the vocabulary will represent the features of the training and test data. Moreover, for each word t in the test dataset, the $tf(t, d)$ score equals to the frequency of this word to the test document d and the $idf(t, D)$ score has been learnt during training (so D corresponds to the training dataset) .

Our training dataset contains 89065 distinct words, so the tf-idf document-term matrix has 89065 columns. In order to visualize, each vector of the matrix, we first applied PCA to reduce the dimensions to 2000 as t-SNE is computationally quite heavy for high dimensional data. Then, we applied t-SNE to take 2-dimensional vectors and then visualize them by colouring each of them by a colour which corresponds to its label: **culture**, **no_event**, **sport** , **politics** and **other_event**.

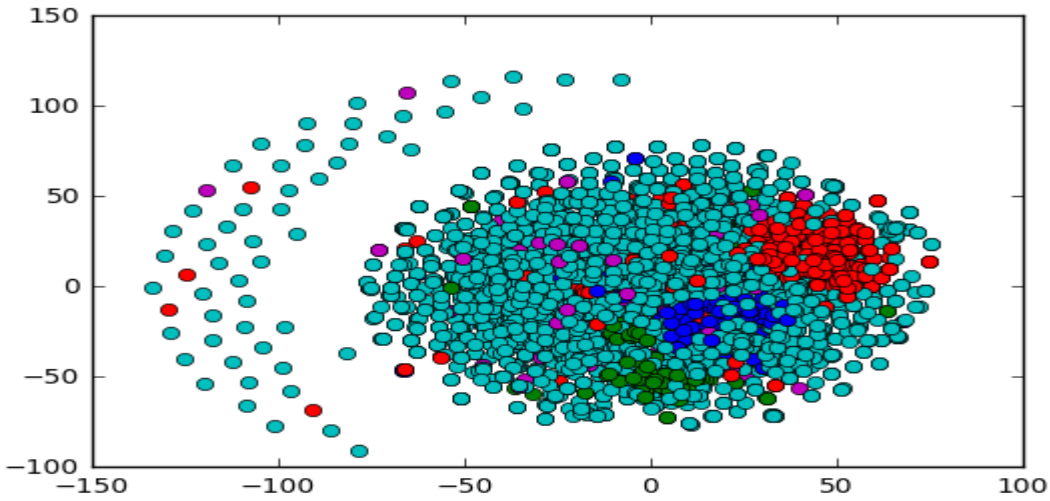


Figure 3 .t-SNE plot for visualizing the separability between the Tf-IDF text vectors of different classes

From the above scatterplot, we can see that there is “weak” separability between the points that correspond to texts of different classes. Moreover, the points of the texts which belong to classes with less observations overlap many points of the texts that correspond to no_event class.

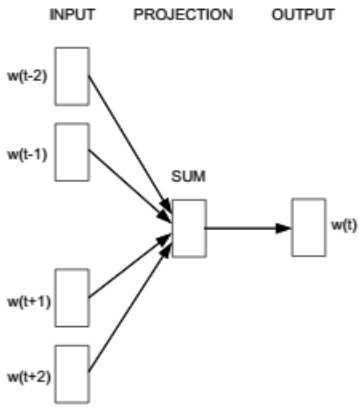
3.3 Word Embeddings

Traditionally, a document is represented as bag-of-words (BOW) vector i.e. a vector whose components (viewed as features) show which words are present in the text (for Boolean features), or the $TF - IDF$ (or other similar) scores of the words in the text, in both cases ignoring word order[6]. Unfortunately, BOW representation may lead to a large number of features, one for each vocabulary word [7]. Although standard feature selection (Information Gain, x^2) or dimensionality reduction techniques can solve this problem when we do not have a large number of features(otherwise they cannot be applied easily because of their computational complexity), another problem of BOW representations of texts is that they hardly capture the semantics of words or the distances between them[8].This means that words like "walk", "run" and "eat" are equally distant in spite of the fact that "walk" should be closer to "run" than "eat" semantically.

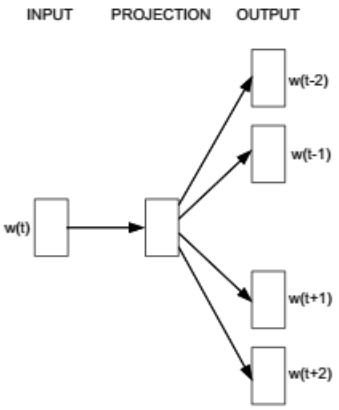
In recent years, several methods have been proposed [9,10,11,12,13,14] that map each vocabulary word to a *dense vector* of a space with a much lower dimensionality (often 100-300 dimensions in practical applications), so that words that occur in similar contexts are mapped to similar vectors (e.g. in terms of cosine similarity). Vectors of this kind, known as *continuous space word vectors* or *word embeddings*, have been found to capture morpho-syntactic and semantic properties of the corresponding words[15].So, in the previous example, "walk" and "run" will be very close to each other in the embedding space.

Different approaches have been proposed to compute them from large corpora. They include neural networks[11,16,17], dimensionality reduction on the word co-occurrence matrix [18] and explicit representation in terms of the context in which words appear[19].A study of Levy et. Al. [14] reveals that the hyperparameter optimizations and certain system design choices have a considerable impact on the performance of word embeddings, rather than the embedding algorithms themselves.

Word2vec is a tool which provides an efficient implementation of the continuous bag-of-words(CBOW) and skip-gram architectures for computing vector representations of words[20] in order to preserve the semantic and syntactic similarities between them. In the CBOW method, the goal is to predict a word given its past and future context, by averaging the contextual word vectors and then running a log-linear classifier on the averaged vector to get the resultant word[21]. Skip-gram is the converse: it predicts a window of contextual words given the current word. Also, the context is not limited to the immediate context, and training instances can be created by skipping a constant number of words in its context, for instance, $w_{i-3}, w_{i-4}, w_{i+3}, w_{i+4}$, hence the name skip-gram[21]. Both methods use artificial neural networks as their classification algorithm.



CBOW



Skip-gram

To obtain dense word vectors we applied *word2vec* to 18900 Greek News articles including the data that we have annotated with labels. Word embeddings were trained on the same type of data we have for our classification task as experimental results have shown that context is important when a classification task is at hand[8]. We used the 'skip-gram' model of word2vec and we set the dimensionality of the dense vectors to 300. We also set the context (window) size equal to 10.Context (window) size is equal to the number of words that would be included as context words before and after a given word. Punctuation symbols, brackets etc. were removed and all letters were converted to lower case. All words with total frequency lower than 5 were ignored. Because of that, although our vocabulary equals to 89065 words, we constructed 62241 word embeddings. Constructing them, took approximately 2 minutes.

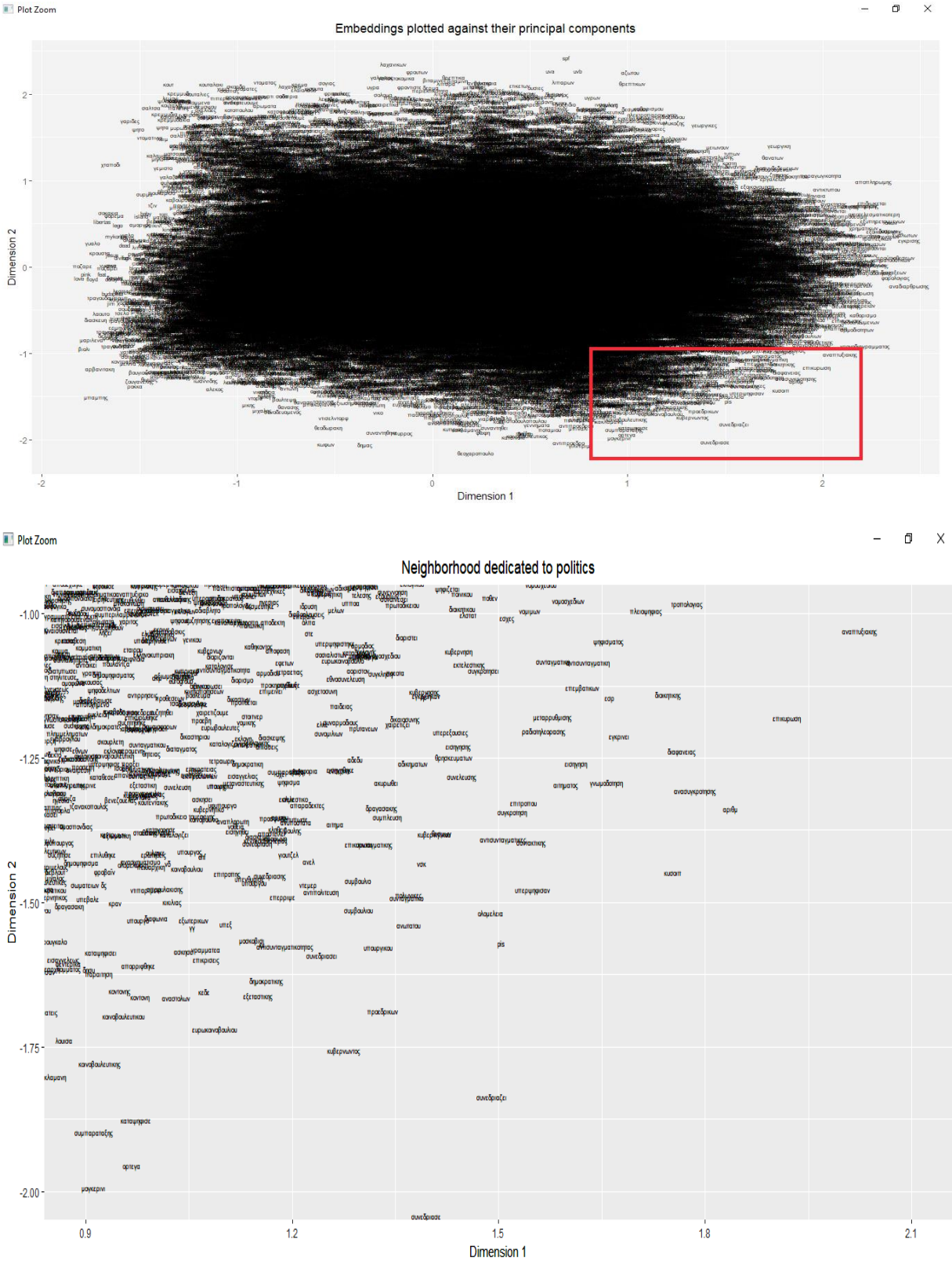


Figure 4.Embeddings plotted against their principal components. Contextual "neighborhoods" arise

Finally, we must mention that the following text representation models in which we use word embeddings, ignore word order.

3.3.1 Sum–of –Word-Vector

Having computed the word embeddings of all the vocabulary words, the simplest method to obtain a dense vector $\vec{t}$ (of the same dimensionality) for a text $\vec{t}$ = <w_1,w_2,...,w_n> of n consecutive word occurrences is to simply sum the dense vectors $\vec{w}_i$ of the word occurrences. Then we normalize the text vector with Euclidean norm so that all text vectors will have the same magnitude:

$$\vec{t} = \frac{\sum_{i=1}^{n} \overrightarrow{w_i}}{||\sum_{i=1}^{n} \overrightarrow{w_i}||}$$

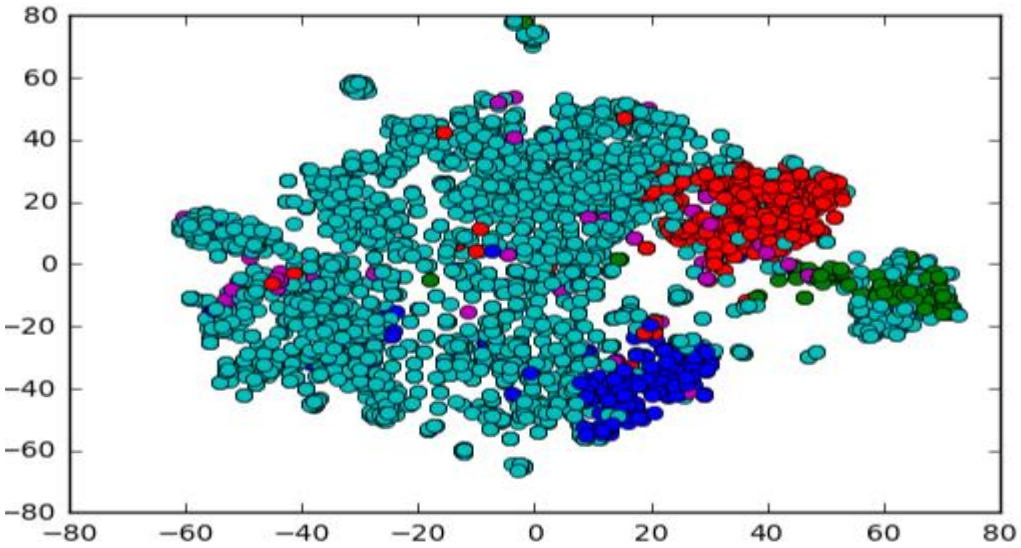


Figure 5. .t-SNE plot for visualizing the separability between the sum-of-word text vectors of different classes

3.3.2 Centroid

Another simple method to obtain a dense vector $\vec{t}$ (of the same dimensionality) for a text $\vec{t}$ = <w_1,w_2,...,w_n> of n consecutive word occurrences is to simply compute the centroid of the dense vectors $\vec{w}_i$ of the word occurrences[4,5,22]. Documents are projected in the word embedding space as the centroids of their words:

$$\vec{t} = \frac{1}{n} \sum_{i=1}^{n} \overrightarrow{w_i}$$

For the words of the vocabulary that we have not constructed word embeddings, we use zero vectors of 300 dimensions.

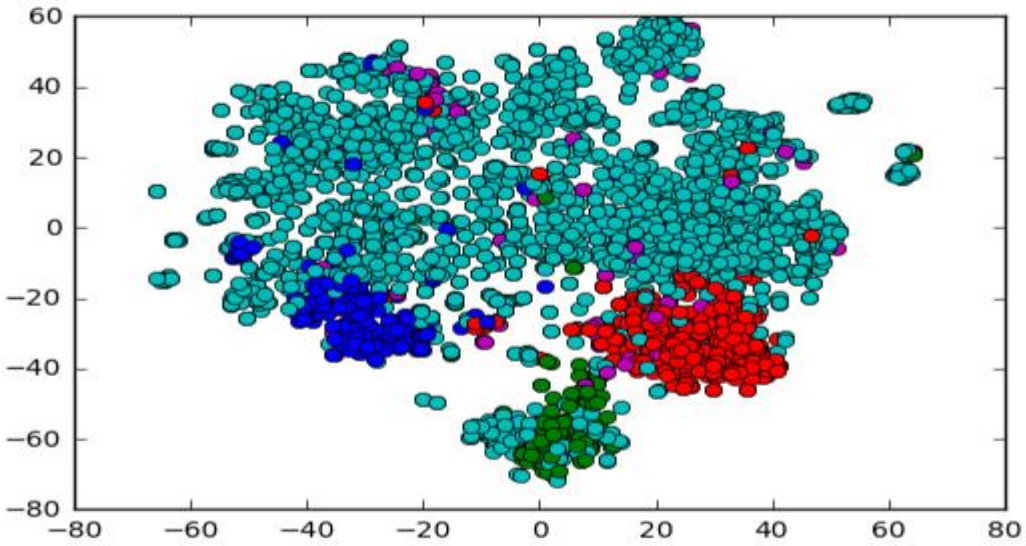


Figure 6. t-SNE plot for visualizing the separability between the centroid-text vectors of different classes

3.3.3 IDF-Weighted Centroid

We can also obtain a vector for a text $\vec{t}$ = <w_1,w_2,...,w_n> by including the inverse document frequencies $IDF(w_j)$ in the centroids of the texts as follows [4]:

$$\vec{t} = \frac{\sum_{j=1}^{|V|} \overrightarrow{w_j} \cdot TF(w_j, t) \cdot IDF(w_j)}{\sum_{j=1}^{|V|} TF(w_j, t) \cdot IDF(w_j)}$$

where $IDF(w_j) = \log \frac{|D|}{|D(w_j)|}$, $|D|$ is the total number of documents that are used as training data and $|D(w_j)|$ is those documents that contain the word w_j.

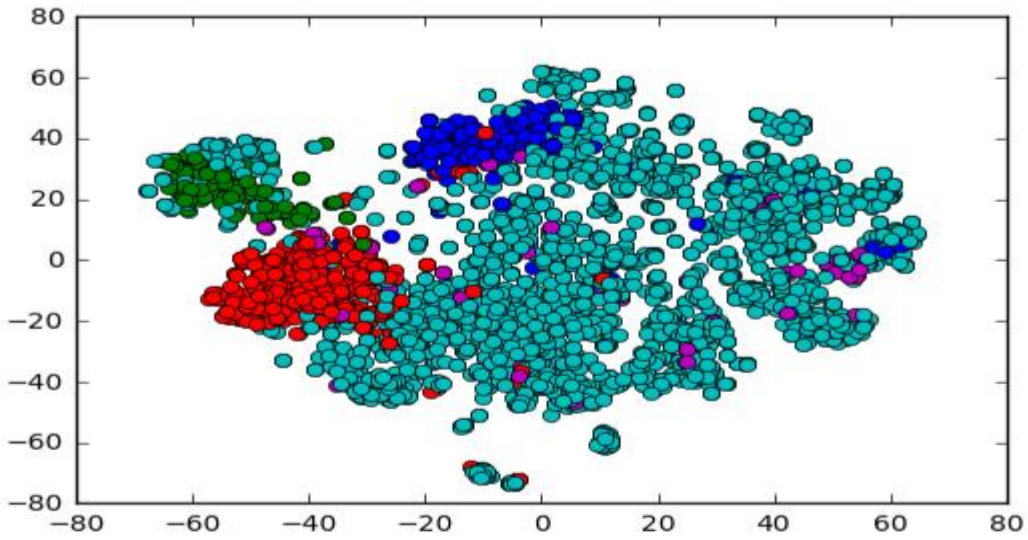


Figure 7. t-SNE plot for visualizing the separability between the idf-weighted centroid text vectors of different classes

3.3.4 Multivariate Gaussian Document Representation from Word Embeddings for Text Categorization

Now we model documents as multivariate Gaussian distributions [23]. Let D = {d_1, d_2, . . . , d_m} be a set of m documents. The vocabulary of the corpus V is extracted from both training and test data. To obtain a distributed representation for each word w ∈ V, we use the word embeddings that we extracted.

To generate a representation for each document, we assume that its words were generated by a multivariate Gaussian distribution. Specifically, we regard the embeddings of all words w present in a document as i.i.d. samples drawn from a multivariate Gaussian distribution:

$$\boldsymbol{w} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

where $\boldsymbol{w}$ is the distributed representation of a word w, **μ** is the mean vector of the distribution and **Σ** its covariance matrix.

We set **μ** and **Σ** to their Maximum Likelihood estimates, given by the sample mean and the empirical covariance matrix respectively. More specifically, the sample mean of a document corresponds to the centroid of its words, i. e. we add the vectors of the words present in the text and normalize the sum by the total number of words. For an input sequence of words d, its mean vector **μ** is given by:

$$\boldsymbol{\mu} = \frac{1}{|d|} \sum_{w \in \boldsymbol{d}} \boldsymbol{w}$$

where $|d|$ is the number of words of the text d. The empirical covariance matrix is then defined as:

$$\boldsymbol{\Sigma} = \frac{1}{|d|} \sum_{w \in d} (\boldsymbol{w} - \boldsymbol{\mu}) \cdot (\boldsymbol{w} - \boldsymbol{\mu})^T$$

Hence, each document is represented as a multivariate Gaussian distribution and the problem transforms from classifying textual documents to classifying distributions.

To measure the similarity between pairs of documents, we compare their Gaussian representations. More specifically, the similarity between two documents d_1 and d_2 is set equal to the convex combination of the similarities of their mean vectors $\boldsymbol{\mu_1}$ and $\boldsymbol{\mu_2}$ and their covariance matrices $\boldsymbol{\Sigma_1}$ and $\boldsymbol{\Sigma_2}$ .The similarity between mean vectors $\boldsymbol{\mu_1}$ and $\boldsymbol{\mu_2}$ is calculated using cosine similarity.

$$sim(\boldsymbol{\mu_1}, \boldsymbol{\mu_2}) \frac{\boldsymbol{\mu_1} . \boldsymbol{\mu_2}}{\|\boldsymbol{\mu_1}\| \cdot \|\boldsymbol{\mu_2}\|}$$

Where $\|.\|$ is the Euclidean norm for vectors. The similarity between the covariance matrices $\boldsymbol{\Sigma_1}$ and $\boldsymbol{\Sigma_2}$ can be computed using the following formula:

$$sim(\boldsymbol{\Sigma_1}, \boldsymbol{\Sigma_2}) \frac{\sum \boldsymbol{\Sigma_1} \circ \boldsymbol{\Sigma_2}}{\|\boldsymbol{\Sigma_1}\|_F \cdot \|\boldsymbol{\Sigma_2}\|_F}$$

Where (°) is the Hadamard or element-wise product between matrices (we sum over all its elements) and $\|.\|_F$ is the Frobenius norm for matrices. Hence, the similarity between two documents is equal to:

$$sim(d_1, d_2) = a \cdot \big(sim(\boldsymbol{\mu_1}, \boldsymbol{\mu_2})\big) + (1 - \alpha) \cdot (sim(\boldsymbol{\Sigma_1}, \boldsymbol{\Sigma_2}))$$

where $a \in [0,1]$.For the purpose of this assignment, we set a equal to 0.5.

For the training data we have a matrix in which, the number of rows and columns equal to the number of training documents. The element in row i and column j equals to the similarity between the train document i and the train document j. Each row represents the text vector of a document.

For the test data we have a matrix in which, the number of rows equals to the number of test documents and the number of columns equals to the number of training documents. The element in row i and column j equals to the similarity between the test document i and the train document j. Each row represents the text vector of a document.

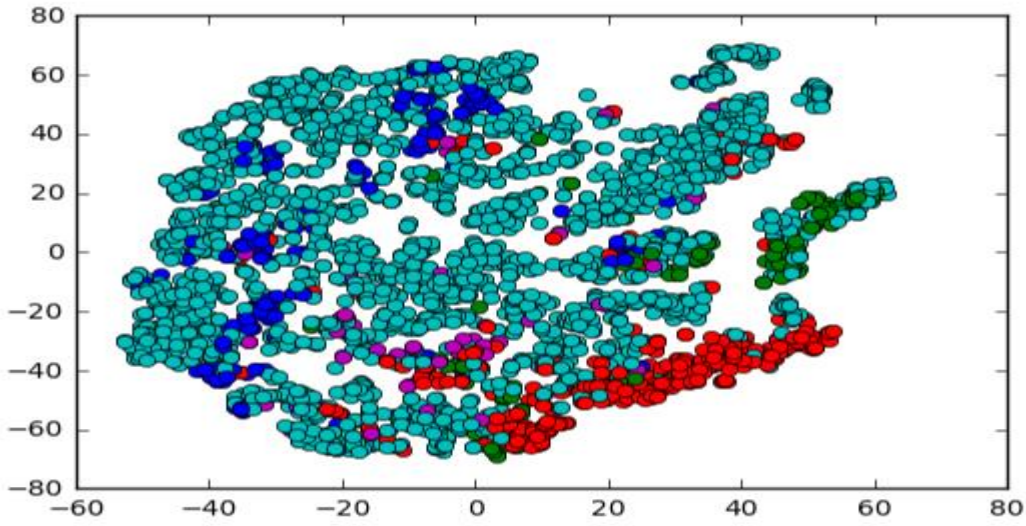


Figure 8.t-SNE plot for visualizing the separability between the Gaussian text vectors of different classes

3.3.5 Doc2Vec

All the text representation models that have been described until now lose word ordering. As a way to summarize bodies of text of varying length, Quoc Le and Tomas Mikolov came up with the *Paragraph Vector*, an unsupervised algorithm that learns fixed-length feature representations from variable-length pieces of texts, such as sentences, paragraphs, and documents [25]. The algorithm represents each document by a dense vector which is trained to predict words in the document. Its construction gives to the algorithm the potential to overcome the weakness of word order losing.

The idea behind Paragraph vector is straightforward: we act as if a paragraph (or document) is just another vector like a word vector. We determine the embedding of the paragraph in vector space in the same way as words. Our paragraph vector model considers local word order like bag of n-grams, but gives us a denser representation in vector space compared to a sparse, high-dimensional representation [25].

To obtain dense document vectors, we used *doc2vec* which modifies the word2vec algorithm to unsupervised learning of continuous representations for larger blocks of text**.** Since the Doc2Vec class extends gensim's original Word2Vec class, many of the usage patterns are similar. We can easily adjust the dimension of the representation, the size of the sliding window, the number of workers, or almost any other parameter that we can change with the Word2Vec model. The one exception to this rule are the parameters relating to the training method used by the model. In the word2vec architecture, the two algorithm names are **"Continuous Bag Of Words"** (CBOW) and **"Skip-gram"** (SG); in the doc2vec architecture, the corresponding algorithms are **"Distributed Memory"** (DM) and **"Distributed Bag Of Words"** (DBOW).

In **DM**, the paragraph vectors are obtained by training a neural network on the fake task of inferring a center word based on context words and a context paragraph. A paragraph is a context for all words in the paragraph, and a word in a paragraph can have that paragraph as a context[25].In other words, **DM** attempts to predict a word given its previous words and a paragraph vector. Even though the context window moves across the text, the paragraph vector does not (hence distributed memory) and allows for some word-order to be captured .In **DBOW**, the paragraph vectors are obtained by training a neural network on the fake task of predicting a probability distribution of words in a paragraph given a randomly-sampled word from the paragraph. In other words, **DBOW** predicts a random group of words in a paragraph given only its paragraph vector.

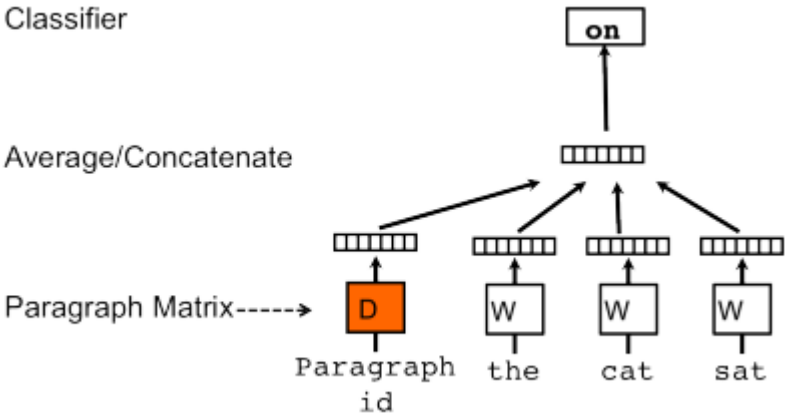


Figure 9. Distributed Memory (DM) version of paragraph vectors.DM attempts to predict a word given its previous words and a paragraph vector

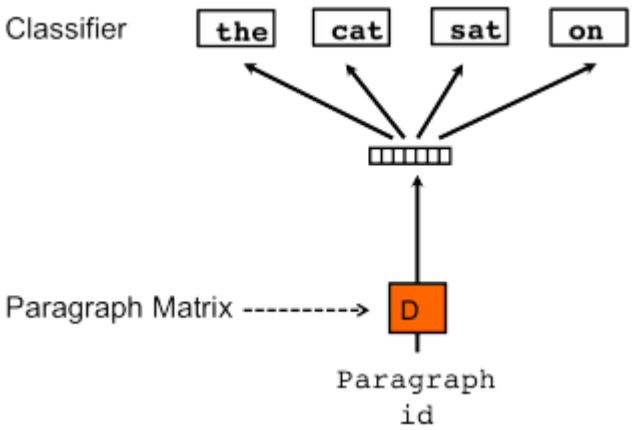


Figure 10.Distributed Bag of Words(DBOW) version of paragraph vectors. In this version, the paragraph vector is trained to predict the words in a small window.

