The King's Christmas Speech corpus

Elena Álvarez Mellado

December 2018

1 Introduction

Every Christmas Eve, the king of Spain delivers a speech to the nation where he shortly addresses some of the public issues that affect the country. This tradition began on 1975, just a few weeks after the dead of fascist dictator Francisco Franco, when democracy in Spain was still rather a hope than a reality. This traditional speech continues up to today, which means that we have had 43 Christmas speeches so far.

These speeches are a very interesting piece of both linguistic data and sociopolitical information: through the analysis of keywords, n-grams and lexical frequency shifts over time we can have a portrait of some of the changes on political matters, social concerns and public speech discourse that have taken place in Spain during the last 43 years. In addition, the former king of Spain that started this tradition (Juan Carlos I) abdicated in 2014 after a few years of scandals and growing unpopularity. His son Felipe VI was then crowned king and has maintain the tradition of the Christmas Eve speech. This recent change of king and the late drift of public opinion concerning the Spanish monarchy make the comparative analysis of these speeches particularly interesting.

The aim of this project is to generate a linguistic corpus that gathers the 43 Christmas speeches delivered so far from 1975 to 2017 (39 speeches delivered by king Juan Carlos I, 4 speeches delivered by his son king Felipe VI). Although the speech is massively followed, analyzed and commented nationwide, there is no easy or comfortable interface that allow to interact and aggregate the data from the different speeches. The speeches are published on the royal family website, but there is no available corpus that allows data retrieval and linguistic analysis in the same way as the Inaugural Address Corpus from the `NLTK` library. Therefore, the aim of the project will be to create the programming infrastructure needed to query the corpus and retrieve lexical information from it, either individually (one speech only), aggregating the information from different years or globally on the entire speeches corpus as a whole.

Secondly, we will use various libraries to obtain visualizations to compare the differences between time periods and kings' styles using frequency measures such as the TF-IDF. Finally, we will use the corpus as training material for an automatic speech generator.

The project can be found on the GitHub repository at `https://github.com/lirondos/orgulloysatisfaccion`. The name of the GitHub project (*Orgullo y Satisfacción*, literally "Pride and satisfaction") is a humorous nod to the common belief (and even national joke) that all royal speeches contain the phrase *me llena de orgullo y satisfacción* (which would translate to "it is with great pride and satisfaction"). Fun fact: contrary to the common belief, the phrase *orgullo y satisfacción* appears only once in the entire corpus.

2 Creation of the corpus

2.1 Web scrapping of the speeches

The first step is to compile the corpus. The corpus compilation was done by performing web-scrapping techniques on the royal family official website (`http://www.casareal.es/`). We used the Python library `newspaper` in order to retrieve all the speeches. The crawling script with the scrapped URLs can be found inside the Python project as `crawler.py` file. However, due to the poor infrastructure of the royal family website that causes recurring timeout errors, some of the information had to be manually supervised and corrected.

As a result, the compiled corpus consists of 43 speeches (one for every year since 1975), which makes a total of 63035 tokens. The 43 speeches can be found as individuals `txt` files under the project folder `speeches`.

2.2 Corpus model

Once the corpus is compiled, we need to create the programming infrastructure that will enable to query the corpus and analyze the linguistic data. Two classes where created to access and query the speeches: the `Speech` class and the `Corpus` class.

2.2.1 The `Speech` Class

The `Speech` class formalizes the structure and data related to every single speech. Every speech object has the following fields:

(a) the raw text of the speech

(b) the list of word tokens

(c) the list of word types

(d) the list of sentences

(e) the list of paragraphs

(f) a lemmatized and POS-tagged version of the text (this information is created using the Python library `SpaCy`)

(g) a `NLTK` Text object version of the text

(h) the year when the speech was delivered

(i) the king it was delivered by

(j) whether the speech date corresponds to the first half or the second half of the corpus in terms of time

(k) the historical period when the speech was delivered

The scrapped information (raw text and year) allows us to create fields a-h, while elements i-k are metadata related to the time and historical circumstances in which the speech was delivered. This metadata is created from the year the speech was delivered. Speeches that are previous to 2013 will be tagged with *Juan Carlos* as king (who was king of Spain between 1975-2014), while speeches from 2014 to 2017 will have *Felipe* as king.

