Mixing unstructured and structured analysis to predict similarity

Nicolas Turpault

August 25, 2017

Contents

Abstract

This document presents the second part of my work during my apprenticeship at ATOS. I compared a deep learning approach with a traditional approach using Ngrams and term frequency inverse document frequency for a concrete natural language processing problem. The deep learning approach using word embeddings as input to (bidirectional) long-short term memory networks succeed better on this task than the traditional approach.

Introduction and Context

The French family allowances organism (*Caisse nationale d'allocations familliales* – CNAF) uses an online portal whose maintenance relies on a ticketing system. A ticket is written by a user (employee or providers) who uses an internal application of the French organism and report a bug or ask for an evolution. A team is assigned to the ticket, treat it if possible or forward it to a more relevant team otherwise. Since 2012, 350 000 tickets indicating bugs or asking for evolutions have been received on their online portal. This is generating a massive amount of data traffic within the helpdesk system and a significant amount of work for the helpdesk teams.

CNAF was looking for a way to optimize their helpdesk ticketing system. However, neither did they know which parts could be optimized thanks to machine learning nor did they identify clearly the potential problems. They provided me with their data and asked me to help them reducing the global cost of their solution. The dataset contains tickets that are composed of a text written by a user and describing his problem, some categorical columns (to describe the user, the priority of the problem,...) and utilities data (dates, indexes).

The first step of this work was to find the optimization problem that I could try to deal with. It was an important part of my work, because it requires to have domain specific skills to understand the data, ask the right questions and the problem identified need to be solvable. After some visualisations and meetings with employees, I identified two problems that I could deal with using machine learning.

- **For a single problem, there is usually several tickets.** It is important to identify the tickets that refer to the same problem. Indeed every similar ticket that is not identified is generating additional workload for the team allocated to the ticket.
- **Each ticket is received by the support team and should be dispatched to the appropriate specialised team.** Delivering the ticket to the right team (the one able to solve the problem) can be quite tricky. It is common to see a ticket going from team to team before reaching the team will be able to solve the problem. With this current ticket allocation system specialised teams get more tickets than they should and need to redirect them. The operational mode is time consuming and the number of unsolved tickets is important.

To solve these problems, I proposed to develop a tool which could help the support team to take a decision concerning the similarity of the tickets and the team able to solve the problem. For

both problems two approaches are proposed: one that uses only text and another that uses both text and categorical data.

Text analysis is an important topic for CNAF because they have multiple problems that could be solved thanks to this approach (i.e. customer satisfaction, help with post letters, ...). That is why, my work focuses in priority on the team prediction using only the text of a ticket. The approache to predict similarity and the approaches using categorical data to improve the predictions will be described as well. However, due to the lack of time, these descriptions will not be followed by the presentation of experimental results.

CNAF and ATOS are new to deep learning approaches. Therefore, an important part of my work during this apprenticeship was to explain deep learning bases to them and compare my approaches with the traditional approaches in machine learning they know.

This document will first present the state of the art techniques for text classification. Then the team prediction will be developed using different approaches using only texts or texts and categorical variables. Similarity prediction will also be discussed in details following the same approach before presenting the results and conclude.

State of the art

Text classification is a classical topic of natural language processing (NLP). While traditional approaches which use lexical features such as Ngrams and term frequency inverse document frequency (TFIDF), and then apply probabilistic, linear or kernel based classifiers are mostly used in French companies, new techniques using deep learning begin to be developed by some specific companies but not well known in general.

While deep learning outperformed the other techniques by a large margin in computer vision and speech recognition, it is not necessarily the case for NLP tasks where deep learning has good results for many tasks but not always with a large margin. Word embedding [1] which represent words in a continuous manner is one of the task which is done efficiently by neural networks. Multiple ways to achieve it appeared like word2vec [2] or GloVe [3] to improve this representation. While word representation using deep learning is the state of the art method, it is not so clear how we need to represent a sentence or a document to classify them. Traditional approaches usually use Ngrams to have words with context and TFIDF to represent them. Deep learning based approaches trying to outperform it are using convolutional neural networks [4] [5], advanced versions of it [6] and recurrent neural networks (RNN) based for example on long short term memory (LSTM) [7] have shown some good results.

LSTM has improved in time [8] and is recognized to be able to store and access information over a long period of time. In NLP tasks, some approaches use other recurrent neural network (RNN) like gated recurrent units (GRU) [9] in complex configurations [10]. GRU and LSTM are really close, because each of them are using gates and recurrent architecture, however, GRU does not have a memory cell which reduce calculation complexity and still show good or better results than LSTM for certain tasks.

Similarity is also a tremendous NLP task. To achieve similarity classification on words, embedding techniques allows us to choose a distance that suits us to compare words (usually cosine similarity). Deep learning and traditional approaches focus alternatively in characters and words. Character approaches allows to describe specificity of some languages and complex relations. Words are better to capture the semantic relations. Specific variation of these algorithms have been introduced to deal with sequence of words, relying on both traditional NLP methods [11] and deep learning [12, 13].

Team prediction

Baseline NLP approach on text

This approach relies Ngrams and TFIDF. Ngrams determines the features used in the data, TFIDF gives a score for each feature, then we apply a classification algorithm using these scores.

Ngram

Ngrams take in account series of multiple items. In this case it is words, but sometimes it refers to characters. Here is an example of taking unigrams (one word), bigrams and trigrams of a sentence: "A cat is in the tree"

- Unigrams : {a}, {cat}, {is}, {in}, {the}, {tree}
- Bigrams : {a, cat}, {cat, is}, {is, in}, {in, the}, {the, tree}
- Trigrams : {a, cat, is}, {cat, is, in}, {is, in, the}, {in, the, tree}

In this document, when I am using Ngrams, it is every Ngrams from size one to six.

TFIDF

TFIDF is a classical method used in a lot of NLP tasks [14], and have multiple variants [15]. In this document we will use the traditional method. Considering a specific term t in document d, the main idea is to find how important is the term in this document (term frequency) and then multiply it with a measure of how rare is this term in other documents.

Term frequency ($tf_{t,d}$) is a count of the occurences of the term t in the document d. In this case, it is divided by the number of words in the document to adjust with the length of each document, but multiple variants exist.

$$tf_{t,d} = \frac{\text{frequency of term } t \text{ in document } d}{\text{number of words in } d}$$

Inverse document frequency ($idf_{t,d}$) is high when the term t is specific to the document d and is low if the term t is present in every document. This equation allows to penalize a frequent term which is in every document and improve a rare term specific to a document. It can be seen as a measure of capacity to predict the document d from the term t.

$$idf_{t,d} = \frac{\text{number of documents}}{1 + \text{number of documents where } t \text{ is present}}$$

The additional one in the denominator of $idf_{t,d}$ is a regularisation term to prevent the division by 0. The result TFIDF of a term in a document is given by

$$tf_{t,d} * idf_{t,D}$$

.