In our case, each document vector is created by concatenating two vectors, one from **DBOW** and one from **DM** model. In each model, the learned vector representations have 100 dimensions. So after concatenation, the learned vector representations will have 200 dimensions.

In **DBOW** model, we set the window size ,which is the maximum distance between the predicted word and the context words used for prediction within a document, equal to 10 and we ignore words with total frequency less than 2.We also set the hyperparameter *negative* equal to 10.So the model will be trained with negative sampling. With negative sampling, we modify the optimization objective so that each training sample updates only a small percentage of the model's weights. This technique has shown to improve the quality of the resulting word vectors as well. In **DM** model, we set the window size equal to 5 and we ignore words with total frequency less than 2.We also set hyperparameter *negative* equal to 10.Then we train our models in order to take vectors for training documents. During training, the learning rate decreases over the course of several iterations.

Then, we can "infer" a vector for any text by using each of the mentioned models. So, for each test sample we "infer" a vector from the DBOW model and a vector from the DM model and then we concatenate them. So, the text vectors of test data have 200 dimensions.

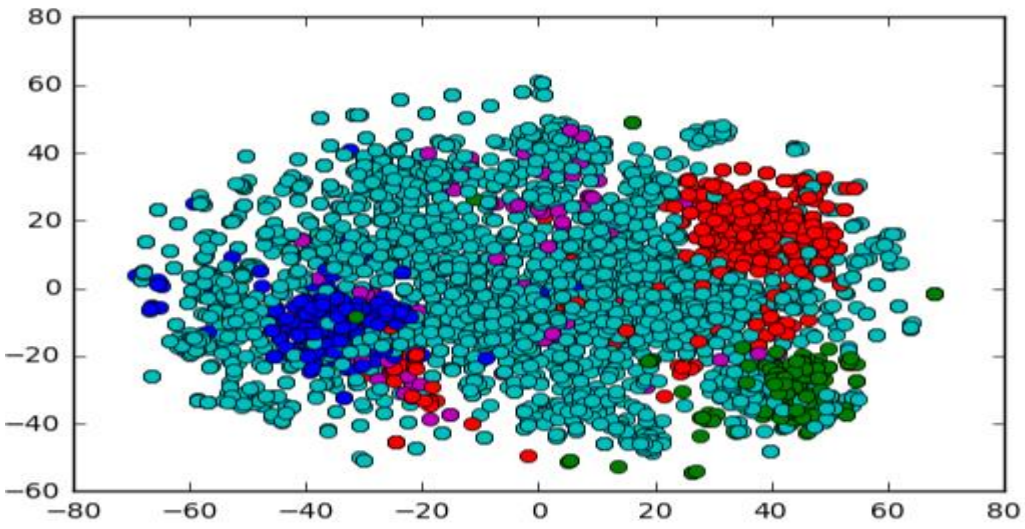


Figure 11.t-SNE plot for visualizing the separability between doc2vec text vectors of different classes

Chapter 4

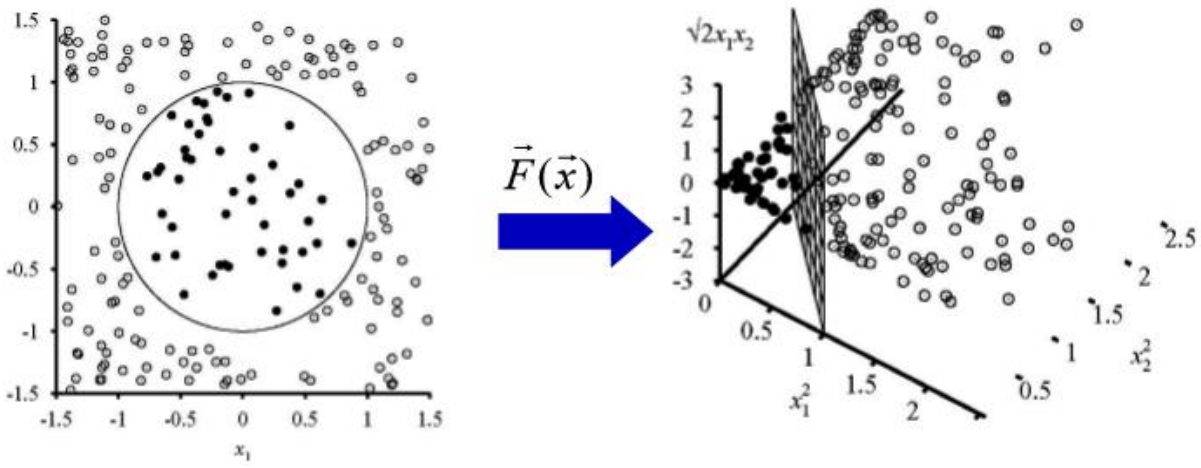
Text Classification

4.1 Classifiers and parameters tuning

Now we present the classifiers that we used for our classification task. Moreover, for each classifier, we mention the hyperparameters that we tuned in order to take better results.

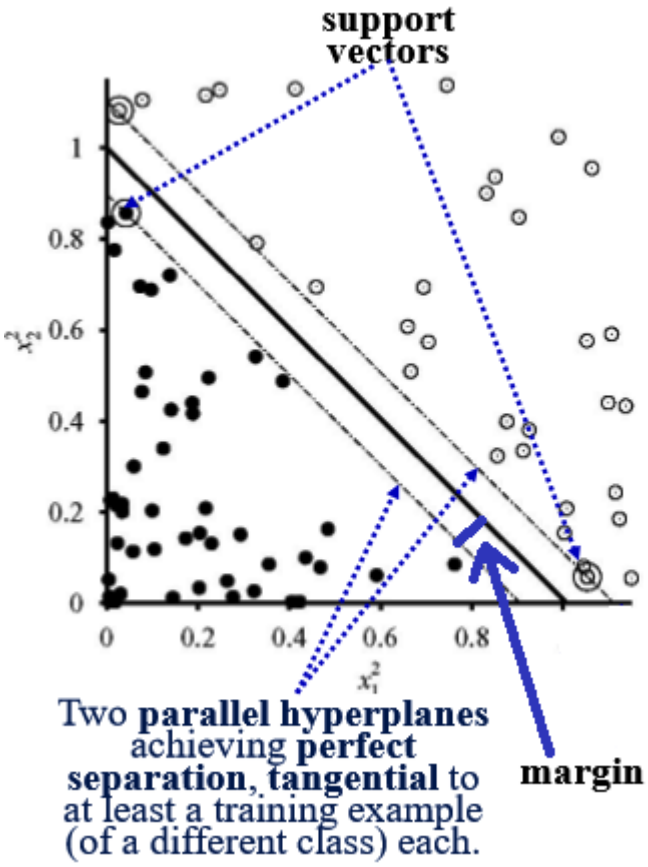
Support Vector machines (SVM)

With an appropriate transformation, an originally non-linearly separable dataset may become linearly separable.



In the example above: $\vec{F}(\vec{x}) = <x_1{}^2, x_2{}^2, \sqrt{2} \cdot x_1 \cdot x_2>$.

SVMs search in the new vector space for a hyperplane that separates the examples, with the maximum margin.



In the previous figure, the separating hyperplane (bold line) is between the two tangential hyperplanes. Maximizing the margin leads to better generalization over the entire population.

- Equation of the separating hyperplane:

$$\vec{w} \cdot \vec{F}(\vec{x}) + b = 0$$

- For simplicity, we require the tangential hyperplanes to have the following equations:

$$\vec{w} \cdot \vec{F}(\vec{x}) + b = \pm 1$$

Then the margin is : $\frac{2}{\|\vec{w}\|}$

- Minimization problem: $\min_{\vec{w},b} \frac{\|\vec{w}\|^2}{2}$

such that : $(\vec{w} \cdot \vec{F}(\vec{x_j}) + b) \cdot y_j \geq 1$

A training example

its correct class (here +/-1)

We require all the training examples to be on the correct sides and outside the margin.

In practice, even if the data are linearly separable (in some space!) the SVMs are trained by allowing errors in the classification of the training data. This is achieved by introducing slack variables $\boldsymbol{\xi_j}$:

$$\min_{\vec{w},b,\xi} C\sum_{j=1}^{N}\xi_j + \frac{\|\vec{w}\|^2}{2}$$

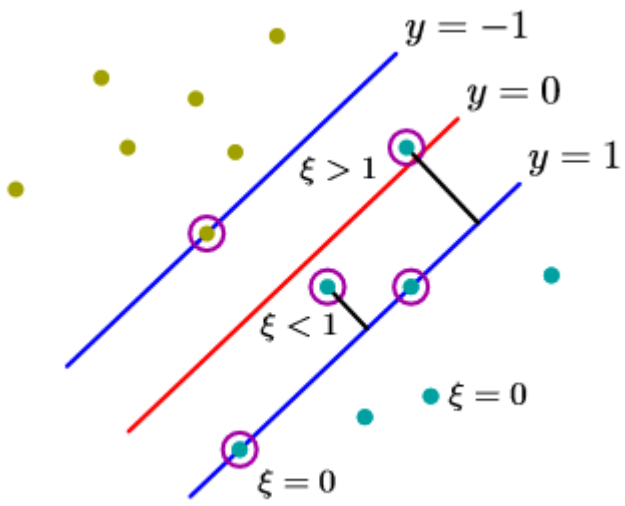
$$\text{subject to: } \left(\vec{w}\cdot\vec{F}\left(\vec{x_j}\right)+b\right)\cdot y_j \geq 1-\xi_j$$

$$\xi_j \geq 0$$

Interpretation of each slack variable:

- ξ_j = 0 : Data point is correctly classified and either it satisfies the margin or it lies in the correct side.
- $0 < \xi_j \leq 1$: Data point is classified correctly but it lies between the decision boundary and the margin .
- $\xi_j > 1$: Data point is misclassified.

C is a regularization parameter that determines how much we want the slack variables to go to zero.



SVM uses a function $K(\vec{x_i},\vec{x_j})$ that computes the inner product $\vec{F}(\vec{x_i})\cdot\vec{F}\left(\vec{x_j}\right)$ in some new vector space, where a transformation F takes us. This function is called **kernel**.

In scikit-learn, an estimator for classification is a Python object that implements the methods fit(X,y) and predict(T). The class **sklearn.svm.svc** implements support vector classification. The constructor of an estimator takes as arguments the hyperparameters of

the model. The hyperparameters that we tune in order to improve the performance of the classifier are:

C : the regularization parameter that mentioned before.

kernel: Specifies the kernel type to be used in the algorithm. The available choices are:

- linear : $\langle x,x'\rangle$.
- poly: $(\gamma\cdot\langle x,x'\rangle+r)^d$. d is specified by the degree[2] hyperparameter, r by the coef0[3] hyperparameter.
- rbf: $\exp(-\gamma\cdot||x-x'||^2)$. γ is specified by the hyperparameter *gamma*, must be greater than 0.
- sigmoid: $\tanh(\gamma\langle x,x'\rangle+r)$, where r is specified by the hyperparameter *coef0*.

gamma: Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

In the second chapter we mentioned that there is a strong between-class imbalance. Most of the articles correspond to no_event class.

There are some techniques that can help us to deal with imbalanced datasets. One way to fight this issue is to generate new samples in the classes which are under-represented. The most naive strategy is to generate new samples by randomly sampling with replacement the current available samples. Several authors [26], [27] agree that random over-sampling can increase the likelihood of occurring overfitting, since it makes exact copies of the minority class examples. Maybe, if we had more data, we could use this technique.

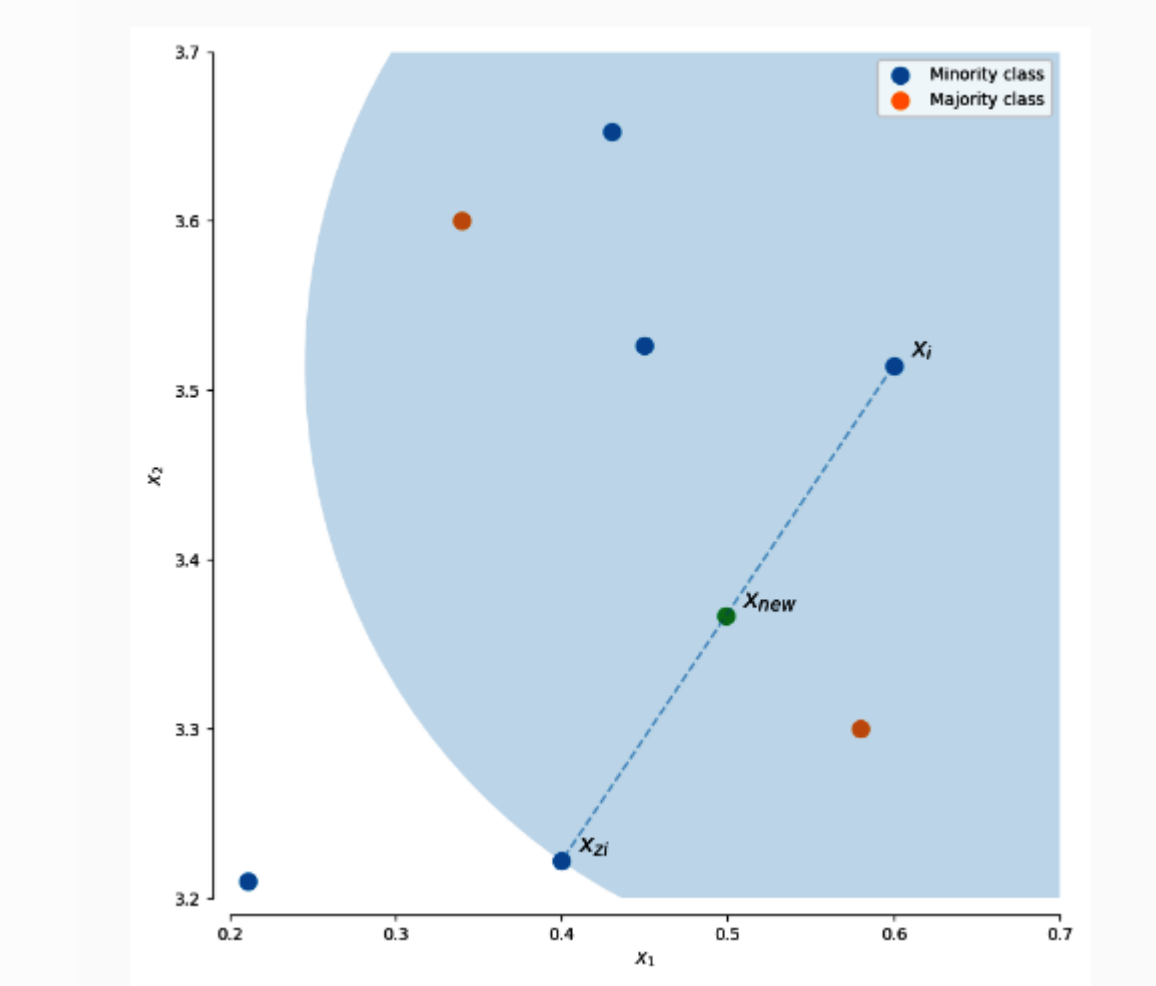
A popular method to over-sample minority classes is called **Synthetic Minority Oversampling Technique (SMOTE)**[26]. Its main idea is to form new minority class examples by interpolating between several minority class examples that lie together.

Considering a sample x_i , a new sample x_{new} will be generated considering its k nearest-neighbours of the same class. For instance, the 3 nearest-neighbours are included in the blue circle as illustrated in the figure below. Then, one of these nearest-neighbours x_{zi} is selected and a sample is generated as follows:

$$x_{new} = x_i + \lambda \cdot (x_{zi} - x_i) \text{ , where } \lambda \in [0,1].$$

This interpolation will create a sample on the line between x_i and x_{zi} as illustrated in the image below:

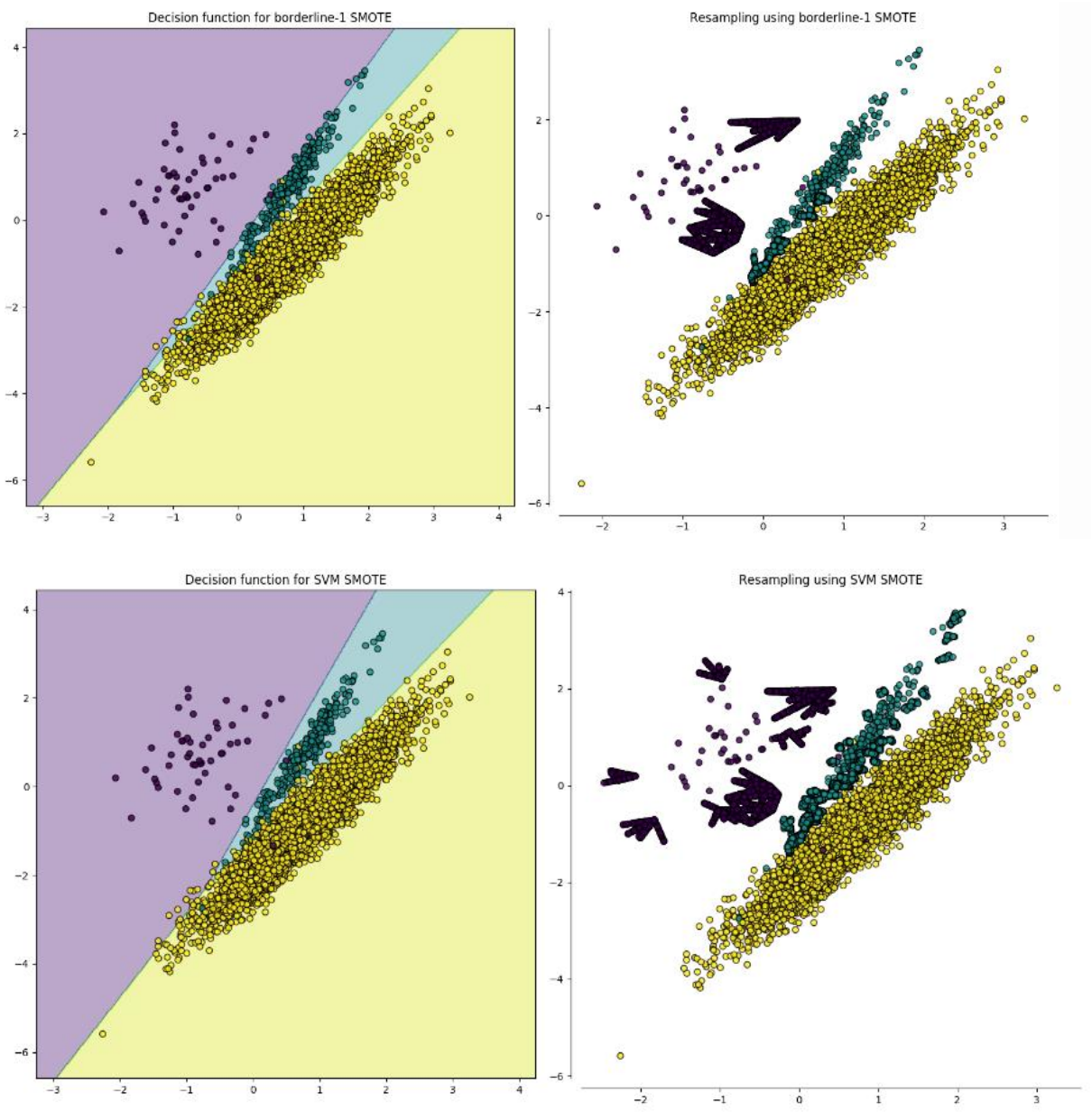
[2] Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.
[3] Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.



In our problem we use two SMOTE variant techniques:

- **Borderline-1** SMOTE: The **borderline** SMOTE when instantiating a SMOTE object — will classify each sample x_i to be (i) noise (i.e. all nearest-neighbours are from a different class than the one of x_i), (ii) in danger (i.e. at least half of the nearest neighbours are from the same class than x_i or (iii) safe (i.e. all nearest neighbors are from the same class than x_i). **Borderline** SMOTE will use the samples *in danger* to generate new samples. In **Borderline-1** SMOTE, x_{zi} will belong to the same class with sample x_i.
- **SVM** SMOTE: **SVM** SMOTE when instantiating a SMOTE object — uses an SVM classifier to find support vectors and generate samples considering them. Note that the **C** parameter of the SVM classifier allows to select more or less support vectors.

After adding new samples with each of the mentioned techniques (the samples of the classes culture, sport, other_event and politics will be 500 now), we train a SVM classifier at each new dataset. We call **SM-B1** the SVM classifier that is trained on the dataset in which its new samples were the result of **Borderline-1** SMOTE technique and we call **SM-SVM** the SVM classifier that is trained on the dataset in which its new samples were the result of **SVM** SMOTE technique.



Neural network

We also created a *two-hidden* layer *neural network*. Particularly, we first normalized the features X_i because it is easier to find the minimum of the loss function when the contours are almost circular. Otherwise, we need to take smaller steps (especially in SGD) to avoid going to fast along a dimension and getting into the plateau. We also initialized all the weights to small random numbers by taking sample from *glorot normal* distribution and we applied *l2-norm* regularization to penalize large values of the weights W in order to avoid overfitting. So, if L is the loss function, after regularization we have:

$$L' = L + \lambda \cdot \frac{\|W\|^2}{2}$$

We also used *relu* activation function to the hidden layers.

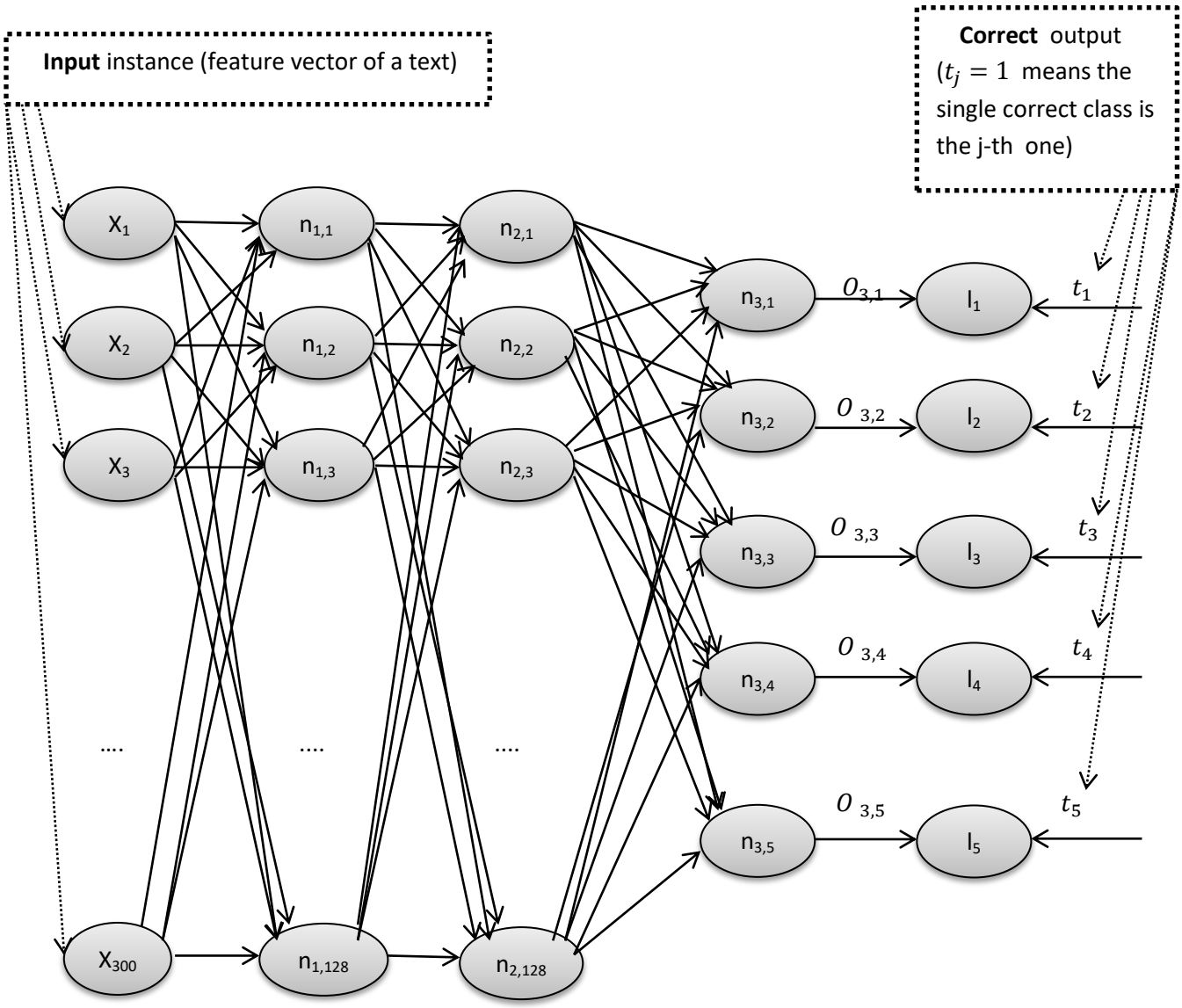
Moreover, each mini-batch is *scaled by the mean/variance* computed on just that mini-batch. This adds some noise to the values $z^{[l]} = w^{[l]} \cdot x^{\{t\}} +$ b within that minibatch, where l is the corresponding layer, $w^{[l]}$ are the weights of this layer, t is the number of minibatch, $x^{\{t\}}$ is the minibatch and $z^{[l]}$ is the output of the layer l . This technique has a regularization effect and it helps to speed up learning.
We also used *dropout* which is a regularization technique which reduces overfitting in neural network by preventing complex co-adaptations on training data.With dropout we set a percentage of activations (outputs of layers) to zero. We repeat it for different combinations of activations, and at the end we take the average of the results of them. So the network can never rely on any given activation because they might be squashed. So dropout makes possible for the network to learn a redundant representation for everything to make sure that at least some of the information remains.

Next, we choose the optimizer which is the algorithm which carries out the learning process in the neural network i.e Stohastic Gradient Descent(SGD).The learning process is formulated as searching for a parameter vector of weights, at which the loss function takes a minimum value.

The results of the output layer are the probabilities for the current instance to belong to each class. We use softmax function at the output layer in order to take these probabilities. These probabilities and the real probabilities t_j are then used at the *cross entropy loss function*.

The framework that we used for creating and training the neural network was Keras. The hyperparameters that we tuned are the number of neurons of the hidden layers, the kind of the distribution we took sample for the weights, the learning rate value, the percentage of the activations that were set to zero (dropout_rate), the kind of activation functions at the hidden layers, the batch size and the optimizer algorithm.

We tuned the hyperparameters of the neural network for almost all the text representation models. Then we took the values of the parameters that fit most of them. Particularly, we use 128 neurons at each hidden layer, we sample weights from glorot normal distribution centered on zero, we set learning rate equal to 0.001 and we do not use dropout. We also use *relu* activation function at the hidden layers and we set the batch size equal to 256. Finally, we use the Adam optimizer algorithm. We call this model **NN**.



$$\vec{o}^{(1)} = \boldsymbol{relu}\left(W^{(1)}\vec{x}\right)$$

$$\vec{o}^{(2)} = \boldsymbol{relu}\left(W^{(2)} \cdot \vec{o}^{(1)}\right)$$

$$\vec{o}^{(3)} = \boldsymbol{softmax}\left(W^{(3)} \cdot \vec{o}^{(2)}\right)$$

$$\boldsymbol{l} = -\sum_{j=1}^{5} \boldsymbol{t_j} \cdot \log \boldsymbol{o_{3,j}}$$

Cross entropy loss at the **current training** instance

Two-hidden Neural Network

Logistic Regression

Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a **logistic function**:

$$f(x) = \frac{1}{1 + e^{-x^T \cdot w - c}}$$

The implementation of logistic regression in scikit-learn can be accessed from class *LogisticRegression*. This implementation can fit binary, One-vs- Rest, or multinomial logistic regression with optional L2 or L1 regularization.

L2 regularized logistic regression solves the following optimization problem:

$$\min_{w,C} \|w\|_2 + C \cdot \sum_{i=1}^{n} \log(\exp\left(-y_i\left(X_i{}^T \cdot w + c\right)\right) + 1)$$

The solvers which carry out the learning process (solvers are algorithms that search for a parameter vector $\overrightarrow{W}$ at which the loss function takes the minimum value) implemented in the class LogisticRegression are the “liblinear”, “newton-cg”, “lbfgs”, “sag” and “saga”:

The solver “liblinear” uses a coordinate descent (CD) algorithm. The “lbfgs”, “sag” and “newton-cg” solvers only support L2 penalization and are found to converge faster for some high dimensional data. The “sag” solver uses a Stochastic Average Gradient descent. It is faster than other solvers for large datasets, when both the number of samples and the number of features are large. The “saga” solver is a variant of “sag”. The “saga” solver is often the best choice. The “liblinear” solver is used by default for historical reasons.

The hyperparameters that we tune in order to improve the performance of the classifier are:

C: Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

solver: the algorithms that we mentioned before and which carry out the learning process.

tol: Tolerance for stopping criteria.

max_iter: Useful only for the newton-cg, sag and lbfgs solvers. Maximum number of iterations taken for the solvers to converge.

We call this classifier **LR**.

Stohastic Gradient Descent

Stohastic Gradient Descent (SGD) is a technique that is used to find the minima of a function. Scikit-learn provides the *SGDClassifier* which is a linear classifier (SVM, logistic regression, a.o.) that uses SGD for training (that is, looking for the minima of the loss using SGD).According to the documentation[4] of scikit-learn, this estimator implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). SGD allows minibatch learning too.

This implementation works with data represented as dense or sparse arrays of floating point values for the features. The model it fits can be controlled with the loss parameter; by default, it fits a linear support vector machine (SVM). The regularizer is a penalty added to the loss function that shrinks model parameters towards the zero vector using either the squared euclidean norm L2 or the absolute norm L1 or a combination of both.

The hyperparameters that we tune in order to improve the performance of the classifier are:

loss: The loss function to be used. The possible options are 'hinge', 'log', 'modified_huber', 'squared_hinge','perceptron'. If the loss function is the hinge then linear SVM is used. The 'log' loss gives logistic regression, a probabilistic classifier. 'modified_huber' is another smooth loss that brings tolerance to outliers as well as probability estimates. 'squared_hinge' is like hinge but is quadratically penalized. 'perceptron' is the linear loss used by the perceptron algorithm.

alpha: Constant that multiplies the regularization term.

tol: The stopping criterion of the iterations.

n_iter : The number of passes over the training data (aka epochs)

max_iter : The maximum number of passes over the training data (aka epochs).

We call this classifier **SGD**.

Random Forest

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and

[4] http://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier

control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if the hyperparameter *bootstrap*=True.

The hyperparameters that we tune in order to improve the performance of the classifier are:

n_estimators: The number of trees in the forest.

Criterion: The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

bootstrap: Whether bootstrap samples are used when building trees.

We call this classifier **RF**.

Ensembles of Classifiers

Finally we used the naïve version of sklearn.ensemble.VotingClassifier in order to create an ensemble of the SVM (Support Vector Machine) , RD (Random Forest) and SGD(Stohastic Gradient Descent) classifiers.

Before using VotingClassifier, we find the best hyperparameters of each classifier with tuning. The idea behind the VotingClassifier is to combine conceptually different machine learning classifiers and use a majority vote.

In majority voting, the predicted class label for a particular sample is the class label that represents the majority of the class labels predicted by each individual classifier.

E.g., if the prediction for a given sample is

- classifier 1 -> class 1
- classifier 2 -> class 1
- classifier 3 -> class 2

the VotingClassifier (with voting='hard') would classify the sample as "class 1" based on the majority class label.

In the cases of a tie, the VotingClassifier will select the class based on the ascending sort order. E.g., in the following scenario

- classifier 1 -> class 2
- classifier 2 -> class 1

the class label 1 will be assigned to the sample. In our case, if all the classifiers predict different classes for a sample, the class that is chosen is the predicted class of SGD.

We call this model **ENS**.

4.2. Metrics

The goal of this classification task is to find all the events even if we have some wrong-classified texts to one of the culture, sport, politics and other_event classes. So the main measure is macro-averaged recall. Moreover, we calculate macro-averaged $F1$ (macro $F1$) and macro-averaged precision (macro precision).Macro-averaged $F1$ is a combination (harmonic mean) of macro-averaged precision (macro precision) and macro-averaged recall (macro recall). These measures are defined as follows. The precision (P_i) of a class C_i is the number of test instances correctly classified in C_i (true positives,TP_i), divided by the number of test instances the classifier placed in C_i (true positives and false positives,$TP_i + FP_i$). The recall (R_i) of a class C_i is the number of test instances correctly classified in C_i (TP_i), divided by the number of test instances that truly belong in C_i(true positives and false negatives, $TP_i + FN_i$). The $F1$ score of C_i combines P_i and R_i.

$$P_i = \frac{TP_i}{TP_i + FP_i} \quad , \quad R_i = \frac{TP_i}{TP_i + FN_i} \quad , \quad F_{1,i} = \frac{2 \cdot P_i \cdot R_i}{P_i + R_i}$$

Macro precision and macro recall average P_i and R_i, over all the $|C|$ classes.

$$Macro\ Precision = \frac{1}{|C|}\sum_{i=1}^{|C|} P_i \quad , \quad Macro\ Recall = \frac{1}{|C|}\sum_{i=1}^{|C|} R_i$$

Macro F1 combines macro precision and macro recall.

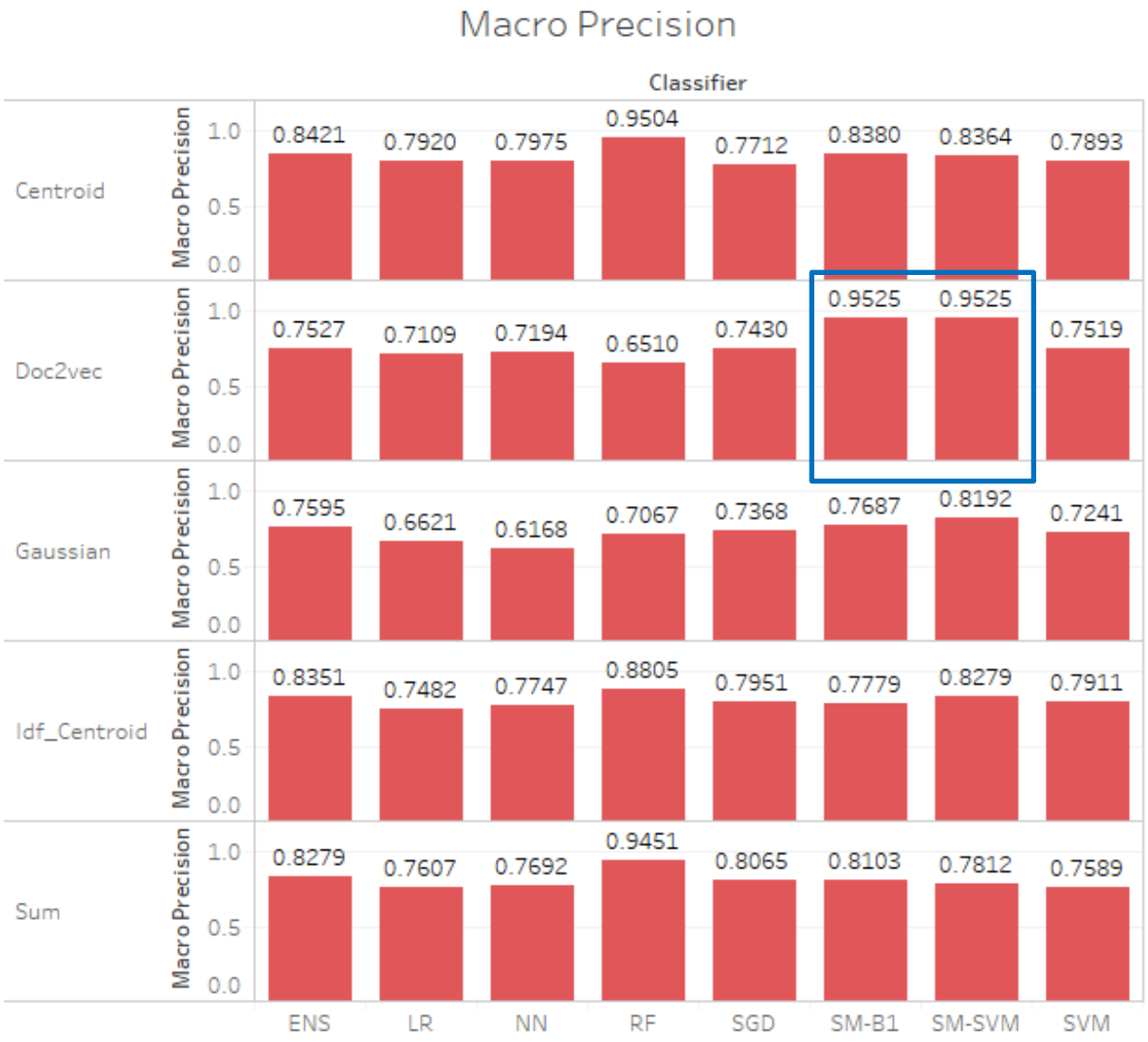
$$Macro\ F1 = \frac{2 \cdot Macro\ precision \cdot Macro\ Recall}{Macro\ precision + Macro\ Recall}$$

We do not use micro-averaged measures because they assign greater importance to classes with more test instances; consequently, rare classes do not influence much the results. So, micro average is a measure of how effective the classifier is on the large classes in the collection. By contrast, the macro-averaged measures assign equal importance to all classes. So, macro average is a measure of how effective the classifier is on the small classes. Finally, we do not use accuracy as metric because our class labels are not uniformly distributed.

4.3 Classifiers Performance

Now, we will present the results of the Precision, Recall and F-measure metrics for each classifier and text representation model.

Precision of a class is the metric which shows **how many** of the instances **classified in the class** ($TP_i + FP_i$) **are true members** of the class (TP_i).



For the **centroid** model, RF gives 95.04% which is the best macro-precision score. Then, ENS, SM-B1 and SM-SVM follow it with their macro-precision scores to be equal to 84.21%, 83.80% and 83.64% respectively. In general, all the classifiers give good macro-precision scores but RF is superior to others.

For the **doc2vec** model, both SM-B1 and SM-SVM give 95.25% as macro-precision score which is the best score. The other classifiers do not give so good macro-precision scores compared to SM-B1 and SM-SVM.

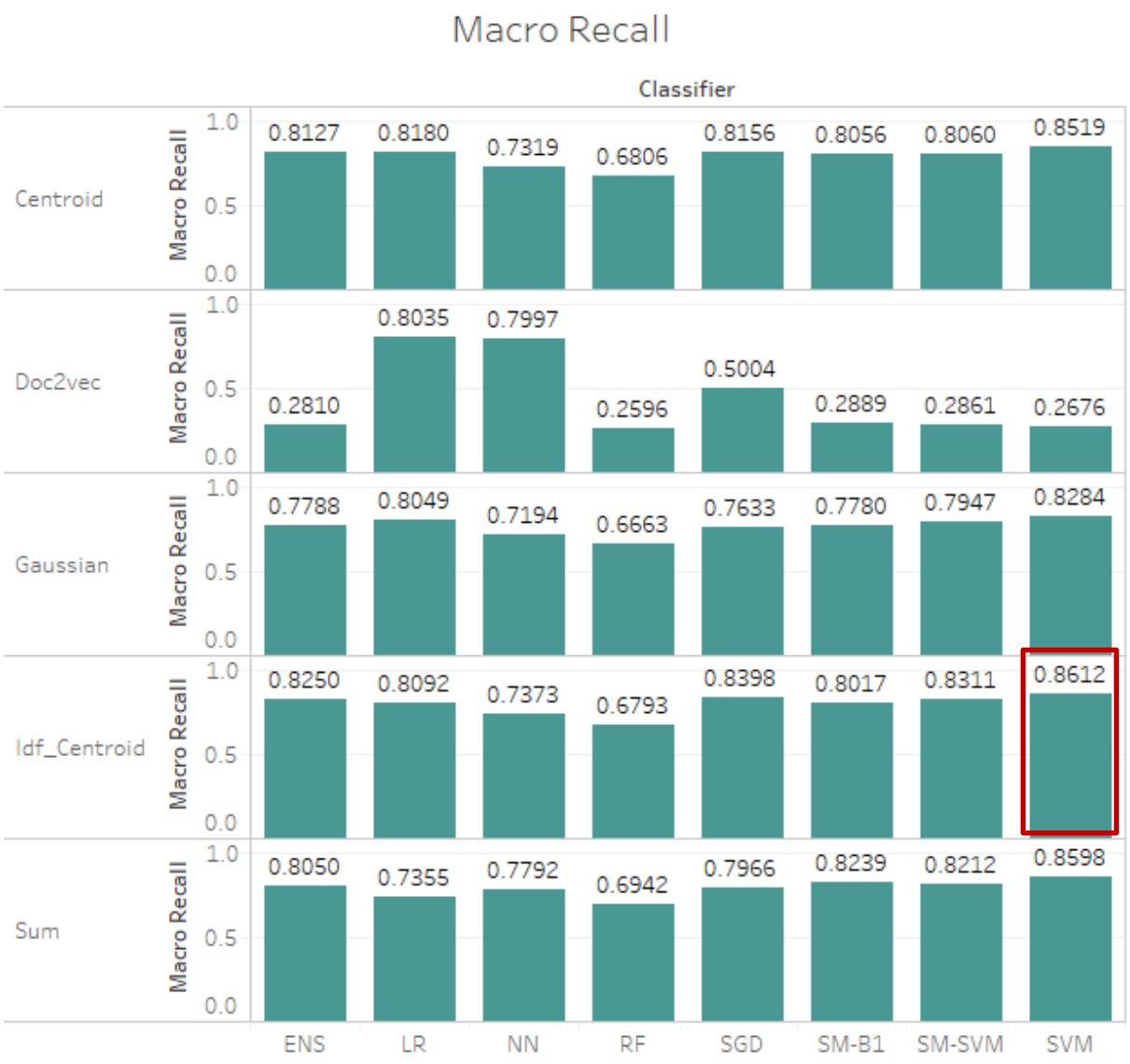
For the **Gaussian** model, SM-SVM gives the best macro-precision score which is equal to 81.92%.The scores of the other classifiers are between 60%-77%.

For the **Idf_Centroid** model, RF gives the best macro-precision score which is equal to 88.05%. Then, ENS and SM-SVM follow it with their macro-precision scores to be equal to 83.51% and 82.79% respectively.

For the **Sum** model, RF gives the best macro-precision score which is equal to 94.51%. Then, ENS, SGD and SM-B1 follow it with their macro-precision scores to be equal to 82.79%, 80.65% and 81.03% respectively.

Finally, the best macro-precision score which is equal to 95.25% is given by both SM-B1 and SM-SVM classifiers when the text representation model is the **doc2vec**.

Recall of a class is the metric which shows **how many** of the **true members** of a class ($TP_i + FN_i$) are **classified in the class** (TP_i).



For the **centroid** model, SVM gives the best macro-recall score which is equal to 85.19%. Then, the classifiers with higher scores are LR with 81.8%, SGD with 81.56, ENS with 81.27%, SM-SVM with 80.6%. and SM-B1 with 80.56%.Also, the macro-recall score of NN is equal to 73.19% and the macro-recall score of RF is equal to 68.06%.

For the **doc2vec** model, LR gives the best macro-recall score which is equal to 80.35. Then, NN follows it with its macro-recall score to be equal to 79.97%.The rest classifiers give lower macro recall scores that are between 28%-50%.

For the **Gaussian** model, SVM gives the best macro-recall score which is equal to 82.84%.Then, LR gives the second higher macro –recall score which is equal to 80.49% and

SM-SVM follows it with its macro-recall score to be equal to 79.47%.The macro-recall scores of the rest classifiers are between 66%-78%.

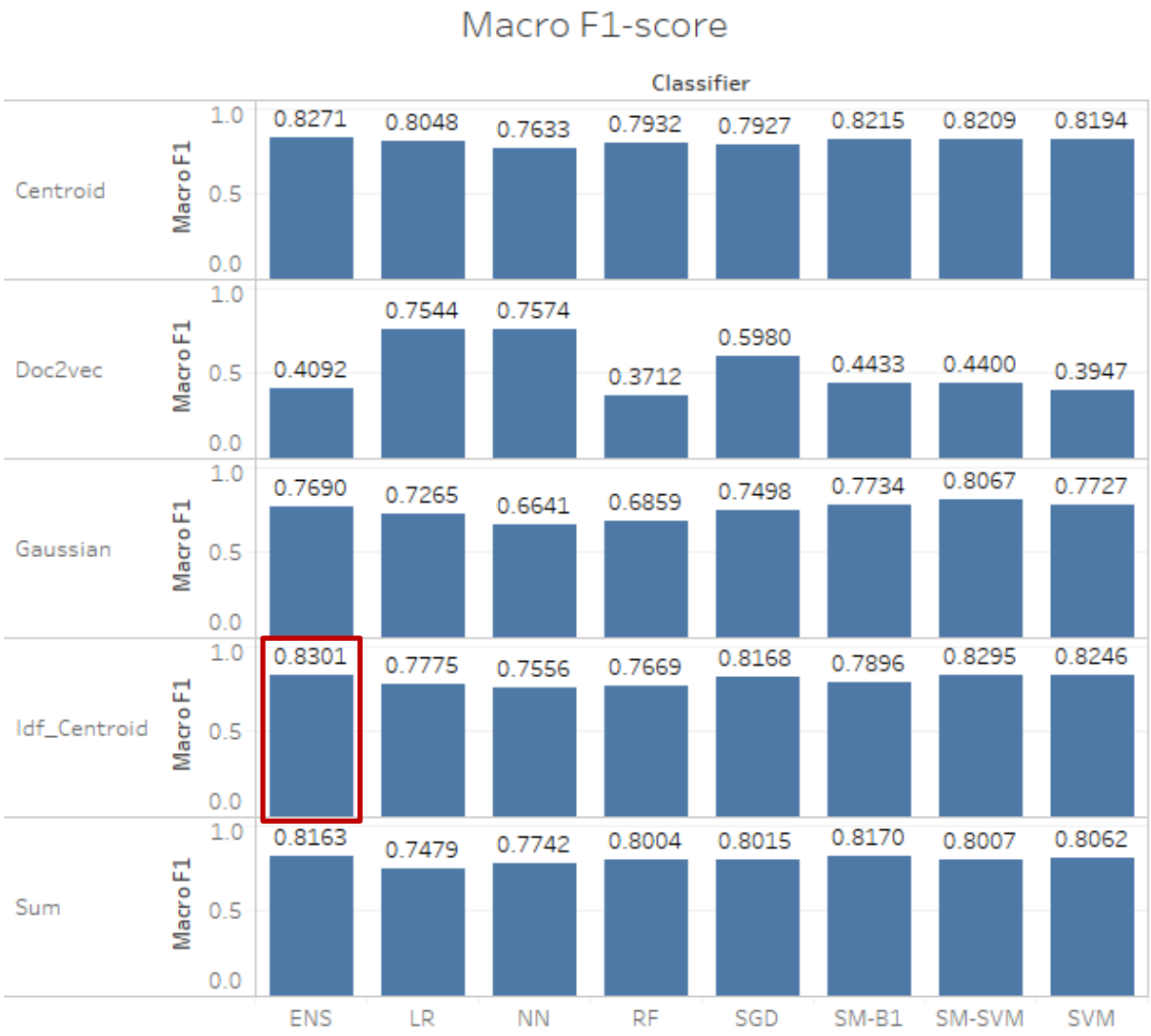
For the **Idf_Centroid** model, SVM gives the best macro-recall score which is equal to 86.12%.Then, SGD follows it with 83.98% , SM-SVM with 83.11%, ENS with 82.5%, LR with 80.92% and SM-B1 with 80.17%.RF has the lower macro-recall score which is equal to 67.93%.

For the **Sum** model, SVM gives the best macro-recall score which is equal to 85.98%. Then, SM-B1 follows it with its score to be equal to 82.39%, SM-SVM with 82.12%, ENS with 80.5% and SGD with 79.66%.The macro-recall scores of the rest three classifiers are between 69%-78%.

From the previous figure, we can see that most of the classifiers give high macro-recall scores(>80%) for **Centroid**, **Sum** and **Idf_Centroid** text representation models. SVM gives the highest macro recall scores for all the text representation models, except for the doc2vec. Also, there are very small differences between the macro-recall scores SVM classifier gives for **Centroid**, **Sum** and **Idf_Centroid** text models. Maybe, if we had more data, the differences between them would have been more obvious.

Finally, the best macro-recall score which is equal to 86.12% is given by SVM classifier when the text representation model is the **Idf_Centroid**.

Macro-F1 score is the harmonic average of the macro-precision and the macro-recall and reaches its best value at 1 (perfect precision and recall) and worst at 0.



For the **Centroid** model, ENS classifier gives the best macro-f1 score which is equal to 82.71%.Then, SM-B1 follows it with its macro-f1 score to be equal to 82.15% ,SM-SVM with 82.09%, SVM with 81.94% and LR with 80.48%.The macro-f1 scores of the rest classifiers are between 76%-79%.

For the **Doc2vec** model, NN gives the best macro-f1 score which is equal to 75.74% and the second best macro-f1 score is given by LR classifier and is equal to 75.44%. The rest classifiers give lower macro-recall scores which are between 37%-60%.

For the **Gaussian** model, SM-SVM gives the best macro-f1 score which is equal to 80.67%.The macro-f1 scores of the rest classifiers are between 66%-77%.

For the **Idf_Centroid** model, ENS classifier gives the best macro-f1 score which is equal to 83.01%.The following classifiers are the SM-SVM with macro-f1 score equal to 82.95%, SVM with 82.46% and SGD with 81.68%.The macro-f1 scores of the rest classifiers are between 75%-79%.

For the **Sum** model, SM-B1 classifier gives the best macro-f1 score which is equal to 81.7%. Then, ENS follows it with macro-f1 score equal to 81.63% , SVM with 80.62% , SGD with 80.15% , SM-SVM with 80.07% ,RF with 80.04%, NN with 77.42% and LR with 74.79%.

From the previous figure, we can see that most of the classifiers have high macro-f1 scores(>80%) for **Centroid**, **Sum** and **Idf_Centroid** text representation models. Finally, the best macro-f1 score which is equal to 83.01% is given by ENS classifier when the text representation model is the **Idf_Centroid**.

4.3.1 Ensembles of Classifiers

We also tried to create ensembles of classifiers so that the predicted classes will depend on the classifiers and on the weights that each of them had. So, we had to find the most appropriate combination of the weights by using cross validation.

First, we created a list of tuples. Each tuple contained a candidate combination of weights which their sum was equal to one. Each weight was a decimal number with one decimal place. We excluded the tuples: (0.5, 0.5, 0.0), (0.5, 0.0, 0.5) and (0.0, 0.5, 0.5).We also decided to create an ensemble of three classifiers. Each weight of the tuple corresponded to a particular classifier.

In order to find the best combination of the weights, we used cross validation. First, we divided the dataset into 5 parts and we preserved the class ratios in all of them. We kept the last part for test data. So, for each tuple of weights:

1. We performed 4 iterations with the remaining parts.
2. In each iteration:

   a. We used a different part as test data and the other three parts as training data. We trained the three classifiers and we took the predictions of them for the test data.

   b. We made three arrays with n rows, in which n was equal to the number of test data. The number of columns was equal to the number of classes and each class corresponded to a particular column.

   c. Each array corresponded to a classifier. In each row of each array, we set the number 1 to the column that corresponded to the predicted class of the classifier and all

the other elements of the row took the value zero. Then we multiplied each array with the weight of the classifier it corresponded to.

d. After that, we created an array F which was equal to the sum of the three arrays and we created a list with equal size with the number of rows of the arrays. Each element of the list was equal to the number of the column with the maximum value at the corresponding row of the array F. So, we created a list with the predicted values.

e. Then we calculated macro recall score.

3. At the end, we calculated the average of the macro recall scores over the iterations.

Finally, we can find the tuple of the weights that gave the maximum average macro recall score.

The three classifiers we tried to create an ensemble of were the SVM, SM-SVM and RD. The corresponding weights that gave the maximum average macro recall score were 0.6, 0.0 and 0.4. So, we cannot create an ensemble of these classifiers because it will keep only SVM predictions.

Then, we tried to create an ensemble of SVM, RD and NN. The corresponding weights that gave the maximum average macro recall score were 0.6, 0.2 and 0.2. So, we cannot create an ensemble of these classifiers because it will keep only SVM predictions.

In both cases, we represented each text by calculating its centroid.

4.4 Final Model

Now we will present the best values of the hyperparameters we tuned for the models with the best scores per metric. Moreover, we will show the confusion matrices of these models in order to explain more clearly the reason we choose the macro recall score as the main metric for our classification task.

In each row of the confusion matrix, we can see how the classifier distributes the true members of a class to all the classes during prediction. Each column corresponds to a particular class and shows how many of the instances classified in this class are true members of the class(True Positives or TP) and how many are wrongly classified to it (False Positives or FP).

As we mentioned before, the best macro-precision score which is equal to 95.25% is given by both **SM-B1** and **SM-SVM** classifiers when the text representation model is the **doc2vec**.

The hyperparameters of **SM-B1** are:

```
C=50.5271797012363
gamma=0.022124526598737985
kernel='poly'
```

Table 1.Confusion matrix of SM-B1

	culture	no_event	other_event	politics	sport
culture	18	**87**	0	0	0
no_event	0	**571**	0	0	0
other_event	0	**16**	1	0	0
politics	0	**36**	0	6	0
sport	0	**39**	0	0	3

The hyperparameters of **SM-SVM** are:

```
C=10.368599944120893
gamma=0.0087359804761899269
kernel='rbf'
```

Table 2.Confusion matrix of SM-SVM

	culture	no_event	other_event	politics	sport
culture	19	**86**	0	0	0
no_event	0	**571**	0	0	0
other_event	0	**16**	1	0	0
politics	0	**36**	0	6	0
sport	0	**40**	0	0	2

At the confusion matrices of the SM-B1 and SM-SVM we can see that the numbers of false positives for all the classes except class no_event are equal to zero. This means that the precision of these classes is equal to 1.Only in class no_event there are false positives. This is the reason we have high macro precision score after averaging the precision scores for all the classes. Moreover, macro precision is not a good metric for this classification task, because we lose many real events which are classified to no_event class.

The best macro-recall score which is equal to 86.12% is given by **SVM** classifier when the text representation model is the **Idf_Centroid**.

The hyperparameters of **SVM** are:

```
C=19.916382677694301
gamma=0.26317770249370487
kernel='rbf'
```

Table 3.Confusion matrix of SVM

	culture	no_event	other_event	politics	sport
culture	97	**3**	3	2	0
no_event	6	**542**	11	6	6
other_event	2	**2**	11	1	1
politics	1	**5**	0	35	1
sport	1	**1**	0	0	40

At the confusion matrix of the SVM we can see that the numbers of false negatives for all the classes are very few. If we see the column that corresponds to no_event class we can see that the number of wrongly classified instances is equal to 11.Compared to the two previous models which have high macro precision scores, this model which has the highest macro recall score loses less real events. So, although we may have some instances that are wrongly classified to the classes culture, sport, politics and other_event, this model detects most of the real events.

The best macro-f1 score which is equal to 83.01% is given by **ENS** classifier when the text representation model is the **Idf_Centroid**. As we have mentioned, **ENS** is a classifier that combines **SVM**, **SGD** and **RD** classifiers.

We presented the hyperparameters of **SVM** for **Idf_Centroid** text representation model previously. Now we will present the hyperparameters of the other 2 classifiers.

The hyperparameters of **SGD** are:

```
alpha=0.0005
max_iter=100
n_iter=60
tol=0.0001
```

The hyperparameters of **RD** are:

```
bootstrap=False
criterion='entropy'
n_estimators=50
```

Table 4.Confusion matrix of ENS

	culture	no_event	other_event	politics	sport
culture	93	**8**	2	2	0
no_event	5	**555**	4	3	4
other_event	2	**3**	9	2	1
politics	1	**5**	0	35	1
sport	1	**3**	0	0	38

At the confusion matrix of the ENS classifier, we can see that there are 19 instances of real events that are wrongly classified to class no_event. These instances are more compared to the corresponding instances of the previous model which has the maximum macro recall score. So, this model loses more real events compared to the previous model but it loses less real events compared to the first two models that have high macro precision scores.

As a result, we choose the model with the best macro-recall score. So, we will use the **Idf_centroid** text representation model with the **SVM** classifier for this classification task.

Now, we present the learning curves of the SVM classifier for the Idf_centroid text model.

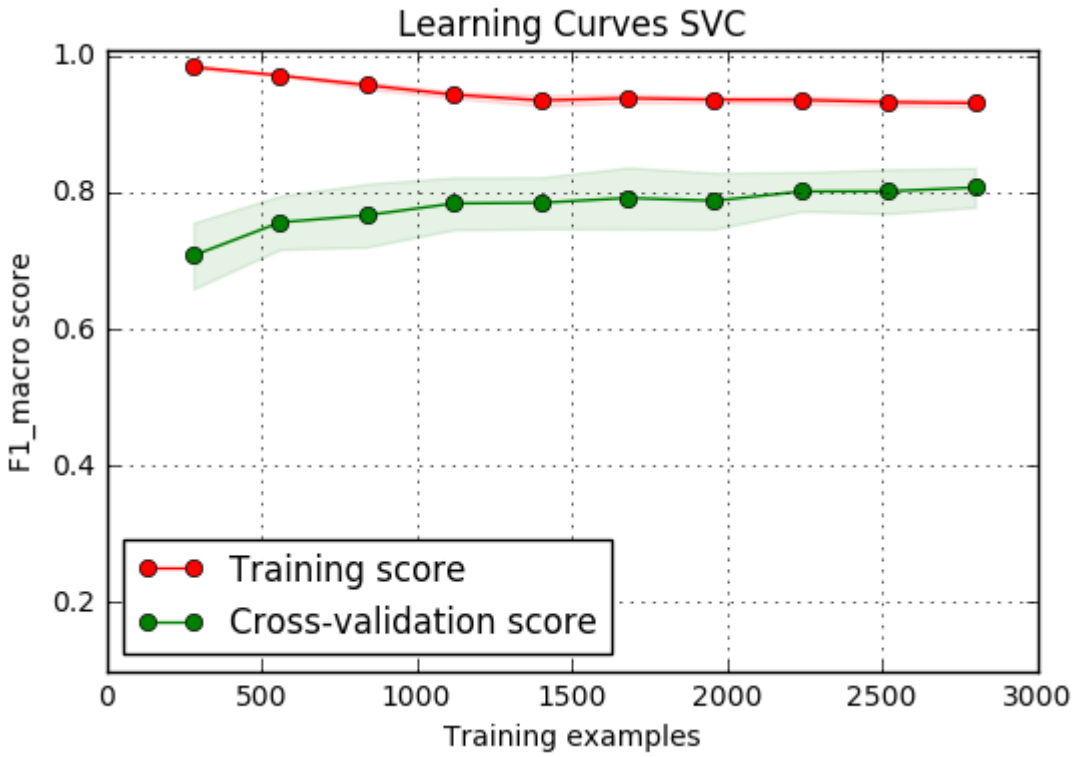


Figure 12.Best Model Learning Curves

We can clearly see that the training score decreases slowly and the cross validation score increases slowly as the number of training data increases. There is a difference between the macro F1 training score and the macro F1 validation score for the same number of training examples, so we may not have underfitting problem. Moreover, our model is simple so we may not have overfitting problem. It seems that by increasing the number of training examples, the validation score could be increased.

Chapter 5

Web service

5.1 Description

The final step was to create a web service in which one or more clients will send requests and the server will be responsible for responding to each of them. Server binds a socket with a specific IP/port and waits for connection requests at this port. Then, the client who knows the IP/port server listens, tries to connect with it. After the connection, server binds a new socket to the same local port so as to be able to receive new requests at the same port. Moreover, server starts in a background thread which then starts one more thread for each client request. This means that server can respond to multiple client requests at the same time.

The dataset that was used in order to find the best model, it is now used as training data. Server downloads test data each time it receives a request with a name of a location and a time interval.

5.2 Test Data Retrieval based on Location Name

In order to retrieve texts that are related to a location, we used Google News. As you can see below, if we give a name of a location in Google News browser, most of the news articles that are retrieved contain this location name or words that are very close to it ,such as Ιωάννινα-Γιάννενα, at the title or/and inside the text:

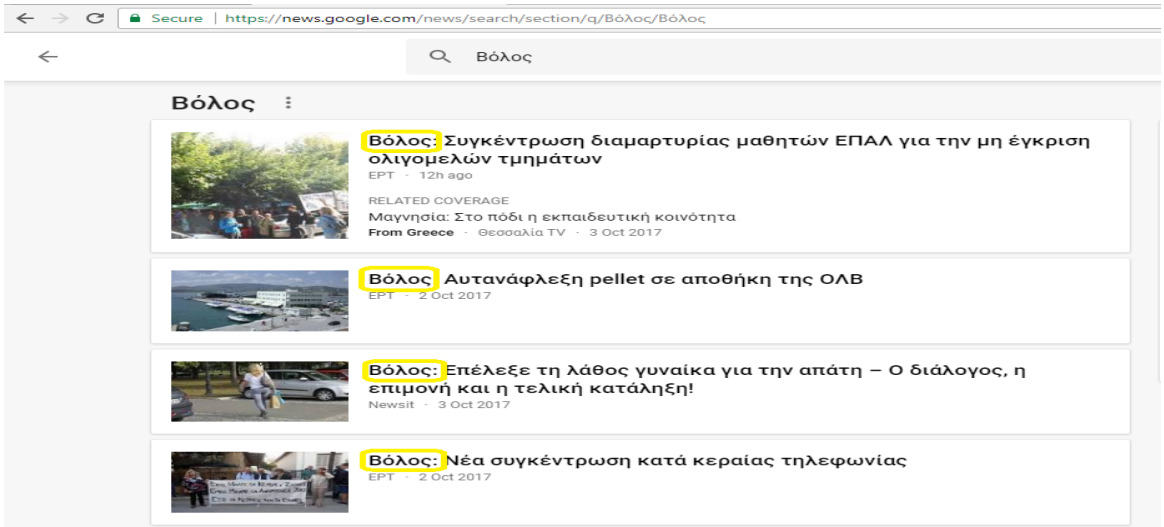


Figure 13.Articles that were retrieved based on a given location

If we give a location name and the domain name of a website at the Google News browser, then most of the retrieved articles will contain the given location name and will belong to this website:

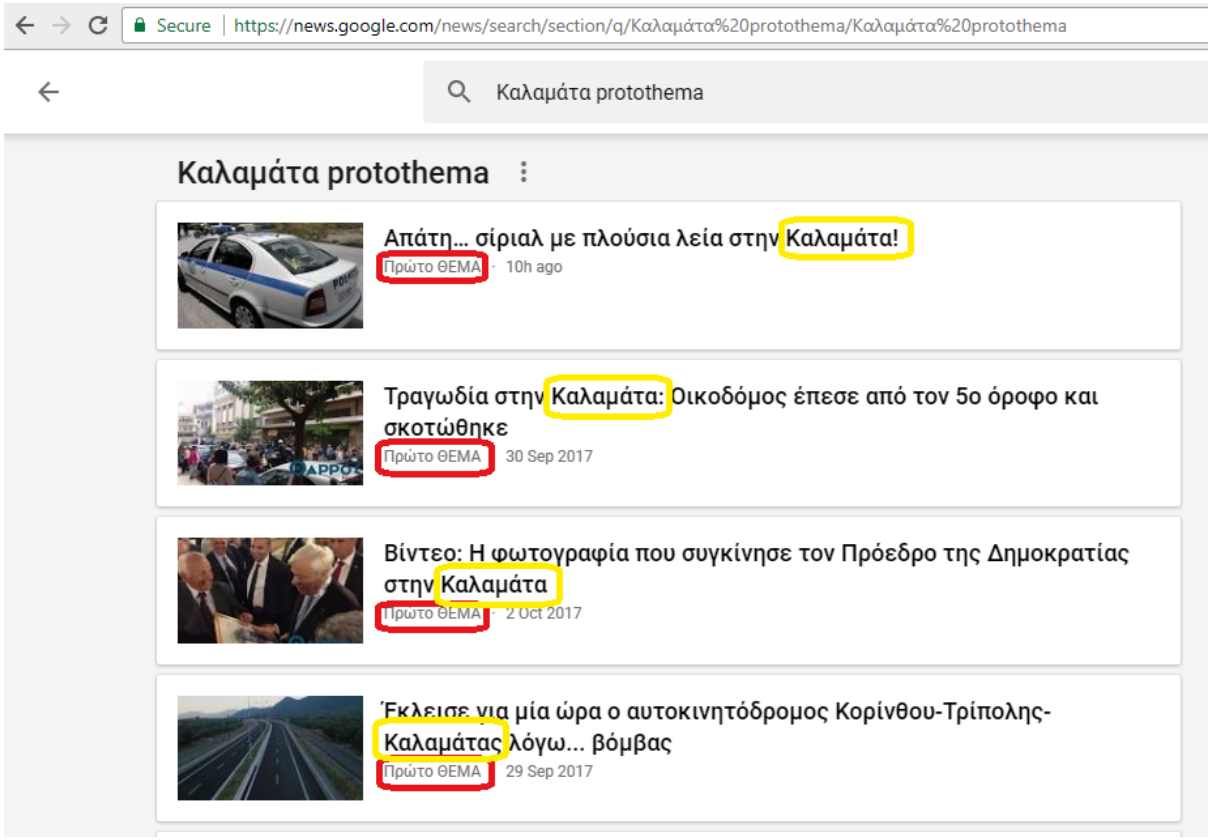


Figure 14.Articles that were retrieved based on a given location and a website domain name

In order to take the title, the text and the published date of these articles we can use the url of the **RSS** (**Rich Site Summary**) which is a type of web feed which allows users to access online content in a standardized, computer-readable format. In this url, we pass the location name and the website domain name in order to access a XML file from which we take the links of the articles. Then, we use the crawler we have created for this website to download test data for this location. This procedure is implemented for all the websites we have created crawlers.

Figure 15. Url for accessing RSS feed XML file that contains links to prototheme news articles

5.3 Date Extraction from Text

Most of the articles, that describe events, usually contain the dates and the locations where the events will occur either at the title or at the first sentence of the text. Of course texts are unstructured data so we cannot be sure for their content, but we can rely on the mentioned hypothesis which has been observed many times in the texts that describe or mention events.

To begin with, we use dateparser[5] library which provides modules to easily parse localized dates in almost any string formats commonly found on web pages. It also recognizes names of week days and time adverbs such as today, yesterday, 2 weeks ago , 1 week and 1 day ago, in 2 days etc. As dateparser does not recognize Greek text, we also use yandex[6] library in order to translate greek text in English.

Because dateparser recognizes only particular words, phrases and dates, we split the text into sentences and we pass into dateparser the trigrams of the first sentence. If dateparser does not find a date, then for the same sentence we take the bigrams and if again it does not return anything then we take the unigrams of this sentence. This procedure is repeated for the next sentences until dateparser returns a date. If dateparser does not return anything, then we take the published date of the article as the possible date of the event. Of course, we cannot be sure that dateparser returns the real date of the event but it seems that works well in many cases.

We have passed the dictionary {'PREFER_DATES_FROM': 'future'} to the parameter *settings* of dateparser.parse() function, because we are especially interested in detecting future events. So if the text refers to a past event and contains only the name of the day the event happened, dateparser will not find the real date of the event. But if it contains the whole date the event happened, then dateparser will find the correct date. Moreover, we ignore dates that are older compared to published date of the article.

Below, we present some examples of how the function in which we call dateparser.parse() works. The parameters of this function are two, a sentence and a date which in our case represents the published date of the article the sentence belongs to.

```
>> extract_date('Carla Bruni will visit Athens on Monday.','2017-09-27')

Output: ['2017-10-02']

>> extract_date('Tomorrow, George will have an interview.','2017-10-07')

Output: ['2017-10-08']
```

[5] https://github.com/scrapinghub/dateparser

[6] https://github.com/dveselov/python-yandex-translate

```
>> extract_date('On 20 & 21 October 2017, EU leaders meet in Brussels to
discuss climate & energy problems.','2017-10-07')

Out: ['2017-10-20', '2017-10-21']

>> extract_date('The festival will start on 5/10','2017-10-05')

Out: ['2017-10-05']

>> extract_date('The High-Level Conference "European Universities in the
Energy Transition: Towards a Clean Energy Future" will take place from 23-
24 October at the Flemish Parliament in Brussels, Belgium','2017-10-17')

Out: ['2017-10-23']
```

5.4 Application of the Web Service and Results

Each time we crawl websites in order to extract test data, we use all the texts we have stored (some of them are used as training data) and the new texts that will be used as test data in order to obtain new word embeddings.

Now we represent the results that the web service returned after 2 client requests. In the first example, 5 Greek News websites crawlers were used in order to obtain results very quickly and in the second example 20 Greek News website crawlers were used. The computational time depends on the client input as the number of texts that are going to be retrieved for the location name and the time interval we give as input, is probably different each time we send same or different requests. Moreover, we need to mention that Google News only returns content from the past 30 days and Google News archive which offers a wealth of digitized historic newspapers online is not available in Greek.

In the first example, the client gives as input the location from where he/she wants to find events and the range of the published dates of the articles the events will be extracted.

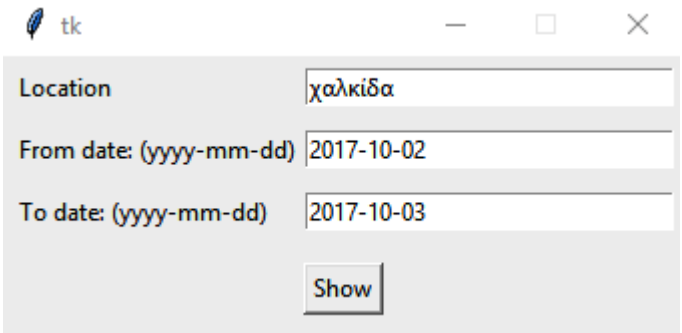


Figure 16. Client request (example 1)

The server finds two possible events for each of which it returns the text of the article, the published date, the topic which is the label and the possible date the event will occur.

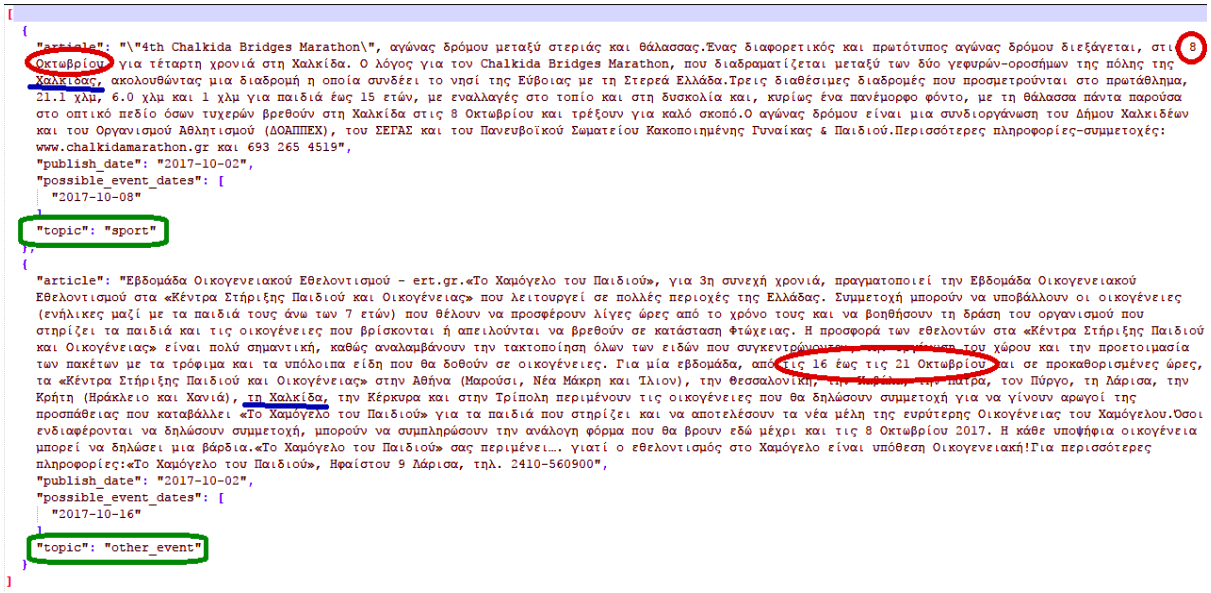


Figure 17.Results of the Web Service after Client request (example 1)

The first article refers to a road race that took place on 8 October , in Chalkida . This was a sport event. Our system detected the kind of the event and the date it took place.

The second article describes a “weaker” event that took place from 16 to 21 October in Chalkida. Our system, found the first day of the event and the kind of this event.

In the next example, we can see the inputs client gives to the web service:

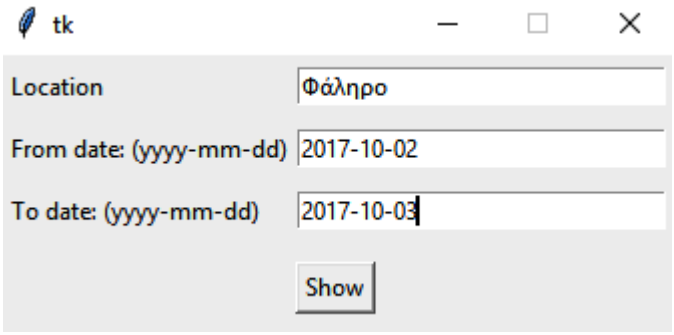


Figure 18.Client request (example 2)

Now, five articles are returned:

Figure 19.Results of the Web Service after Client request (example 2)

The first article refers to a football match which took place on Sunday 1 October. As we can see, our system detects the kind of the event but it does not detect the correct date it took place because this article refers to an event that happened in the past and our system detects events that happen in the same date with the publish date of the article(for example if the article contains the word today, tonight etc.) or in a date that is mentioned inside the text and does not precede the today's date or in a future date. The date that was extracted is 8/10/2017 which is the next Sunday after the published date of the article.

The second article refers to a football match that happened in Faliro and probably to other sport events that will happen. No dates are mentioned in the text and as a result the system gave the published date of the article as the possible date of the event.

The third event is a concert that happened on 11 October, in Faliro. Our system detects the kind of the event and the correct date it took place.

The fourth article refers to a sport event. It is not mentioned any date in the article, so the system gave the published date of the article as the possible date of the event.

Finally, the last article refers to a cultural event. It may be a 'weak' event. Our system does not detect the correct date.

Chapter 6

Conclusions and Future Work

During this master thesis, a simple web service was created which takes as input the name of a location and a time interval and gives as output some articles of Greek News websites that describe events, the kind (cultural, sport etc.) and the possible date of each event. So, first we created 48 Greek News website crawlers in order to extract articles from the websites. Then, we annotated each text with a label that described the kind of the event. For the texts that did not describe events, we used the no_event class. We also used all the texts in order to take word embeddings that were used at four out of five text representation models. Particularly, we created five text representation models and for each of them we trained eight classifiers i.e. 40 models. The metric that we used in order to evaluate the models was the macro recall score because the goal of our classification task was not to lose the real events even if many instances were wrongly classified as events. SVM had the best macro recall scores for all the text representation models apart from one (i.e. doc2vec).However, the differences between the macro recall scores were negligible between most of the text representation models especially when the classifier we used was the SVM. Maybe, if we had more data these differences would have been more obvious and we would have been sure for the best model for this task. The final model we chose was the one with the maximum macro recall score which was equal to 86.12% and was given by the **SVM** classifier when the text representation model was the **weighted-Idf centroid** (we call it Idf_centroid). Finally, we use this model in order to classify the new articles that are downloaded each time we run the web service.

Landauer et al. [24] estimates that 20% of the meaning of a text comes from the word order. According to this, our first four text representation models (sum-of-the-vector, centroid, idf-weighted centroid, and multivariate Gaussian distribution for text representation) are oversimplified because of the loss of order information. On the other hand, doc2vec which is used for summarizing bodies of text of varying length do not seem to have good performance in our task, especially in macro recall score which is our main evaluation metric. So, it would be extremely interesting to see how the neural networks which were designed to handle sequence dependence will perform in this task. So, a future work will be to collect more data, annotate them with labels and develop Long Short-Term Memory (LSTM) and Bidirectional Long Short-Term Memory networks which are types of Recurrent Neural Network (RNN) and Bidirectional Recurrent Neural Network (Bi-RNN) respectively.

References

[1]. Hotelling H.(1933).Analysis of a complex of statistical variables into principal components. Journal of Educational Psychology, 24, 417-441

[2]. Torgerson, W. S. (1952). Multidimensional Scaling: I. Theory and Method. Psychometrika, 17, pp. 401-419.

[3].Laurens van der Maaten and Geoffrey Hinton.Visualizing Data using t-SNE. Journal of Machine Learning Research 9 (2008) 2579-2605

[4]. A. Kosmopoulos, I. Androutsopoulos and G. Paliouras, "Biomedical Semantic Indexing using Dense Word Vectors in BioASQ". *Journal of Biomedical Semantics*, supplement on Semantics-Enabled Biomedical Information Retrieval, 2016

[5]. G. Brokos, P. Malakasiotis and I. Androutsopoulos. "Using Centroids of Word Embeddings and Word Mover's Distance for Biomedical Document Retrieval in Question Answering". Proceedings of the 15th Workshop on Biomedical Natural Language Processing (BioNLP 2016), at the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016), Berlin, Germany, 2016.

[6]. Silla Jr, C.N., Freitas, A.A. A survey of hierarchical classification across different application domains. Data Mining and Knowledge Discovery 22(1-2), 31–72 (2011)

[7]. Yuan, G.-X., Ho, C.-H., Lin, C.-J. Recent advances of large-scale linear classification. Proceedings of the IEEE 100(9), 2584–2603 (2012)

[8]. Quanzhi Li, Sameena Shah, Xiaomo Liu, Armineh Nourbakhsh. Data Sets: Word Embeddings Learned from Tweets and General Data. Proceedings of the Eleventh International AAAI Conference on Web and Social Media (ICWSM 2017)

[9]. Bengio, Y., Ducharme, R., Vincent, P., Janvin, C. A neural probabilistic language model. The Journal of Machine Learning Research 3, 1137–1155 (2003)

[10]. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J. Distributed representations of words and phrases and their compositionality. In: Proceedings of 27th Conference on Neural Information Processing Systems, Harrah's LAke Tahoe, USA, pp. 3111–3119 (2013)

[11]. Pennington, J., Socher, R., Manning, C. GloVe global vectors for word representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, Doha, Qatar, pp. 1532–1543 (2014)

[12]. Levy, O., Goldberg, Y. Linguistic regularities in sparse and explicit word representations. In: Proceedings of the 18th Conference on Computational Natural Language Learning, Ann Arbor, Michigan, pp. 171–180 (2014)

[13]. Levy, O., Goldberg, Y. Neural word embedding as implicit matrix factorization. In: Advances in Neural Information Processing Systems, Montreal, Canada, pp. 2177–2185 (2014)

[14]. Levy, O., Goldberg, Y., Dagan, I. Improving distributional similarity with lessons learned from word embeddings. Transactions of the Association for Computational Linguistics 3, 211–225 (2015)

[15]. Mikolov, T., Yih, W.-t., Zweig, G. Linguistic regularities in continuous space word representations. In: Proceedings of 13th Conference of the North American Chapter of Association for Computational Linguistics, Atlanta, US, pp. 746–751 (2013)

[16]. Mikolov T., Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. In Proceedings of Workshop at ICLR

[17]. Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural Language Processing (Almost) from Scratch. volume 12, pages 2493–2537. JMLR.org.

[18]. Lebret R. and Collobert. R. 2013. Word emdeddings through hellinger pca. arXiv preprint arXiv:1312.5542.

[19]. Levy O. and Goldberg Y. 2014. Dependency based word embeddings. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, volume 2, pages 302–308.

[20]. Word2vec, Tool for Computing Continuous Distributed Representations of Words. https://code.google.com/p/word2vec/

[21]. Sahar Ghannay1 , Benoit Favre2 , Yannick Esteve 1 and Nathalie Camelin1. Word Embeddings Evaluation and Combination.

[22]. Lebret R. and Collobert R. 2015. “The Sum ´ of Its Parts”: Joint Learning of Word and Phrase Representations with Autoencoders. arXiv preprint arXiv:1506.05703

[23]. Nikolentzos G.,Meladianos P., Rousseau F. ,Vazirgiannis M. and Stavrakas G. Multivariate Gaussian Document Representation from Word Embeddings for Text Categorization.

[24]. T. K. Landauer. On the computational basis of learning and cognition: Arguments from lsa. Psychology of learning and motivation, 41:43–84, 2002.F

[25]. Le, Q.V., Mikolov, T.: Distributed representations of sentences and documents. In: Proceedings of the 31th International Conference on Machine Learning, Beijing, China, pp. 1188–1196 (2014)

[26]. N. V. Chawla, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Oversampling TEchnique. Journal of Artificial Intelligence Research, 16:321357, 2002.

[27]. M. Kubat and S. Matwin. Addressing the curse of imbalanced training sets: One sided selection. In Proceedings of the Fourteenth International Conference on Machine Learning, pages 179-186, Nashville, Tennesse, 1997. Morgan Kaufmann.