In terms of historical periods, four periods have been considered bearing in mind the historical circumstances and events that have taken place in Spain from 1975 to nowadays:

- The period of time that goes from 1975 (the first year where the king speech took place, which is the same year in which fascist dictator Franco died) until 1982 is considered *transition*, as it correspond to the Spanish transition from a dictatorship to a stable democracy.
- The period of time that goes from 1982 to 1996 is called the *socialist period*. This period of time covers the years when the Spanish Socialist Party was in power.
- The period of time that goes from 1996 to 2008 are the years of the *economic bubble*. It starts with the victory of the Conservative Party in 1996 and ends with the beginning of the economic crisis.
- The period of time that goes from 2008 to nowadays are the years of the *economic recession*. During this historical period of time the previous king Juan Carlos I abdicated and current king Felipe VI was crowned.

These periods of time will become particularly relevant during the visualization section.

The methods within the `Speech` class query the speech and extract relevant lexical information from it. The `Speech` methods can be divided in three groups: methods that provide absolute lexical information (not filtered or ordered by frequency), methods that provide information filtered by or related to frequency and methods that query for information on a particular word in the speech.

1. Methods that return lexical information (on absolute terms):
   - `length()`: number of words in the speech.
   - `content_words()`: list of content words (words that do not appear on the stop word list).
   - `bigrams()`: list of bigrams.
   - `trigrams()`: list of trigrams.
   - `content_bigrams()`: list of content bigrams (bigrams where both words are content words, i.e. words not in the stopwords list).
   - `content_trigrams()`: list of content trigrams (trigrams where the first and third words are content words, i.e. words not in the stopwords list).
   - `longest_words()`: list of the longest words in the corpus.
2. Methods that return frequency distributions:
   - `frequencies()`: frequency distribution of all words in the speech.
   - `most_frequent_content_words()`: list of (word, frequency) pairs ordered on the frequency where word is a content words (words that do not appear on the stop word list).
   - `most_frequent_bigrams()`: list of ((word1, word2), frequency) elements where word1 and word2 are content words.
   - `most_frequent_trigrams()`: list of ((word1, word2, word3), frequency) elements where word1 and word3 are content words.
   - `hapaxes()`: list of hapaxes (words that only appear once in the speech).
3. Word query methods:
   - `word_appearances(word)`: number of times that `word` appears on the speech.
   - `concordance(word)`: concordances of that given `word` (contexts in which the word appear)
   - `similar(word)`: words that tend to appear in the same contexts as the specified `word`.
   - `dispersion_plot(words)`: creates a dispersion plot that display the appearences of a list of words.

Methods of the Speech class		
Lexical information	Frequency information	Word queries
`length()`	`frequencies()`	`word_appearances(word)`
`content_words()`	`most_frequent_content_words()`	`concordance(word)`
`bigrams()`	`most_frequent_bigrams()`	`similar(word)`
`trigrams()`	`most_frequent_trigrams()`	`dispersion_plot(words)`
`content_bigrams()`	`hapaxes()`	
`content_trigrams()`		
`longest_words()`		

Finally, the `Speech` class has one last method that is not really part of any of the three general groups above. The `speech_to_dict` method is an auxiliary method that transform a `Speech` object into a Python dictionary. This method is tailored made to be used during the visualization process (see Visualization section).

2.2.2 The `Corpus` Class

The `Speech` object is a model for every speech. However, we may may want to analyze several speeches at the same time: we might want to know what are the most frequent content bigrams, not only on a given speech, but on all the speeches in a decade, for instance. Therefore, we need a way of drawing together several speeches, so that we can perform the same types of queries and analysis on a set of speeches (not just on a single one). This is what the `Corpus` class if for.

The `Corpus` object is a collection of `Speeches`. The `Corpus` object is highly customizable: we can create a `Corpus` object that contains all the speeches in the `speeches` folder (all speeches from 1975 to 2017), create a corpus that only contains the speeches of a given period of time, or even build an object that contains only one speech or a selected choice of speeches. There are several methods to call the `Corpus` constructor depending on which of these options we want to build. Every `Corpus` object consists of:

- A `PlaintextCorpusReader` object from the `NLTK` library.
- A dictionary `Speeches`, whose keys are the years contained in our `Corpus` and the value for every year is the `Speech` object of the speech for that year. This means that every speech on the corpus has its own `Speech` object. This enables us to do queries only on a given year in the corpus.
- A list `years` with the years covered by our `Corpus` object.
- A `Speech` object that takes all the single speeches texts in our selection and transform them into one big `Speech` object. This enables to treat the set of individual speeches as just one long speech. As a result, we are able to apply the same queries that we did for a `Speech` object on various speeches at the same time.

– A `Complementary Corpus`. This object is only created when the `Corpus` object does not contain the entire set of speeches. The `Complementary Corpus` is a type of `Corpus` that contains all the speeches that are not in the current selection of our `Corpus`. This field enables to know what words, bigrams, trigrams etc that appear in our `Corpus` are unique to that subset of the corpus, i.e. they do not appear in the `Complementary Corpus`.

– The `unique_words` field contains all the words that are in our current `Corpus` that are unique to the speeches in the `Corpus`, that is, words that are not found in the `Complementary Corpus`.

Both `Complementary Corpus` and `unique_words` are null when the `Corpus` contains all the speeches between 1975-2017. In other words, `Complementary Corpus` and `unique_words` will be not null only if our current `Corpus` contains a subset of all the speeches (that is, if there is complementary corpus that can be built).

The most important method for the `Corpus` object is the `radiography()` method. Given a `Corpus` object, `radiography()` returns some relevant lexical information about that `Corpus`. If the `Corpus` was built with all the speeches, the information returned by the `radiography` method will concern the entire period of time from 1975 to 2017. If only a subset of speeches (or even just one) was used to build the `Corpus` object, then the returned information will concern only those speeches.

The `radiography()` method displays the following information:

– Years included in the corpus.

– Total number of speeches in the corpus.

– Total number of words.

– Words per speech ratio.

– Frequency distribution for the words in the corpus.

– Frequency distribution for content words.

– Hapaxes contained in the corpus.

– Words that are unique to the corpus, that is, words that appear exclusively on the created corpus, but **not** in the `Complementary Corpus` (if there is no `Complementary Corpus` because the created `Corpus` contains all the speeches a message is displayed).

– Frequency distribution for content bigrams.

– Frequency distribution for content trigrams.

2.3 Corpus analysis examples

The `Corpus` class allows performing a corpus analysis on the entire set of speeches or only on a set of years.

The default case is calling the constructor with the name of the speech files that we want to analyze:

```
myCorpus = Corpus(["1975.txt", "1976.txt", "1977.txt"])
```

The previous call will create a `Corpus` object that contains the speeches for 1975, 1976 and 1977. The years do not need to be consecutive:

```
myCorpus = Corpus(["1985.txt", "1995.txt", "2005.txt"])
```

If we want to analyze the entire corpus of speeches, the list can be left blank, as the constructor will understand that all the speeches in the speeches folder need are to be included in the object:

```
myCorpus = Corpus([]) # All speeches will be analyzed
```

If we want to make the corpus with a subset of consecutive years, we could just call the corpus listing the years to be analyzed:

```
myCorpus = Corpus(["1975.txt", "1976.txt", "1977.txt", "1978.txt",
"1979.txt", "1980.txt", "1981.txt"]) # Speeches from 1975 until 1981
will be analyzed
```

However, to make it simpler, the following method is also available if we want to create a corpus for a period of consecutive years:

```
myCorpus = create_corpus(1975, 1981) # Speeches from 1975 until 1981
will be analyzed
```

Once the corpus with the desired years is created, we can call:

- Any of the methods from the `Speech` class on the `Speech` field of our corpus.
- Any of the methods from the `Speech` class but applying it only to one of the years within the `Speeches` field of our corpus.
- The `radiography` method to get a general overview of the lexical information of our `Corpus`.

Some code examples with the method calls mentioned above:

```
   >>> myCorpus.speech.concordance("orgullo")

   Displaying 20 of 20 matches:
nuestra empresa con dignidad , con orgullo y , sobre todo
nuestra patria , y hemos de sentir orgullo de serlo , lo
idad .  No abandonemos jamás nuestro orgullo español , nues
encia de nuestra unión y de nuestro orgullo nacional .  Yo
hortaba a que sintiéramos juntos el orgullo de ser español
e sentido podemos mirarnos con sano orgullo y compartir nu
agamos todo lo posible para que ese orgullo esté justifica
este aspecto una cura del excesivo orgullo y la seguridad
idades que esperan , nos llenará de orgullo y satisfacción
ía , con serenidad y firmeza , y el orgullo del patrimonio
quede limitada por sentimientos de orgullo o de amor prop
nos dan un ejemplo que nos llena de orgullo .  También , de
cientes conmemoraciones reavivan el orgullo colectivo de l
voluntad de servicio y mi profundo orgullo por haber podi
paz en Oriente Medio .  Me llena de orgullo la ilusión col
Autónomas .  Sigamos adelante .  Con orgullo y autoestima ,
o a conocer a fondo y a admirar con orgullo .  Llevamos var
e es mucho y lo debemos valorar con orgullo .  Aunque tambi
futuro .  La clave para recuperar el orgullo de nuestra con
ndo , una emoción sincera , y es un orgullo muy legítimo .


   >>> myCorpus.speeches[1980]concordance("orgullo")

   hortaba a que sintiéramos juntos el orgullo de ser españoles

   >>> myCorpus.radiography()

   Lexical data for period from 1975 to 2017
43 total speeches
63035 total words
1465.9302325581396 words per speech
Frequency distribution:
[(',', 3771), ('de', 3534), ('y', 2796), ('.', 2173), ('que', 2150),
('la', 2095), ('en', 1616)...
Content words frequency distribution:
[('españa', 326), ('españoles', 207), ('año', 161), ('sociedad', 145),
('paz', 138), ('debemos', 137), ('futuro', 132)...
Hapaxes:
['acentúan', '1976', 'finaliza', 'sello', 'Generalísimo', 'testamento',
'dirigido', 'documento'...
Unique words frequency distribution:
The corpus contains all speeches, so no comparison can be made
```

```
Most frequent content bigrams:
[(('unión', 'europea'), 25), (('mejores', 'deseos'), 22), (('sociedad',
'española'), 22), (('crisis', 'económica'), 19)...
Most frequent content trigrams:
[(('hombres', 'y', 'mujeres'), 16), (('cuerpos', 'de', 'seguridad'),
12), (('económicos', 'y', 'sociales'), 12)...
```

The main method in the `corpus.py` file contains the following calls as code examples:

- A `Corpus` object is built that contains all the speeches. The `radiography()` method is called on that object.
- A `Corpus` object is built for every historical period of time (transition, socialism, economic bubble and economic recession) and the `radiography()` method is called on every object.

The response to all these calls can be checked on the file `output.txt`. The analysis (and especially the content bigrams and trigrams lists) shows the changes that have taken place in the political topics during the different time periods in Spain.

3 Visualization

The file `visualize.py` is a Python script that interacts with our `Corpus` class and the Python library `scattertext` in order to create visualizations of our corpus. `Scattertex` is a library that splits a given corpus of text into two subcorpus and produces interactive visualizations that display the lexical differences between the two subcorpus.

Our speech corpus can be divided into several subcorpus based on different distinctive features:

- Based on the king delivering the speech, we can split the corpus into Juan Carlos's speeches and Felipe's speeches.
- Based on the year when the speech was delivered, we can split the corpus into a first half with older speeches (before 1996) and a second half with more recent speeches (after 1996).
- Based on the historical moment when the speech was delivered, we can split the corpus into four periods: transition period from the dictatorship to the democracy (1975-1981), socialist period (1982-1995), economic bubble (1996-2007) and economic recession (2008-2017).

The `visualize.py` script splits our `Corpus` containing the entire collection of speeches into these different groups and feeds them to the `scattertext` library. The `scattertext` library produces `HTML` interactive visualizations that portray the main lexical differences between the two subcorpus according to TF-IDF frequencies and displays them on an Cartesian axis.

For example, we can provide `scattertext` with two subcorpus from our speeches corpus, one subcorpus with all Juan Carlos's speeches and the other subcorpus with Felipe's speeches. `Scattertext` will measure the frequency information in terms of the TF-IDF values and display on a graph the differences in word usage. The y axis will display the TF-IDF values for one of the subcorpus, the x axis will display the TF-IDF values for the other. Consequently, words that are displayed with a high value for both x and y are words that have high TF-IDF values in both subcorpus. On the other hand, words that are displayed with a low value for x and a high value for y are words that have low TF-IDF values in one subcorpus but a high value in the second subcorpus. Words that are near the diagonal are words that have similar TF-IDF values in both subcorpus.

We have applied `scattertext` with several subcorpus fragmentation of our corpus: we have splitted it according to kings (Juan Carlos vs Felipe), historical periods (transition speeches vs other periods, socialist speeches vs other periods, economic bubble vs other periods and economic recession vs other periods) and time halves (speeches before 1996 vs speeches after 1996). As a result, six `HTML` visualization have been produced (they can be found in the `visualization` folder from the project and they can be opened with any regular browser). The `HTML` visualization also includes the possibility of localizing a certain word on the graph and displaying its concordances and frequency values (note that not all words on the corpus make it to the visualization, only the most significant words are displayed).

These visualizations are a good portray of how the Spanish society has changed during the years. Words like *dios* ("god") or *patria* ("homeland") that were frequent a few decades ago are not longer used in the king's speech. The emergence of other words related to the political debate (like *Europa* during the oughties or *Cataluna* in recent years) are a good lexical thermometer of the political state of affairs in Spain during the last 43 years.

4 Markovian speech generator

Finally, we are going to use the compiled corpus as training material for an automatic generator of royal speeches. To do so, we will use the Python library `markovify`. `Markovify` takes real sentences as training input and produces new sentences automatically. These new sentences are produced following a markovian chain created using the training sentences as model and therefore bear a stylistic resemblance to the training sentences.

We will use the real king's speeches as training sentences. These real sentences will be fed to the library functions and as a result we will get randomly generated sentences that, although ungrammatical in most cases, are extremely similar to the royal speeches style (and quite hilarious for those that understand Spanish and are familiar with the king's speeches).

However, a speech is not just a concatenation of sentences. Any discourse has an internal structure: an introductory part with greetings, a body were ideas are developed and a closing section with conclusions and a farewell. A random concatenation of automatic sentences would not emulate the actual internal structure of a speech. Therefore, in order to improve the result of our automatic speech generator, we have divided our training sentences into three groups: sentences that appear in the first two paragraphs of the king's speeches; sentences that appear in the last four paragraphs of the king's speeches and sentences that appear in the central part of the speeches (which includes all sentences that were not included in the previous two groups). This way, we will not only get random sentences, but we will get automatic sentences that belong to the introductory part of the discourse (because they have been trained on the opening sentences of the speeches), a set of body sentences (trained on the sentences from the body of all discourses) and a set of closing and farewell sentences (trained on the closing section of speeches).

In addition, the automatic speeches were generated alternating between long sentences and short sentences (140 characters or less) to provide the automatic speeches with some oral rhythm.

The execution of the the `markov.py` file produces a randomly markovian-generated speech that is saved as a `txt` file in the `markovian_speeches` folder. Here is an example of a randomly generated speech:

Y lo hago desde esta Casa, que es una gran Nación. