Abstract

Ever since IBM came up with the idea of developing a system capable of being the best at Jeopardy!, back in 2004, Watson has been through many changes. After its victory against the best players ever on the show, Watson has changed from a system built only to answer to clues on a contest, to a system capable of answering almost any question. But, despite its success, no comparison between Watson and other NLP toolkits has been performed. Because of this, we propose a formal evaluation of Watson’s capabilities, comparing its performance with that of another toolkit, over a set of predefined NLP tasks. IBM provides access to Watson’s API through an online platform called Bluemix, which was used to develop part of the solution. Insights over the existing NLP tools are presented. Details about the implementation of the chosen tasks are discussed, along with the evaluation methodology that was followed and the results that were obtained.

Keywords: IBM Watson, Natural Language Processing, Bluemix, Natural Language Processing Toolkits
tasks proposed, and these must be open-source. All toolkits that do not meet the criteria mentioned before, will not be considered.

2.2. IBM Bluemix

Bluemix is IBM’s online platform, and with it developers can develop web applications without needing to do all the underlying setup or worry about the infrastructure. Of the many available services in Bluemix, we are more interested in the ones that use the API of Watson. This API provides multiple services, but those that are of greater interest to this work are the Watson Assistant service, the Natural Language Understanding service and the Natural Language Classifier service. The first service will be used to implement a natural language interface for database, since it can be used to recognize intents and entities mentioned by users, and the other two can be used to perform NER and sentiment analysis. In depth descriptions of all services provided by Bluemix can be accessed here\(^4\).

2.3. NLP toolkits

2.3.1 Microsoft Azure

Much like IBM with Bluemix, Microsoft has an online platform, called Microsof Azure\(^5\), that offers a wide array of services for users to build applications, or integrate them in existing software. The AI + Cognitive Services category of services offers a wide scope of AI related functionality, such as image processing, speech recognition and processing, knowledge related services, and, more importantly for this study, NLP related services\(^6\).

2.3.2 Google Cloud Natural Language API

Google has an online platform for the development of software in the field of NLP called Google Cloud Natural Language API\(^7\). This platform offers services that allow users to perform some basic NLP tasks, such as tokenization, sentence splitting, PoS tagging, dependency parsing, sentiment analysis and NER. Several languages are supported by this platform, depending on the tasks the user wants to perform.

2.3.3 NLTK

NLTK\(^8\) is a collection of open-source programs, modules and data sets written in Python, originally created in the Department of Computer and Information Science at the University of Pennsylvania. It can run on all platforms that support Python\(^2, 5\). Nowadays, NLTK has been used widely as a teaching aid, or as an individual learning tool for students and researchers in the field of NLP\(^2\). It comes with over 50 corpora and some other lexical resources, such as Wordnet\(^9\). Also, it possesses modules to perform several core NLP tasks, such as tokenization, tagging, stemming, parsing, among others. A detailed explanation of all the modules and their functions, can be found on the tool’s website\(^10\).

2.3.4 Stanford CoreNLP

Stanford CoreNLP\(^11\) is a Java toolkit developed by the Natural Language Processing Group at Stanford University\(^12\). CoreNLP is one the most used NLP toolkits, being it for educational and research purposes, or for commercial and government applications\(^6\). It provides implementations of several core NLP tasks, including, tokenization, PoS tagging, sentence splitting, NER, stemming, coreference resolution, syntactic parsing, gender identification, and sentiment analysis. An annotation pipeline framework was adopted in this toolkit. Text in natural language is put into an Annotation object. Then, a sequence of analyses is performed using different annotator objects (each NLP task is embedded in a different annotator). All the annotators have some options that the users can change, which will change aspects of their functionality.

2.3.5 OpenNLP

OpenNLP\(^13\) is a free Java library, meant to process text written in natural language. This toolkit supports several NLP tasks, like sentence segmentation, PoS tagging, tokenization, NER, parsing, chunking and coreference resolution\(^14\). OpenNLP is composed of several components, each one of them enabling the user to perform a certain NLP task, to train a model and test it. This can be done using API calls, or through a provided command line interface.

2.3.6 SpaCy

Developed by Explosion AI\(^15\), SpaCy\(^16\) is an open-source Python library meant for performing NLP tasks, such as tokenization, PoS tagging, dependency parsing, NER, and sentence segmentation.

---

\(^4\)https://console.ng.bluemix.net/catalog/
\(^5\)https://azure.microsoft.com/pt-pt/
\(^6\)https://azure.microsoft.com/pt-pt/services/cognitive-services/
\(^7\)https://cloud.google.com/natural-language/
\(^8\)http://www.nltk.org/
\(^9\)https://wordnet.princeton.edu/
\(^10\)http://www.nltk.org/api/nltk.html
\(^11\)https://stanfordnlp.github.io/CoreNLP/
\(^12\)https://nlp.stanford.edu/
\(^13\)https://opennlp.apache.org/
\(^14\)http://opennlp.apache.org/docs/1.7.2/manual/opennlp.html
\(^15\)https://explosion.ai/
\(^16\)https://spacy.io/
SpaCy can be used for software production, and actual real-life situations, contrary to the research and educational purposes of most other tools.

2.3.7 Other open-source NLP toolkits

There are many other NLP toolkits being used by community nowadays, to the point we cannot cover all of them in this paper. But, having in mind the context of our work and the goals we want to achieve, these are the the ones that are part of our list of candidate toolkits:CogCompNLP, DELPH-IN, Freeiding, General Architecture for Text Engineering (GATE), Gensim, LinguaStream and MontyLingua.

2.4. Choice of NLP toolkit to use

In light of all the requirements mentioned in Section 2.1, some toolkits were immediately discarded, either because they are paid toolkits, or because their focus is on other specific fields, less related to the proposed tasks. Another aspect that influences the decision, is how up to date are the toolkits. Software that is not updated in a long time (3 or 4 years) is not desirable. Due to the fact that the date of their last update was not found, or was a long time ago, DELPH-IN and MontyLingua were ruled out. Popularity is also an important factor to take into account in our decision. After consulting several people in the field of NLP, and searching online for papers that use, or make reference to the toolkits mentioned in this section, the ones that stand out are NLTK and Stanford CoreNLP. Almost all remaining toolkits were ruled out, mainly because they do not offer the necessary functionality to implement all tasks, leaving us with three remaining candidates: NLTK, Stanford CoreNLP and SpaCy . The final choice ended up falling upon NLTK, due to its higher degree of popularity among researchers and students, and also because it provides all the tools necessary to implement the three chosen tasks.

3. Sentiment Analysis

Generically, sentiment analysis consists of analyzing the polarity of texts[3]. The goal is to see if the texts transmit a more positive sentiment, or a more negative one. In this study, we will be performing sentiment analysis over texts that are reviews or opinions about movies, and over social media texts (tweets).

3.1. Corpora Selection

We chose to use corpora that is already part of NLTK’s pool of NLP related resources, and are specifically made for the task of sentiment analysis. Two different corpora were used for this task. The first is the movie review corpus. It consists of 1000 movie reviews labeled “positive” and 1000 movie reviews labeled “negative”. The other is the Twitter corpus. As the name suggests, it consists of a set of tweets taken from Twitter17 interactions, each labeled “positive” or “negative”. There are 5000 positive tweets and 5000 negative tweets.

3.2. NLTK implementation

After choosing which corpora to use, the next choice was which classifier to use. There were some options to choose from, such as a Naïve Bayes, Maximum Entropy, or a Decision Tree classifier. We chose to use the Naïve Bayes classifier, due to its simplicity. This classifier accepts as input a set of features for training or classifying purposes. In our case, four word feature generating functions were implemented. The first and simplest, the baseline feature function, generates word features using all the words in each review or tweet (only unigrams), with no additional filtering. The second feature function does the same as the previous function, but filters out stopwords contained in NLTK’s English stopword set, excluding the words “no”, “nor” and “not”. Next is the most significant bigrams feature function, which uses NLTK’s chi-square scoring function to score bigrams in the text provided as input. Only the 200 most significant bigrams for the input text are considered. The last and most complex function, in addition to what the previous one does, considers only the 10000 most common words used in the training set, and computes the 200 most significant bigrams across the training set.

Since our goal with this task is to compare the performance of NLTK and Bluemix in the execution of sentiment analysis, we did not focus in performing very complex feature engineering. Instead, we focused on implementing simple, but effective features, which will allow us to perform this comparison.

3.3. Bluemix implementation

To implement this task in Bluemix, the NLU and NLC services were used, leading to two separate sentiment analysis systems. IBM does not specify which classifier is used in either service. Additionally, both services only classify one string at a time, leading to large execution times. Implementing this task using the NLU service was straightforward. The first step was to retrieve the raw text from both corpora, which is what the service accepts as input. This involved some manipulation of the Twitter corpus, which was simple to bypass using NLTK’s built-in functions to manipulate corpora. After having all the input ready, we passed all the strings (individual reviews or tweets) to the service, and registered the top classes indicated in

17https://twitter.com/?lang=pt
its response.

Regarding the implementation using the NLC service, we can say that it had a higher degree of complexity, due to some of the service’s restrictions regarding the data it accepts as input. One of the restrictions was that it only accepts as input strings with a maximum of 1024 characters. On the movie corpus there are less than 15 reviews that have under 1024 characters, meaning this corpus could not be used with the NLC service. The other restrictions had to do with the training data. It must be passed to service in the form of a CSV file, with a large number of restrictions.

Despite these restrictions, the execution flow is quite simple. The first step is to retrieve the raw text from the Twitter corpus. After that, the training file is produced, according to the restrictions mentioned. Having produced the file, it is then passed on to the service, which will train the classifier. After the classifier has passed the training stage, the remainder of the corpus is passed as input, one tweet at a time.

3.4. Evaluation Methodology

We used an evaluation methodology that leverages, not only on the performance of the systems when performing the NLP task considered, but also a series of other factors related to ease of use and functionality. To measure the systems’ performance, precision, recall, f-measure, and accuracy will be used.

The other factors that will be taken into account are more related to the development stage of NLP tasks. The first one has to do with the difficulty the developer experiences when installing the chosen toolkit. If this is a complex activity, using the toolkit in question is less appealing, so we will take this factor into account. The next factor regards the toolkit’s documentation. First time users most likely do not know how the toolkit works. In these cases, most times developers recur to the documentation available, making it important to factor in if the documentation is clear and easy to read, or not. The last factor has to do with what we call community support. When users encounter difficulties developing their application, besides the documentation available, they resort to the Internet to find a solution. So, we will take into account if it is easy to find online useful information about how to implement a specific task, using the toolkit chosen.

3.5. NLTK Test Results

We used an n-fold cross validation strategy, with a total of 72 tests, 36 to each corpus, combining each feature function, with the execution of cross validation using 2 to 10 folds. After executing all tests, we can say that the best results, for the movie corpus, were obtained using the most complex features, that include only the bigrams with the best score among the 10000 most common words in the training set. For any of the tests with 10 folds, all measures exhibit values of approximately 83%, a major over the other features. Regarding the Twitter corpus, we can say that more consistent results were achieved across all feature functions, with a minimum of almost 75% in all measures using the baseline features, and a maximum of 76,5% using the most complex features. Once again, the best results were achieved with the same feature function.

3.6. Bluemix Test Results

Starting with the NLU service, the evaluation methodology adopted was to pass the whole corpora to the service, for it to classify all the reviews and tweets present in them. Both of Bluemix's services used in this task took a long time during classification or training (for the NLC service). More specifically, for the movie review corpus, the NLU service took 26 minutes to classify all reviews, whilst for the Twitter corpus, it took 125 minutes. Regarding the test results, the NLU service obtained, for the movie corpus, an accuracy of almost 79%, precision of 96,5% and recall of 59,9%. As for the Twitter corpus, the results are worse than with the movie review corpus, with all measures ranging between 53,5% and 57%.

Moving to the NLC service, since this service requires training data, the methodology was somewhat different. We chose to perform several tests, in each one choosing a predefined number of tweets from the corpus, at random, and using these random tweets to train the classifier. The execution times are quite long, ranging from a minimum of two hours and six minutes to a maximum of two hours and 28 minutes. Regarding the test results, they are more or less similar to the ones achieved using the baseline word features in NLTK.

3.7. Conclusions

Regarding execution time, the advantage goes to NLTK. With it, users can train and test classifiers in a matter of seconds, whereas with Bluemix, this can take several hours. As for the system’s performance, better results were achieved using NLTK, although it requires additional additional complexity. The installation of all the toolkits and libraries for this task, for both toolkits, is quite simple and fast, with neither toolkit having an advantage in

\[18\text{https://console.bluemix.net/docs/services/natural-language-classifier/using-your-data.html#data-preparation}\]

\[19\text{https://console.bluemix.net/docs/services/natural-language-classifier/using-your-data.html#using-your-own-data}\]
this field. NLTK’s documentation, in this case, was more enlightening and provided more detail than the documentation of the NLC and NLU services, giving NLTK an advantage in this field. Finally, regarding community support, NLTK has a clear advantage in this field. Having all these factors in mind, we can say that NLTK is a better choice than Bluemix for the implementation of a sentiment analysis system.

4. NER
NER consists in assigning pre-defined categories to words in a text. There are many categories that can be attributed, some of the most common being: location, person, organization or time.

4.1. Corpus selection
Due to the fact that NLTK does not have a tagged English corpus for NER, we had to search other resources available publicly. We ended up choosing the GMB corpus. It corpus is composed of over 60000 tagged sentences, with each word belonging to one of the following categories: art(artifact), eve(event), geo(geographical entity), gpe(geopolitical entity), nat(natural event), org(organization), per(person), tim(time), or 0(none of the previous). Besides these main categories, each word is tagged with a subcategory, but, we chose not to include these subcategories in our study.

4.2. NLTK implementation
We used NLTK’s CRF Tagger, which is based on the implementation of a CRF model in Python’s pycrfsuite library. This tagger takes as input features generated by a feature generation function, implemented by the user, which will be used to choose the most likely tag for each word. Additionally, we chose to implement three different feature generation functions.

The baseline function considers only the current word being passed to it, without any information about any previous or upcoming words. Regarding the features being extracted from each word, this function returns the word being analyzed, its stemmed form and PoS tag, along with a series of truth values that indicate: if the word contains a dash, a dot, if it has only capital letters, if it’s capitalized, and if it contains only letters. The next feature function generates the same features as the previous one, with the difference it also takes into account the previous and the next word. As so, it generates the same features but for these three words. The final feature function does the same as the two previous functions, with the addition it has access to a greater history (two previous words) and to the two next words.

4.3. Bluemix implementation
Looking at Bluemix’s list of services available and after analyzing what each of them offers, we chose to use the NLU service to implement this task. As it does not require any training, the execution flow is quite simple.

The first step is to retrieve the raw text from the corpus, which is what is passed as argument to the service. Having the text ready, the next step was to divide the raw text into chunks of 5 sentences each and pass each chunk to the service. As the results were returned by the service, the categories identified had to be “translated” to the ones present in the corpus. This was the most difficult part to implement, requiring much time analyzing what categories were returned by Bluemix, and which ones could be “renamed” to match the categories in the corpus. For it to be possible to evaluate and compare the results achieved with NLTK, this step had to be done. Finally, having recorded all the entities recognized and their categories, accuracy, precision and recall were calculated, the same way they were calculated for NLTK.

4.4. Evaluation Methodology
To evaluate NLTK’s performance in this task, we will use an n-fold cross validation strategy, with a total of 27 tests, 9 for each feature generation function, varying the number of folds from 2 to 10 folds. The formulas to calculate precision, accuracy and recall, are different from the ones used in sentiment analysis. We chose to follow one methodology which takes into account both the named entities’ text boundaries and the categories assigned to them. This is the scoring system used in Message Understanding Conference (MUC) events[7].

Using this scoring system there are two events that have to be recorded. The first one (correct type) is when the system assigns the correct category to a named entity, as long as there is an overlap between the entity recognized by the classifier, and the original one. The second (correct text) is when the boundaries of the named entity recognized by the system and the original one are the same, regardless of their types. For both these events, three measures must be kept:

- COR - the number of correct answers. Increases when a correct type or text are recorded;
- ACT - the number of guesses made by the system. This consists in the number of texts plus the number of types that were guessed by the system;
- POS - the number of texts and types present in the corpus.

http://gmb.let.rug.nl/
Precision can be defined as:

\[ \text{Precision} = \frac{\text{COR}}{\text{ACT}} \]

Moving on to recall, it can be defined as:

\[ \text{Recall} = \frac{\text{COR}}{\text{POS}} \]

The f-measure is calculated the same way it was for sentiment analysis. The formula adopted for accuracy had in mind the definition of accuracy in the binary classification scenario. In that case accuracy can simply be expressed as the number of true positives plus true negatives divided by the number of data samples. Adapting this concept for this particular task, accuracy can be defined as:

\[ \text{Accuracy} = \frac{\text{COR(\text{TEXT}) and COR(\text{TYPE})}}{\text{ACT}} \]

Finally, for the NLU service, the categories of named entities that this service recognizes are different from the ones present in the GMB corpus. After some experimentation we came to the conclusion that the the Quantity category (returned by the NLU service) and the \textit{gpe} and \textit{tim} categories, present in the corpus, had no “translation”, and so, they were not considered during evaluation.

4.5. NLTK Test Results

Starting with the execution times, they ranged from almost 3 minutes using the baseline function with 2 folds, to approximately 61 minutes using the feature function that generates the most features with 10 folds. Moving on to the test results regarding performance, the best results were achieved with the feature function that considers the current word, along with the two previous and two next words. It registered the best values for all measures except precision, although the difference to best value registered is very small. More specifically, with this feature function the system achieved an accuracy of 82.37%, a precision of 88.8% and a recall of 88%.

4.6. Bluemix Test Results

Regarding the execution time obtained with the NLU service, feeding it chunks of 5 sentences in each request, it took approximately 2 hours and 38 minutes to process all the sentences in the corpus. As for the test results regarding performance, there is a significant decrease in accuracy (of almost 20%) in relation to what was obtained with NLTK. Precision also displayed a decrease of approximately 16%, in relation to NLTK. Recall registered a less significant drop, of roughly 9%, in relation to the best results achieved with NLTK.

4.7. Conclusions

Having in mind the first factor (execution time), a very clear difference can be seen, with much better execution times being achieved using NLTK. Moving on to performance, comparing the results obtained using both toolkits, we can see that there is a clear advantage for NLTK across all measures. Despite having the values for each measure, since some categories could not be used in this study, these results may not be most reliable to measure the difference in performance. But, considering the categories that were not excluded, NLTK has the advantage. What was said in Section 3.7 regarding documentation still holds for this task. Both NLTK and the NLU service, have extensive documentation to help developers, but, sometimes the documentation provided by IBM just does not cover all the details, giving NLTK a slight advantage in this field. The same happens for community support. Due to NLTK’s popularity, the amount of information that can be found online is much greater than for any of Bluemix’s services.

Having in mind these key points, it can be said that both toolkits have their advantages and drawbacks, with the best choice falling upon NLTK, to implement a NER system. But, if the user does not possess a tagged corpus for training, and is not sure as to which categories of entities he wants the system to recognize, or if he wants a tool which does not involve a lot of code complexity, using Bluemix seems a preferable option, despite the difference in performance.

5. Natural Language Interface for Database

With a natural language interface for database, users can ask questions using natural language, and the system will query its underlying database for the answer[4].

![Diagram of adopted architecture](image)

**Figure 1:** Adopted architecture for the systems developed

Figure 1 shows the high level architecture we have adopted. The first step is to detect what is the user’s intent, and the named entities mentioned the question. Next, a logical form is created, which encapsulates the semantics of the user’s question, and can be easily mapped to a query. Finally, after the query is generated, it is executed by the database and the results returned are showed to the user. Before implementation, the first step taken was to decide which domain to use. After some online research, we ended up choosing the cinema domain.
5.1. Dataset
We chose to use the TMDB 5000 Movie Dataset\(^2\), available on Kaggle\(^2\). This dataset comprises information about approximately 5000 movies, contained in two CSV files. The first file is the movies file, and it contains information about the movies themselves. The other CSV file is the credits file. It contains information only about the people that take part in the production of each movie (the movie’s cast and crew). The next step to be taken is to migrate the data from the dataset to a database.

5.2. Database creation
To create a usable database, the first step was to make a high level design of how the data should be organized across all tables.

5.2.1 Conceptual schema
There was already previous work done by a former student of IST, Ana Guimarães, regarding a natural language interface for database for the cinema domain. Although the database used in Ana’s work\(^4\) contained different information than our dataset, we could see that there were some overlaps in some cases, and so we decided to use some ideas of her design in ours. The final database is composed of 12 tables, with the movies table being the one that contains most information on the movies CSV file. Since some of the information on this file is in JSON, some additional tables were needed for these fields, all of them with foreign keys to the movies table.

5.2.2 Database Creation
The next step was to create all the 12 tables and populate them, resulting in the final database. To create the model that contains the database, MySQL Workbench\(^23\) was used. After creating the model, the final step was to populate the tables with the data from the two CSV files. The most efficient solution for this was to write a program that processes and extracts all the data from the CSV files, and then inserts all the data into the appropriate tables. To access and alter the database created earlier, Python’s MySQLdb\(^24\) library was used.

5.3. Corpora
The final step needed before starting to implement the system is to create two corpora: one for training and one for testing. We created a training corpus that consists of 298 questions that cover most of the information on the database. In addition to this corpus, a test corpus was compiled. We asked people to write questions about the cinema domain, filtered the ones that the database could not answer, and used the remaining ones for the test corpus. It is smaller than the training corpus, having only 40 questions.

5.4. The Voting Model
There are many approaches to build a natural language interface, with the main component being a semantic parser. We chose to use the so-called Voting Model (Bhagat et al., 2005). The premise of this model is that, using statistical learning strategies, it achieves satisfactory results with very little training data. Due the fact that the model is not very complex, and simple to understand, led us to choose it over others.

5.4.1 Data representation
One the key concepts of this model is the concept of frame. It is what the parser outputs and consists of a set of slot-value pairs\(^1\), with each pair being called a “meaning element” or “frame element”.

Entities of the same type can all be grouped into the same category (Movie or Actor categories for instance). With this categorization, it is simpler to represent meaning, by replacing entities that are not important to the semantics of the sentence as a whole, with their corresponding categories.

5.4.2 Training data
The training data is a corpus made of sentences, with manually annotated frames for each sentence. In our case, we used the training corpus already created that was presented in Section 5.3. However, we still had to manually annotate each sentence with its respective frame.

5.4.3 Training the parser
This model uses statistical learning methods to output the final frame. More specifically, conditional probability models are used to calculate the probability of producing a slot-value pair \( f \) as output, given that the parser has seen a certain word or n-gram \( W \) in the input sentence\(^1\). With:

- \( C(f_i | w_j) \) being the number of times the frame element \( f_i \) is seen as output in corpus, given that the sentence contains the word \( w_j \);
- \( C(f_i | w_{j-1} w_j) \) being the number of times the frame element \( f_i \) is seen as output in corpus, given that the sentence contains the bigram \( w_{j-1} w_j \);
- \( C(f_i | w_{j-2} w_{j-1} w_j) \) being the number of times the frame element \( f_i \) is seen as output in cor-

\(^1\)https://www.kaggle.com/tmdb/tmdb-movie-metadata/
\(^2\)https://www.kaggle.com/
\(^3\)https://www.mysql.com/products/workbench/
\(^4\)http://mysql-python.sourceforge.net/MySQLdb.html
pus, given that the sentence contains the trigram \( w_{j-2}w_{j-1}w_j \).

The Voting Model calculates the following probabilities:

\[
P(f_i \mid w_j) = \frac{C(f_i \mid w_j)}{\sum_{k=1}^{n} C(f_k \mid w_j)}
\]

(1)

\[
P(f_i \mid w_{j-1}w_j) = \frac{C(f_i \mid w_{j-1}w_j)}{\sum_{k=1}^{n} C(f_k \mid w_{j-1}w_j)}
\]

(2)

\[
P(f_i \mid w_{j-2}w_{j-1}w_j) = \frac{C(f_i \mid w_{j-2}w_{j-1}w_j)}{\sum_{k=1}^{n} C(f_k \mid w_{j-2}w_{j-1}w_j)}
\]

(3)

These calculations are done for every frame element present in the training corpus, and all the values must be stored, to be later consulted. This model does not record any other type of dependency between words in sentences.

### 5.4.4 Sentence parsing

This is the final step, where the parser uses the probabilities calculated during training to return the most likely frame elements to be part of the final frame. The process is composed of two stages. During the first stage, the parser finds the most suitable candidates for each word, bigram or trigram in the input sentence. This is done by assigning a weight to each frame element existing in the training corpus. The weight of frame element \( f_i \) can be calculated using the following expression:

\[
Wt(f_i) = \sum_{j=1}^{n} P(f_i \mid w_j) + \sum_{j=2}^{n} P(f_i \mid w_{j-1}w_j)
\]

\[+ \sum_{j=3}^{n} P(f_i \mid w_{j-2}w_{j-1}w_j)\]

This measures how likely it is for a frame element to be part of the final frame. The second stage consists of selecting from the list of candidates, the ones that are most likely to be the correct. This selection is done in three stages. On the first stage, for each different attribute from the candidate frame elements, keep only the candidate with highest weight. On the second stage, for each different value from the candidate frame elements, only the candidate with the highest weight is kept. The final stage consists only of applying a cutoff to the remaining candidates.

5.5. Natural language interface implementation using the Voting Model and NLTK

The first step was to create lists with named entities, to be used later on the processing of sentences. These were retrieved from the database using Python's MySQLdb library. Next, all the questions from the corpus, along with their respective frames, were stored in lists, by the order they appear on the corpus. But, the questions and frames had to suffer some changes to be in the correct format for the training stage.

The first change consisted of identifying and replacing entities with their respective categories. This was done on both questions and frames, resorting to the lists of entities created previously. In addition to these lists, we used AhoCorasick string matching algorithm (Aho and Corasick, 1975) to make sure the entities are recognized and replaced correctly. The next step was to remove stopwords from all the questions in the corpus, contributing to smaller and simpler sentences. We used NLTK’s English stopword set to perform stopword removal. The last change was to stem the remaining words, minus the categories that replaced the named entities detected earlier. We chose one of NLTK’s stemmers, the Snowball Stemmer, to perform this task.

The corpus is now stored in its final format, ready to be used to train the model. The next step is to calculate all the probabilities presented in Section 5.4.3 and store them in dictionaries. Once this is done, the system is ready to receive user input. To implement this part of the model, the same steps had to be taken when processing the training corpus. Namely, replacing named entities with their respective categories, removing stopwords and stemming. After this we proceed to calculate weights for the input sentence, yielding a list of frame element candidates ordered by descending weight that will be then filtered. This leads us to the final step, the conversion of the frame into the correct query. This was done through a mapping of each different set of frame elements to the corresponding query. The mapped queries only lack one or two necessary arguments. After converting the frame to the proper query, the only thing left to do is execute the query and show the results to the user.

5.6. Interface implementation using the Watson Assistant service

To implement this task using Bluemix we used the Watson Assistant service, which allows developers to identify different types of entities and intents in user input. Using this service, the first step that has to be taken is to define the intents the service should recognize.

\[\text{https://console.bluemix.net/catalog/services/watson-assistant-formerly-conversation?taxonomyNavigation=watson}\]
5.6.1 Intent creation

The intents defined applied to the questions used on the training corpus created earlier. A total of 40 different intents were created through the service’s web application. All that had to be done to define the intents was to name each one as it was created, and then provide sentence examples for that intent. For all the intents defined, a minimum of 5 examples were given for each one, reaching a maximum of 7 on one of them.

5.6.2 Addition of entities

Our interface recognizes four types of entities: Movie, Actor, Director and Character. For the interface to function properly, a large amount of examples had to be provided for each category, since the service does not recognize any entity of these types by default. Using the service’s API, a total of 100 000 examples were provided, all of them coming from our database.

5.6.3 Receiving and processing user input

After defining all the necessary intents and entities, the system must now process the inputs it receives. For this to be possible, we must first map each intent to a unique query, lacking only the arguments needed to execute it. This must be done before the system receives any input. After the user submits his question, it is passed to the Watson Assistant service and the intent and entities detected are stored. Finally, the system fills in the query mapped to the detected intent, executes it, and shows the results to the user.

5.7. Evaluation Methodology

The strategy adopted was to ask the questions present in the test corpus to each interface, and check the logical expression that was created, not letting the systems transform this logical expression into a query and execute it. The measures used are somewhat different than the ones already used, with two of them (precision and recall) being the same as the ones used by the authors of the Voting Model[1] to evaluate their model. The first measure is precision, and consists in the percentage of slots returned by the interface that were correctly filled, out of the total number of slots returned. The second measure is intent precision, and it is slightly different from precision, with the difference being this one only considers a sentence’s intent. The third measure is recall, and consists in the percentage of correctly filled slots, out of the number of slots present in the test corpus. The final measure is called the number of correct sentences. It is the number of questions in which the interface’s response contained all the slots in the test corpus, for that particular sentence.

5.8. NLTK and the Voting Model Test Results

Out of the 129 frame elements returned by the interface, 83 of them were filled in correctly, leading to a precision value of almost 64.5%. And, taking into account only the number of frame elements present in the corpus, the value is even higher, reaching 73.45% for recall. Also, the system managed to identify the correct intent for 31 of the 40 questions in the corpus, leading to a precision of 77.5% on the task of intent identification. Despite this percentage, the system only managed to output all the necessary correct frame elements in 23 out of the 40 questions. This indicates that the interface had more difficulties in identifying named entities correctly, than the users’ intents.

5.9. Bluemix Test Results

The Watson Assistant service output a total of 110 slots, a number slightly lower than what the Voting Model returned. Also, the number of correct slots was lower in this case, with only 67 correct slots, leading to a precision of almost 61%. This is a similar value to the one obtained with the Voting Model, although somewhat lower. Despite its lower precision, the service managed to identify the sentences’ intent more frequently, with 35 out of the 40 sentences having the correct intent attributed to them. This led to an 87.5% of precision, on the intent level, a 10% increase in comparison with the Voting Model. The elevated number of correct intents reflects itself on the number of correct sentences, increasing from 23 with NLTK, to 30 using the Watson Assistant service. This also lead to an increase in recall, as the system managed to output correctly 87% of the slots present in the test corpus.

5.10. Conclusions

The Voting Model and NLTK achieved better execution times, taking roughly 1.5 to train and process the 40 sentences contained in the test corpus. For the same sentences, Bluemix managed to process them in 24 seconds, granting NLTK and the Voting Model an advantage in this aspect. Comparing the values obtained regarding performance, a slight advantage can be seen when the Watson Assistant service is used. With it there is a sizable increase in intent precision, number of correct sentences, number of correct intents, and recall, despite the small drop in precision.

Another factor that must be taken into account for this particular task is the robustness of the solution, since it is the most complex task. The Watson Assistant service is the least complex, in terms of code. But, with this simplicity comes a disadvantage in relation to the Voting Model. Only one slot is dedicated to a sentence’s intent when using the Watson Assistant service, whereas with the Voting Model, there can be multiple slots dedicated to the
intent. This means that, with Bluemix, no granularity can be achieved in terms of intents. Although this works when there is a relatively small number of intents, as this number increases, this is not appropriate. This makes developers having to define a new intent every time they want to introduce variations in existing intents. The Voting Model and NLTK have the advantage in this aspect. Regarding documentation and community support, the same applies for this task, NLTK has the advantage in these two factors.

Having in mind all these factors, we can say that each solution as its advantages and disadvantages. On one side, with the Watson Assistant service users can quickly define intents and provide entity examples, with no training required on the developer’s end. On the other hand, NLTK provides an additional level of complexity and solidity, which can become useful for large training corpora. Because of this, there is no clear choice between the Watson Assistant service, and NLTK and the Voting Model.

6. Conclusions and Future Work
Having implemented all the proposed tasks, and tested each of the systems developed, we now have a clearer idea of Watson’s capabilities regarding these three tasks. If performance was a determining factor, then NLTK would have the upper hand over Bluemix. But, sometimes, this is not only factor taken into account when choosing a toolkit. For all the three tasks implemented, we used services that did not need any training or implementation of complex models. This a very positive factor for Bluemix, since many users just want to perform small and simple experiments. But, having in mind all the factors analyzed throughout this work, the overall advantage goes to NLTK. With it, users can achieve better results, they can develop their systems for free, and can find much more useful information online. Although the simplicity that can be achieved with Bluemix is a very positive factor, it does not outweigh the others.

Future work possibilities include performing another comparative study, with different NLP tasks. This would give a bigger picture about Bluemix’s performance in relation to NLTK’s, and also about the other factors discussed throughout this work. Another possibility is to implement the same tasks but with another NLP toolkit. This would allow for an even broader comparison of Bluemix’s services, having in mind not just the results achieved with NLTK, providing a greater depth to this comparative study. Lastly, Bluemix is constantly updating its services, either changing existing services or creating new ones. As such, there is always the possibility of trying out new services, or check if the already used ones provide better results, as time passes.

References