Deep learning approach using text only

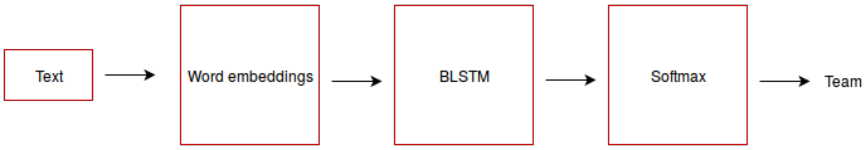

Figure 1: Deep learning model using text only (the BLSTM layer can be replaced by a LSTM layer)

Identifying a customer's problem from the text of a ticket is sometimes tricky for human, so is it for a machine learning approach. The first solution I proposed is presented in Figure 1. A similar approach has already been used for different problems like classify semantic relations [16] or sentiment analysis [17]. Some freedom has been taken from these papers because applications are not similar but the main idea remains the same. The proposed approach relies on word embeddings in a first stage. Then, to classify documents or retrieve semantic relations, we need to get information gain during sentences, that is why it uses a recurrent neural network (in this case LSTM or bidirectional LSTM (BLSTM)).

Word embeddings

Word embeddings is a common practice in NLP and has proven its relevance for common problems. Multiple ways to do it exist like word2vec [2], GloVe [3]. Even ways to keep sentiment representation in embeddings has been proposed [18]. Word2vec and GloVe share some similarities but their loss function are different [19, 20]. Results can be better from one to the other depending on the task. GloVe can also be really similar to word2vec when the parameters are set in a particular way. In this experiment, I use word2vec in Skip Gram (SG) Negative Sampling (NS) configuration.

Word2Vec In word2vec a neural network is trained to predict a word from its context (the surrounding words). At inference time, the final vector representing the word in the new space is obtain from the last hidden layer of this network.

Word2vec is generally associated with two alternative methods: SG and Continuous Bag-of-Words (CBOW). The difference between each of them is the performance and the speed to compute. The model which seems more adapted to my problem is SG, this is why I decided to focus on this approach. In SG with one word as context, the input is a pair $(w_I, w_{\text{context}})$ (representing (input, label)). The input word and the context word are represented as one-hot vectors of the size of the vocabulary (0 or 1, 0 everywhere and 1 to the index corresponding to the word in the vocabulary).

Let's take an example of an architecture from the literature with 10 000 words vocabulary and a 300 neurons in the hidden layer (see also Figure 2)[1]. This means a representative vector of size 300 at the end. The output layer of this network is calculated thanks to:

$$y_{\text{context},j} = P(w_{\text{context},j} = |w_I) = \text{softmax}(\text{score}(w_{\text{context}}, w_I)$$

$$P(w_{\text{context}}|w_k) = \frac{\exp(\text{score}(w_{\text{context}}, w_I))}{\sum_{\text{Word w' in Vocab}} \exp(\text{score}(w_{\text{context}}, w'))}$$

where $\text{score}(w_{\text{context}}, w_I)$ is calculated by multiplying the input vector w_I with the hidden layer weights, then the output weights and finally with the target word w_{context}. But this is when we want to calculate only one word in the context, when multiple words are predicted, it is not so trivial.

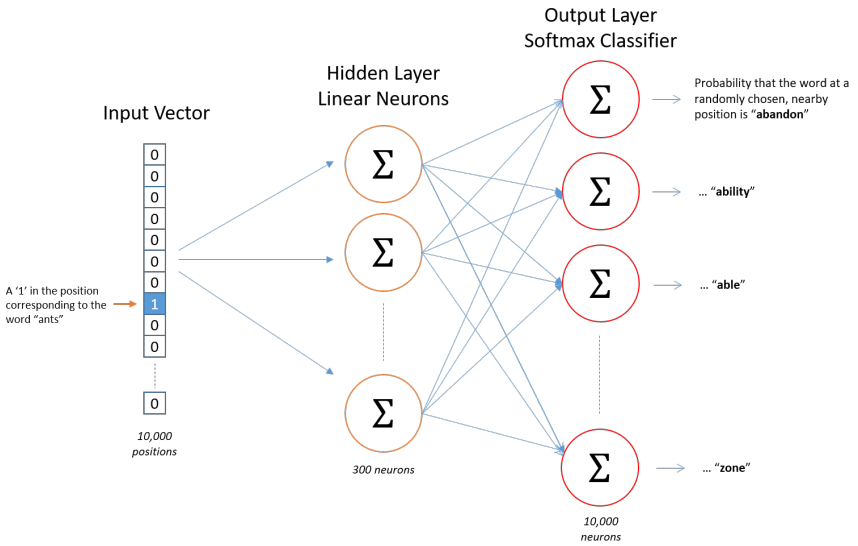

Figure 2: Example of a skip-gram model on 10 000 words vocabulary and 300 representative vector (hidden layer)

Word2vec also uses negative sampling [21] which is a computational method to avoid updating all the weights for every words. We only update the weights of the current word and multiple

[1]Figure 2 and example from mccormickml site

"negative" words, which means, words for which we want a 0 output with the current word (words not in the context of the current word).

The loss function of this algorithm is:

$$E = -\log p(w_{context,1}, w_{context,2}, ..., w_{context,S}|w_I)$$

where S is the window size, in the case that we want only one word, we only have $E = -\log p(w_{context}|w_I)$. The networks is updated using stochastic gradient descent [22] to minimize the error between y_{context} and w_{context}.

Thanks to this method, we can represent our vocabulary in a space where similar words are near to each other and the vector associated with a word can have a reasonable size. We use this technique in this work because it works pretty well in practice and save some time as it is already implemented.

An interesting work could be to compare the different solutions for our problem: different configurations of word2vec (SG vs CBOW, hierarchical softmax vs NS), GloVe and a solution with a LSTM layer from one hot encoder.

Long Short Term Memory

A LSTM is an algorithm which allows to learn a representation of the data and which is adapted for sequence data (i.e. text, sound, ...) and particularly to learn long dependency in comparison of other RNNs [23]. You can see on Figure 4, the principle of RNN cells. It is to keep in memory the state of the neuron and reuse it for next inputs in comparison of a feed-forward (classical) neuron described on Figure 3.

Figure 5 presents the representation of a LSTM cell. You can see that it is a more complex neuron where you can retrieve multiple elements that are computed:

- **input gate** decides which information from this input is used

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

- **forget gate** decides which information to keep from previous state

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

- **cell state** represents the neuron, is is updated using input gate, and forget gate $\tilde{C}_t$ is the candidate value for the states of the memory cell at time t.

$$\tilde{C}_t = tanh(W_c x_t + U_c h_{t-1} + b_f)$$

$$C_t = i_t * \tilde{C}_t + f_t * C_{t-1}$$

- **output gate** can be considered as a "filtered" version of the cell state, and represents the information we want to output.

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + V_o C_t + b_o)$$

- **hidden state** is the representation of the neuron, the state shared through time in the network. It uses output gate and cell state to be computed.

$$h_t = o_t * tanh(C_t)$$

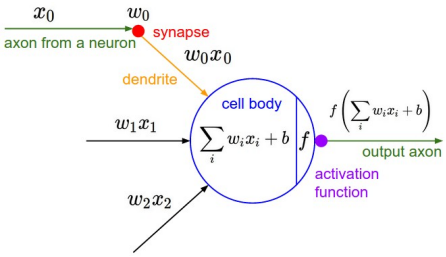

Figure 3: A classical computing neuron

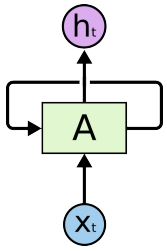

Figure 4: Principle of every RNN cell (source colah's blog

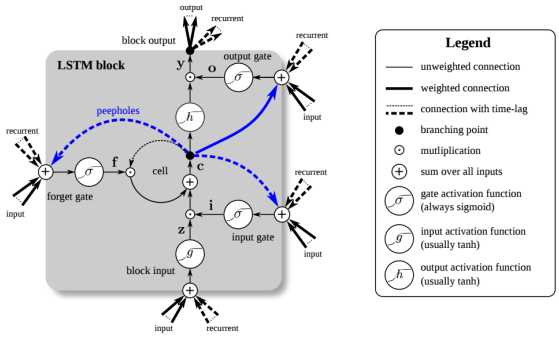

Figure 5: The LSTM cell (source deeplearning4j)

This network is famous to be a RNN which avoids the vanishing gradient problem [24, 25] that exists with some other RNN. This algorithm is well described in literature [26] and some variant and improvement have been proposed from its apparition [27]

Bidirectional LSTMs Bidirectional RNN (BRNN) [28] have been introduced in 1997 and have outperformed many unidirectional RNN. In particular, BLSTM [29] outperformed LSTM on many tasks where they were the state-of-the-art. The general idea of BRNN is to compute in parallel a forward and a backward recurrent layer (Figure 6). This implies the network get the sequence from both side and average, sum, multiply or concatenate the results. In this work, the concatenation function is used, it is really common to use this one.

Approach using text and categorical variables

Categorical variables present in the data (such as type of demand, or service asking) can be used directly (or using one hot encoding depending of the algorithms) as input feature to a decision algorithm. Therefore, the solution I propose here combines the output of the text

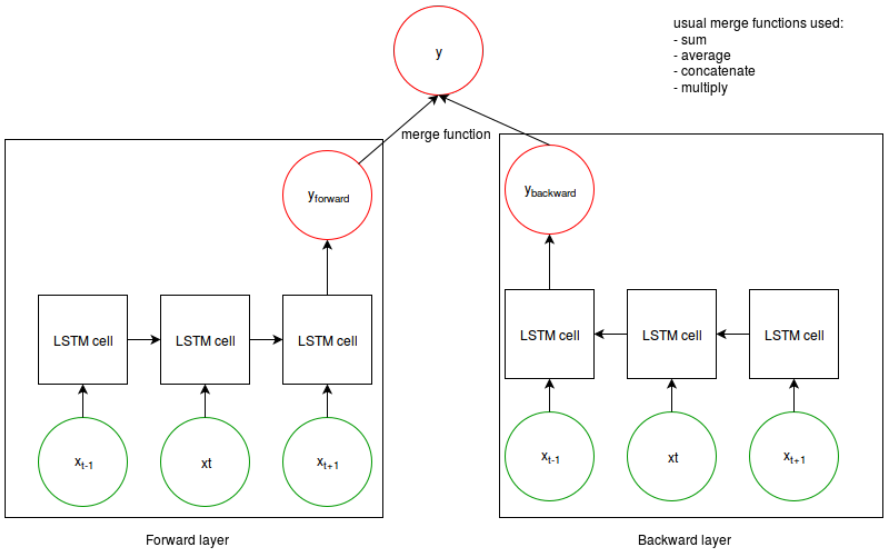

Figure 6: Bidirectional LSTM explanation. (example with 3 timestamps)

analysis system described above with categorical data. The main idea is to concatenate a vector representing text with a vector representing categorical variables, then apply a classification algorithm. In this case, I proposed to use support vector machines (SVM) and random forest (RF) because they are common classification algorithms I know well and show good results in practice without requiring a lot of computing power. Indeed, eventhough the textual part to predict a word vector from a text will be trained only occasionally (once per month or every two months), the other part that predict the team or the similarity thanks to SVM or RF could be retrained daily or weekly to improve its accuracy and to adapt to new incoming teams for example. This explains the importance to have an algorithm without an important computing overhead.

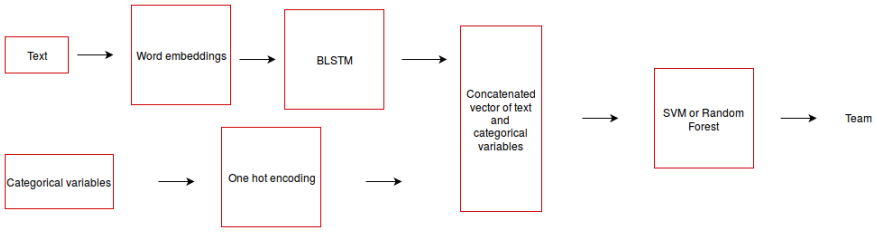

Figure 7: Model using text and categorical variables to predict team

In the figure 7, during the one hot encoding part, we concatenate all the one hot vector produced by categorical variables. Then we concatenate this vector with the output of the BLSTM trained in the previous approach that used only text.

One hot encoding

One hot encoding is a simple method which represents a variable by multiple binary variables. An example: A variable *animal* which can take values: {cat, bird, dog} can be represented by a vector where:

(1, 0, 0) is cat
(0, 1, 0) is bird
(0, 0, 1) is dog

Random forest

I first decided to use a decision tree and then a random forest on my categorical data. A decision tree uses each variable on the table to discriminate as much as possible the data. All the data are used in certain order given by entropy value:

$$H(X) = \sum_{i=1}^{n} P(x_i) log_2 P(x_i)$$

where x_i is the different values taken by a variable. Random forest is a technique proposed by Breiman in 2001 [30]. It helps to reduce some weaknesses of the decision tree, i.e. the best tree is not always a tree that maximise the entropy at each iteration or the generalization problem. The random forest uses a number of decision trees built in a randomly selected subspace of the data set and average their decisions. I use random forest for similarity and team prediction.

Support Vector Machine

I used SVM only for the similarity problem that needed to predict two classes and not multi-classes. It is possible to predict multi classes using SVM but it induces multiple SVM to be computed with one-vs-rest or one-vs-one method. Another method is to use multi classes SVM [31] which takes less time than the other SVM methods. However, for my problem, even multi classes SVM takes too much time because of the important number of teams, that is why I use RF for team prediction. The classification with SVM is a technique which uses another space of representation of the data to find a linear separation (hyperplane) between two categories. This transformation of space is done using a kernel which represents the dot product in the space where we find the linear separation.

Basic principles of Support Vector Machine: On Figures 8 to 10[1], we can see some basic principles of SVM. Figure 8 represents some examples of linear separations between categories, infinity of them exists and some algorithms just predict one between them without knowing if it is the best (for example neural network). SVM is an algorithm which take the line which optimize the margin between categories, it is represented in the figure 9.

[1]Figures 8 to 10 from A Gentle Introduction To Support Vector Machines In Biomedicine book [32]

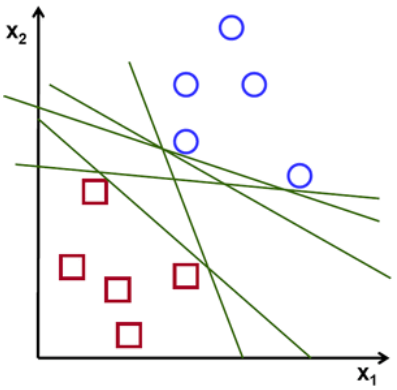

Figure 8: Which linear separation between the data do we use?

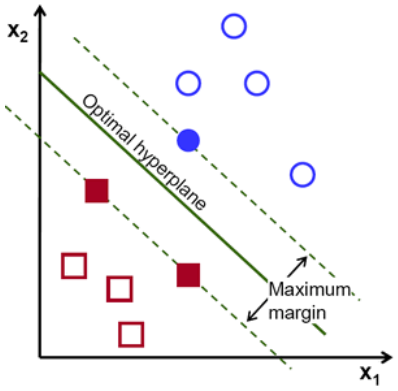

Figure 9: SVM principle is to find the maximum margin and optimal hyperplane

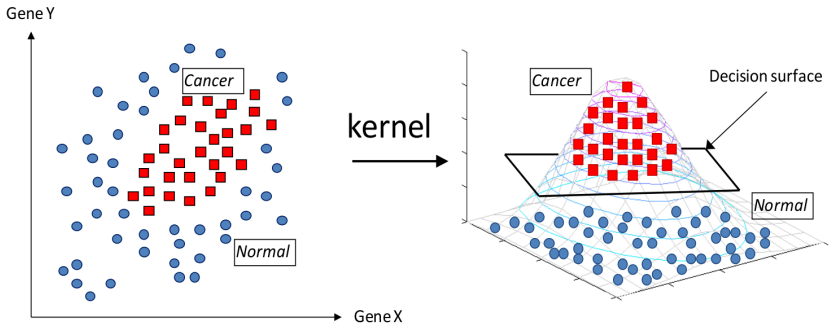

Figure 10: The kernel trick

To perform the classification task, we search the hyperplane based on "support vectors" (dots on the boundaries of each classes).

Given a training dataset composed of N data points $\vec{x_1}, \vec{x_2}, ..., \vec{x_N} \in R^n$ and the corresponding labels $\vec{y_1}, \vec{y_2}, ..., \vec{y_N} \in \{-1, +1\}$. The equation of the optimal hyperplane is:

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

The optimisation problem associated to maximize the margin is:

$$\frac{1}{2} \sum_{i=1}^{N} w_i^2 \text{ subject to constraints: } y_i(\vec{w} \cdot \vec{x_i} + b) - 1 \leq 0$$

Which can in turn be reformulated as the dual problem:

$$\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \boxed{\vec{x_i} \cdot \vec{x_j}} \text{ subject to constraints: } \alpha_i \geq 0 \text{ and } \sum_{i=1}^{N} \alpha_i y_i = 0$$

with $\vec{w} = \sum_{i=1}^{N} \alpha_i y_i \vec{x_i}$ and the solution thanks to Karush-Khun-Tucker complementary conditions:

$$f(\vec{x}) = sign(\sum_{i=1}^{\#\text{support vectors}} \alpha_i y_i \vec{x_i} \cdot \vec{x} + b)$$

As you can see in the equations above, the number of free parameters is bounded by the number of support vectors and not by the number of variables. Moreover, we do not need to access the original data, but only their dot product, that is why we can use the kernel trick originally used for large margin classifiers by Boser et al [33] and described in Figure 10. In this method we define a kernel $\phi(x_i, x_j)$ which is a function that represents a dot product in a particular space. This "trick" has a huge advantage: we do not need to convert every input in this new space but only define the kernel function associated with our examples: $\phi(x_i, x_j) = \vec{x_i} \cdot \vec{x_j}$ (Gaussian and polynomial functions are kernels for example). That's why, in theory, we could even use a space with infinite dimensional vector spaces.

Similarity prediction

I also proposed two approaches for similarity prediction: a first approach using only text and then an improved solution including categorical variables.

Deep learning approach using text only

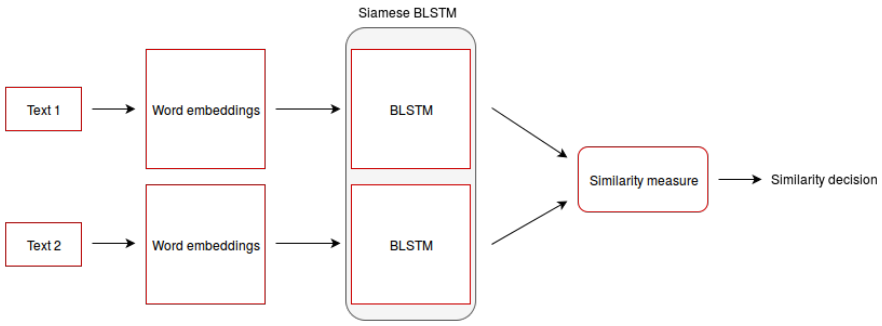

Figure 11: Similarity model using only texts

This approach uses previous work done for the team prediction. Indeed, the word embeddings layer is the same as the one used for team prediction and the LSTM or BLSTM used within the siamese architecture proposed hereare trained for the team prediction problem.

Siamese LSTM

Siamese LSTM [34] is a technique that compares two LSTMs (learned in the same way) and share a final weight (in our case, a cosine similarity) to be able to compare their representations and optimise their learning. In this work, the LSTM is pretrained for team prediction with the text analysis and then optimised for similarity prediction within the siamese framework.

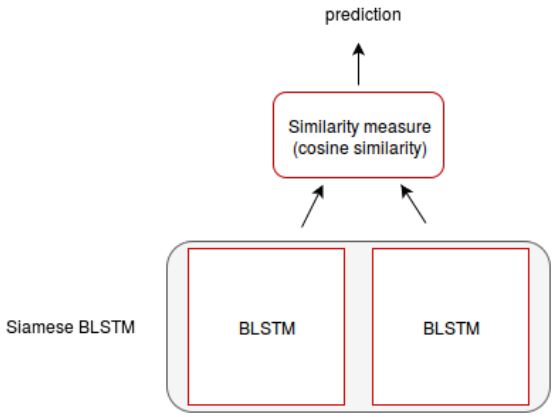

Figure 12: Siamese LSTM architecture [35]

The figure 12 describes an example of a Siamese LSTM architecture. However, for the similarity problem described in this document, we replaced the final weight by the cosine similarity function and the input are the embeddings of every words.

It is important to point out that the two LSTM or BLSTM used in the Siamese architecture are similar (i.e. they have the same weights) and even when we train, or finetune them, they share the same weights. Indeed, this make possible the similarity calculation with the function we want (cosine similarity for the problem described).

Approach using text and categorical variables

In this approach, the main idea is to calculate the similarity separately using text and categorical variables, then calculate the final similarity measure (in this case we average, but it could be a sum) to take the similarity decision.

In both text and categorical variables approaches, we use the cosine similarity function for consistency and ensure an easy calculation of the final similarity measure. I use the cosine similarity function, but it would be interesting to compare results with other similarity functions.

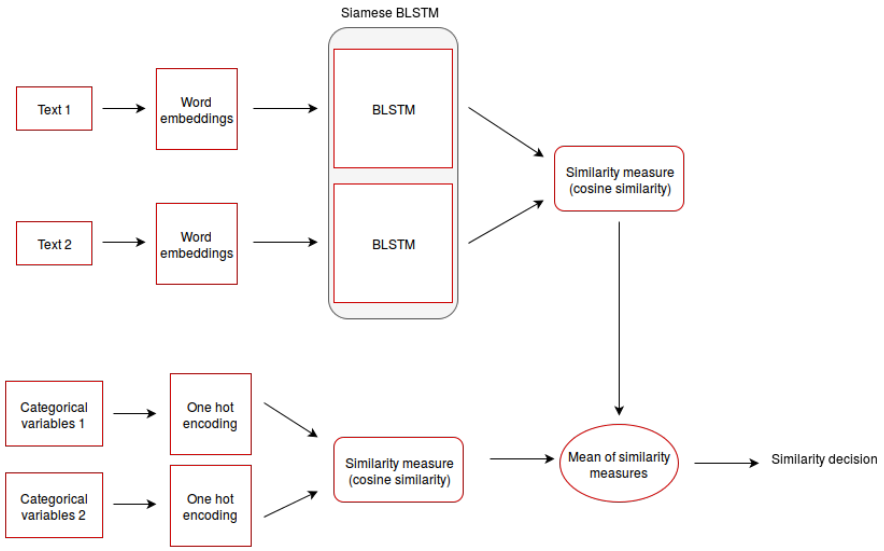

Figure 13: Similarity model using texts and categorical variables

Experiments

Data

Te dataset is composed of 350 000 tickets that represent 27 columns of data in which there is:

- 4 textual columns: Short description, Long description, technical solution, functional solution
- 3 identification columns: Identification number, identification number of the similar ticket, long identification number
- 3 date columns: creation date, solving date, solving predicted date
- 17 categorical columns (not developed because they are very specific to the problem and does not improve the understanding of the problem)

My work focus on two predictions (team and similarity) which appear at the beginning of the process. However in the data collected, some columns are modified during time or not indicated at the beginning. There are also non informative columns, that is why I finally used:

- 1 textual columns: long description
  Simple description is nearly always empty, and the 2 answers contain data that are not defined at the beginning of the process.

- 2 identification columns: Identification number, identification number of the similar ticket **One of them determine the similar ticket**, i.e. it is a label I want to predict. I keep only one of the two others because they determine the same information which is the "key" of the ticket in two different ways.
- 1 date column: creation date
  It is the only date we know at the beginning of the process where the prediction will be applied. We will use it as a delay between the two tickets we want to compare.
- 10 categorical columns:
  - qualification of the problem (bug, evolutions, ...)
  - service asking (production, validation, ...)
  - nature (functional, technical, other)
  - certification
  - site asking (region in France)
  - object (caf.fr, physical desk, human resources, ...)
  - blockage
  - severity
  - security
  - type of problem (internet, collaboration, ...)

  I didn't take the seven other columns because some of these columns change during time (state, priority, ...) and others are not informative (too much categories or is too many times null value)

There is 10% of similar data, it means that 90% of the data are isolated and can not be linked with another ticket. For the problem, I decided to take these 10% of similar data, and only 30% of the isolated ticket (non similar data) to reduce the size effect on the training. The similarity table 1 contains only boolean values because we could not define a measure of similarity (distance between each values of categorical variables) between the categorical variables we had. For example a categorical variable we have is the type of demand we have: bug, asking for improvement, technical question, ... we could not find a measure of similarity between "bug" and "asking for improvement". That is why, when we compare 2 tickets, the only information is a boolean representing if the type of demand is similar or not.

To link the previous paragraph (and Table 1) with Figure 13 representing the model, a simple explanation of the trick needs to be explained. Figure 13 shows two vectors of categorical variables represented by concatenated one hot vectors and the calculation of the cosine similarity between the two vectors. However, in the previous paragraph and in Table 1, it is explained that a single vector representing differences is produced. Indeed, the cosine similarity between the categorical variables of the two tickets can be calculated from this unique vector: The dot product in the cosine similarity is the the number of one (True in the table). The other term (the product of the norm) is the number of categorical variables (before one hot encoding) squared (because they have the same length). This trick is only an computational trick, the idea behind the algorithm is still the one presented in Figure 13.

In the list describing the data, the team we take as label for team prediction is not identified. This information comes from a link database which list all the actions done on a ticket. In this database, we identify the team that performed the action that solved the ticket and merge this information with the other informations thanks to the ticket number. The result merged table contains only 220 000 tickets. It means, a third of tickets did not have a team identified for the action leading to solve it (i.e. similar tickets are in this category), were not already solved (about 10%) or were containing data impossible to use (null value in the identification column or identifier not corresponding between tables)

Table 1: Similarity table

IDT1	IDT2	COLASimilar	COLBSimilar	COL...Similar	Similar tickets
INC0268059	INC0273574	True	False	...	True
INC0135715	INC00607982	False	False	...	False

Table 2: team prediction table

IDT1	IDT2	COLA1	COLA2	COLB1	COL...X	Team
INC0268059	INC0273574	Prod	Prod	TECH	...	011EALLOC
INC0135715	INC00607982	Prod	Rec	FONC	...	031EPILOTAGE

Text analysis and preprocessing Texts contained in tickets are noisy and specific. Many abbreviations, typing errors and specific words exists in these texts. Moreover, they are written in French and many preprocessing methods are lacking in French or give bad results (i.e. stemming, stop words).

That is why, I only used these preprocessing steps:

- convert to lower-case
- lemmatization
- remove punctuations
- remove numbers

Lemmatization has been done using TreeTagger [36] [37]. This method gives a POS-TAG [1] and a lemma [2] for each word. If the lemma is unknown, it returns <unknown>. This step replace each word by its lemma when it was known. Punctuation is also used to lemmatize. That is why, I remove punctuation after lemmatization. (In practice, the punctuation is removed at the same time words are replaced by their lemmatization in sentences).

[1] Pos-tag is the grammatical form of a word: verb, noun, pronoun, ...

[2] A lemma is the dictionary form of a word, for example every conjugated form of a verb refers to the infinitive form

Experimental setup and evaluation

This section will only focus on team prediction using the textual data.

data repartition and teams We have a test set of 15% and a training set of 85% of the data. To have a test set reproducing the reality of our problem, I needed to take the last tickets I had in my database. Indeed, the goal of this algorithm is to predict the team of a new ticket. Taking 15% of the database to test is quite large because it represents the 6 last months of my data (January 2017 - June 2017) but it is better to take this ratio to have a significant accuracy. The number of teams in the resulting training set is 746. Because we have tickets from the last 6 months, we will test our algorithm with some teams not seen in the training set (new teams often appear), this means there is a percentage of teams impossible to predict well. It represents 6,3% of the testing data in this case.

Moreover, the team distribution in the training set is not the same than in the testing set. However, it will be explained later the solution used to bypass this problem.

The experimental model In my model to predict the team using text, I first represent each text associated with a ticket by a sequence of word embedding (figure 14), and then use these sequences to predict the team (figure 15). We add zero or truncate texts (padding) to have every texts with a length of 150 words.

The model uses an embedding layer of 200 neurons. Moreover, in the corpus, I only use words represented more than 100 times in texts. Word2vec with SG and NS is trained during 15 epochs.

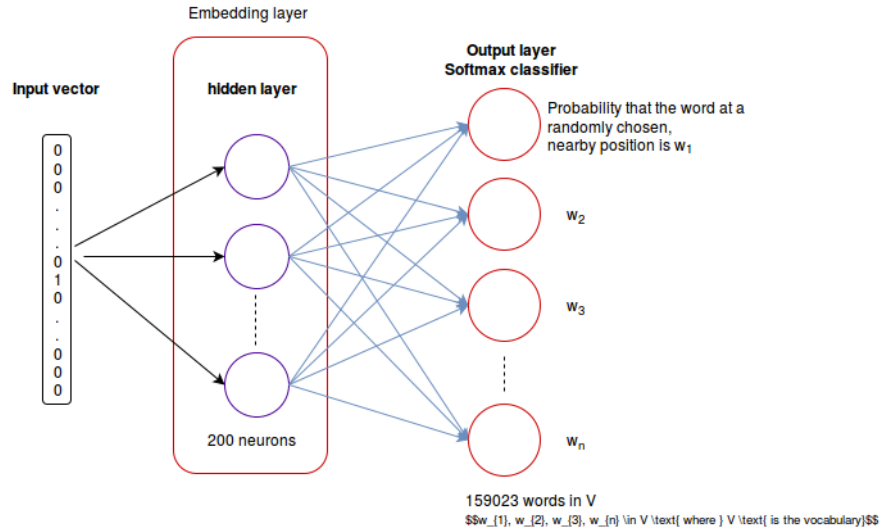

Figure 14: The skip gram model used for word embeddings

To train this model, the training set is divided into training and validation sets. The validation

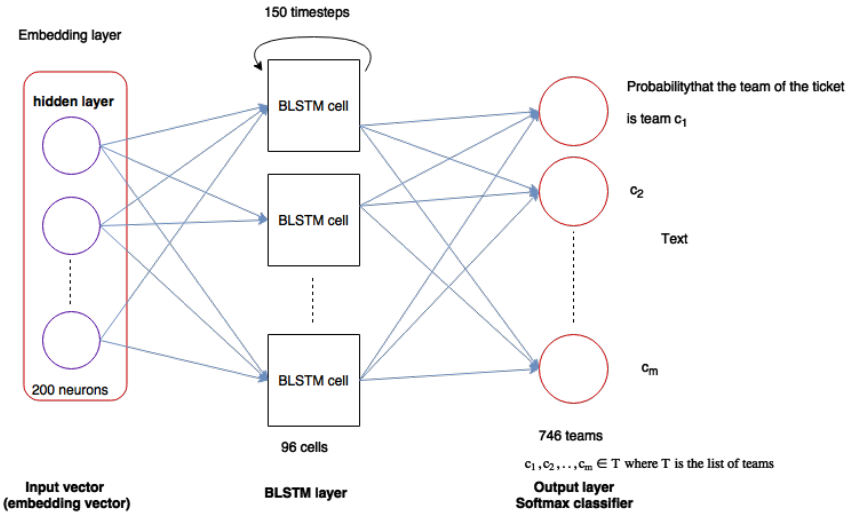

Figure 15: Model used to predict team from text. *"150 timesteps" for BLSTM cells mean we use texts of size 150 and BLSTM cell states are reset after 150 timesteps*

ratio is 20% (of the training set). This step allows to prevent overfitting and to have a score on data not seen by the model. The word embedding model (word2vec) is trained on this train/validation split and my sentences are converted in sequences of word embeddings of size 150 (padding step). The choice of the word2vec model comes from different sources [38] [39] which indicate SG is better to keep semantic information. Moreover, no hyperparameter tuning has been done for this model.

These sequences are used to train the team prediction model using BLSTM. The BLSTM cells can be replaced by LSTM cells and a comparison will be done in the result section. This step (LSTM or BLSTM) to predict team has been a bit more evaluated. Indeed, different configurations have been tested thanks to a grid search to find the best hyperparameters. Parameters that have been tuned are the following:

- number of LSTM or BLSTM cells and number of layers.
- batch size
- optimizer

I ended up with a model having only one layer with 96 cells. The batch size is 64 and the adam optimizer with a learning rate of 0.001.

finetuning I mentioned in the data repartition section that the team distribution in the training set is not the same as in the testing set. Moreover, after splitting the training set into a training and a validation set (randomly), the training and the validation set share the same distribution which is different from the testing set. This problem of distribution comes from the fact that the team distribution evolves through time. It is a slow evolution, but it is important

enough to make the training distribution really different from the testing distribution. To reduce this problem, the model is finetuned with the 20% latest data of the training set which have a distribution very similar from the testing set. This finetuning is 5 epochs of training with this last 20% of data.

Experiments To compare my result with other approaches, I decided to compare different models. Here are the models being compared with the model presented before on Figures 14 and 15:

- A classical model using Ngrams and TFIDF. This one takes Ngrams from 1 to 6 and the words appearing more than 100 times in texts and in less than 80% of the tickets. TFIDF takes as input Ngrams and prediction is done using a stochastic gradient classifier. This can be considered as a baseline system.
- The proposed model using LSTM cells instead of BLSTM ones
- Instead of using the word2vec embeddings as input, take the raw sequences of words (every word is replaced by a number) and an embedding layer with a random initialisation.
- A comparison between my models before and after finetuning.

Results will show every model described above and some of them will be combined but only relevant ones.

Software

The code is developped in python and the important libraries used are gensim for Word2vec, scikit learn for the traditional machine learning approach and treetaggerwrapper using TreeTagger for the lemmatization part. I developped all deep learning based approaches using keras with tesorflow backend. This project has been developed for the CNAF, and they wanted to reuse it. My code has been documented and I presented it to the CNAF teams because some of them want to translate it in R, and others want to be able to reuse the text analysis part for other projects. Thanks to the CNAF and ATOS, the code without data will be available on my github.

Results

To have the best results possible, I used an early stopping on validation loss with a patience of 4 epochs, it means that when the validation stops decreasing for 4 consecutive epochs, the training is stopped and the final model is the one with the minimum validation loss.

Word2vec is an unsupervised learning. Some examples of similarity scores obtained with the word2vec model presented Figure 14 are presented in Table 3. I could have put lots of others, but in this table, it represents quite well the performance of the algorithm. Words appearing a

lot in the dataset are often well correlated with their similar words, but when the word count in texts is low, it is harder to predict it. We can also see that the cosine similarity is smaller for these words, which is better for the next step of predicting the team from word embedding vectors. Indeed, it means that in the space of representations, a word with small word count is more isolated than others, that is why the prediction is not likely to be the same with this word or a similar of it which is a good behaviour knowing that its similar word is maybe wrong.

The second part of the table represents words that should not be similar, and we see that our algorithm is quite good because words have a low cosine similarity.

Table 3: Word2vec examples

Input word (word count in training texts)	most similars	cosine similarity
problème (20145)	dysfonctionnement	0.68
	pb	0.63
	problème	0.61
bonjour (2017)	bonsoir	0.73
	bjr	0.69
prix (100)	financement	0.63
	cej	0.62
Behaviour of words that should not be similar		
word	word	similarity
bonjour	problème	0.24
problème	prix	0.02

Table 4: Results of experiments

Model	Train Accuracy	Validation accuracy	test accuracy
Ngrams + Tfidf	65.61	42.87	23.26
Only majority class	7.15	7.34	4.54
random over the distribution	1.84	1.74	1.21
Word2vec + BLSTM	51.02	**45.10**	32.07
Word2vec + LSTM	50.69	43.83	31.47
Embedding layer + BLSTM	77.49	44.21	31.04
Word2vec + BLSTM after finetuning	67.24	–	**47.12**
Word2vec + LSTM after finetuning	63.60	–	46.42
Embedding layer + BLSTM after finetuning	83.70	–	43.15

In the result Table 4 we can see that Ngrams + TFIDF and models before finetuning have difficulties to predict the team using only texts. It shows the complexity of the task. Moreover, we can see that validation accuracy is far better than the test accuracy, which validate the difference between team distributions. A neural network with a softmax layer is still an optimisation problem which try to learn the probability distribution of the data. Even if it is not limited to linear operators like logistic regression, the labels (teams) will be predicted with the same distribution. This results demonstrates the importance of the finetuning part which make the predictions really better. The validation accuracy has not been used for model with finetuning for the moment but an interesting approach to validate finetuning would be to do cross validation. This process prevents against important overfitting and allows the model to train on the totality

of the training data to improve the test accuracy. Finally, we can see that results of the BLSTM and the LSTM are close, even though, the BLSTM is more accurate for this task. The last comparison we have done in this table is between word2vec and an embedding layer initialized randomly (Embedding layer + BLSTM vs Word2vec + BLSTM). We can conclude that word2vec is slightly better.

In Figure 16 we compare the validation loss through time of the deep learning models. The finetuning part is not presented because I do not have a validation method. In this figure, I decided to take the time spent instead of the number of epochs on the horizontal axis. Indeed, it was one of the problem pointed out by CNAF and Atos, the time spent during training between each solutions to find the best one (not only accuracy). Models using word2vec begin later (4min that is why it is hard to see on the graph) because the training time of word2vec is taken into account. In this figure, we can see that the embedding layer leads to overfitting really rapidly and the model did not train enough to have results as strong as word2vec. Word2vec gives the best results and the time to converge are good even if the results are still quite high due to the complexity of the task. Moreover, we can see on this graph that BLSTM performs better than LSTM for this task.

The figure 17 present the accuracy of my models. In this graph, the Ngrams and TFIDF model has been added. We can see that Ngrams and TFIDFF performs quite well in validation compared to the time it takes to train. It is an interesting approach. However, like we have seen in the table 4, they did not work well for the test set because of the different distribution.

Note: *The results are presented in a scientific and presentation goal with enough data to test on (15%). In a production context, the testing size is only of 5% which represents 11 000 tickets and the finetuning is done on the last 20% of the training data.*

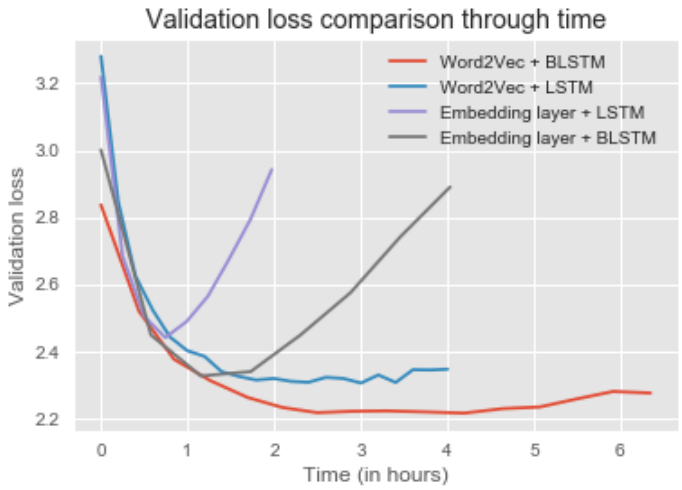

Figure 16: Validation loss through time

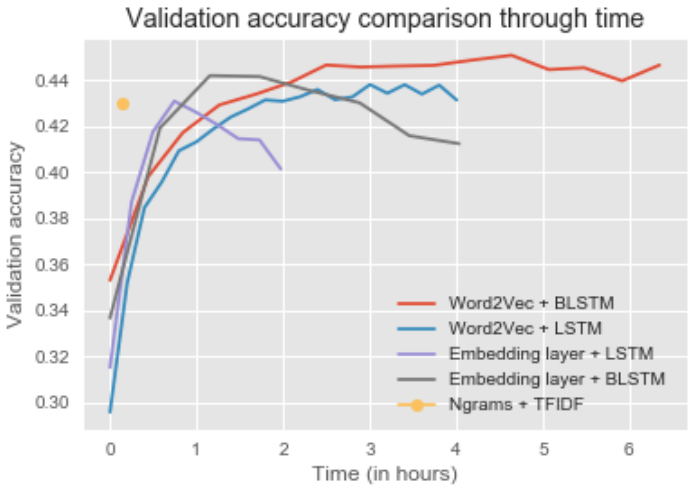

Figure 17: Validation accuracy through time

Conclusion and future work

In this document, we presented a full solution which can be implemented to solve a problem in a production context. The detailed part is the prediction of a team thanks to a text. We proposed to compare a simple approach using Ngrams and TFIDF with deep learning models. Ngrams and TFIDF is much faster than deep learning methods. However, it does not succeed to generalize well from the training and validation set, while validation results are close to deep learning models, it does not predict well on the test set. We can conclude that deep learning models performs better on this task thanks to a better generalization. In deep learning models, we have seen that BLSTM is better than LSTM on this task and using word2vec embeddings is better than an embedding layer initialized randomly which overfits too much. Finally, an interesting part on deep learning models is the importance of finetuning. It allows our model to specialize more on the last texts which increase our results on the test set. To summarize, the best approach on this task is using word2vec and a BLSTM layer finetuned with the last 20% of the training and validation data.

Predict a team over more than 700 of them using only texts is a difficult task and the noisy data is not helping. It is a really interesting task which needs more work to improve the accuracy. For example, the preprocessing steps can be compared with other approaches using stemming and stop words for example, then the word2vec model used could be pretrained with french text like the one of (Fauconnier, 2015) [40]. Find an evaluation for Word2vec could be also very interesting to see if we can improve our results thanks to a better Word2vec model. During the model part, we can compare our best model using BLSTM with a model using GRU or BGRU for example. We can also try to replace the entire model by new techniques using LSTM to learn the words and sentence representation from one hot vectors [41] or convolutional networks on text. An other important approach will be to investigate on how to reduce the impact of the distribution changing through time, finetuning was the first step to see that our model need to adapt through time to predict well in practice.

Bibliography

[1] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[2] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[3] R. JeffreyPennington and C. Manning, "Glove: Global vectors for word representation,"

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, pp. 306–351. IEEE Press, 2001.

[5] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *CoRR*, vol. abs/1404.2188, 2014.

[6] A. Conneau, H. Schwenk, L. Barrault, and Y. LeCun, "Very deep convolutional networks for natural language processing," *CoRR*, vol. abs/1606.01781, 2016.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.

[8] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[9] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014.

[10] Z. Yang, D. Yang, C. Dyer, X. He, A. J. Smola, and E. H. Hovy, "Hierarchical attention networks for document classification.," 2016.

[11] H. Chim and X. Deng, "Efficient phrase-based document similarity for clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, pp. 1217–1229, Sept 2008.

[12] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1188–1196, 2014.

[13] J. Mueller and A. Thyagarajan, "Siamese recurrent architectures for learning sentence similarity.," 2016.

[14] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, vol. 242, pp. 133–142, 2003.

[15] F. Debole and F. Sebastiani, "Supervised term weighting for automated text categorization," in *Text mining and its applications*, pp. 81–97, Springer, 2004.

[16] S. Zhang, D. Zheng, X. Hu, and M. Yang, "Bidirectional long short-term memory networks for relation classification.," in *PACLIC*, 2015.

[17] D. Tang, B. Qin, and T. Liu, "Document modeling with gated recurrent neural network for sentiment classification.,"

[18] D. Tang, F. Wei, N. Yang, M. Zhou, T. Liu, and B. Qin, "Learning sentiment-specific word embedding for twitter sentiment classification.," in *ACL (1)*, pp. 1555–1565, 2014.

[19] T. Shi and Z. Liu, "Linking glove with word2vec," *CoRR*, vol. abs/1411.5595, 2014.

[20] J. Suzuki and M. Nagata, "A unified learning framework of skip-grams and global vectors.," in *ACL (2)*, pp. 186–191, 2015.

[21] Y. Goldberg and O. Levy, "word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method," *arXiv preprint arXiv:1402.3722*, 2014.

[22] L. Bottou, "Stochastic gradient learning in neural networks," *Proceedings of Neuro-Nımes*, vol. 91, no. 8, 1991.

[23] G. Hinton, "Lecture 10 Recurrent Neural Networks," 2013.

[24] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[25] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," *CoRR*, vol. abs/1211.5063, 2012.

[26] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, 2016.

[27] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional {LSTM} and other neural network architectures," *Neural Networks*, vol. 18, no. 5-6, pp. 602 – 610, 2005. {IJCNN} 2005.

[28] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[29] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, vol. 18, no. 5, pp. 602 – 610, 2005. IJCNN 2005.

[30] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[31] J. Weston and C. Watkins, "Multi-class support vector machines," tech. rep., Technical Report CSD-TR-98-04, Department of Computer Science, Royal Holloway, University of London, May, 1998.

[32] A. Statnikov, C. F. Aliferis, D. P. Hardin, and I. Guyon, *A Gentle Introduction To Support Vector Machines In Biomedicine: Volume 1: Theory and Methods*. World Scientific Publishing Co Inc, 2011.

[33] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, (New York, NY, USA), pp. 144–152, ACM, 1992.

[34] P. Neculoiu, M. Versteegh, and M. Rotaru, "Learning text similarity with siamese recurrent networks," 2016.

[35] J. Mueller and A. Thyagarajan, "Siamese recurrent architectures for learning sentence similarity," in *AAAI*, 2016.

[36] H. Schmid, "Probabilistic part-of-speech tagging using decision trees," 1994.

[37] H. Schmid, "Improvements in part-of-speech tagging with an application to german," in *In Proceedings of the ACL SIGDAT-Workshop*, pp. 47–50, 1995.

[38] S. Ruder, "On word embeddings - Part 3: The secret ingredients of word2vec," 2015.

[39] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.

[40] J.-P. Fauconnier, "French word embeddings," 2015.

[41] M. Sundermeyer, R. Schlüter, and H. Ney, "Lstm neural networks for language modeling.," in *Interspeech*, pp. 194–197, 2012.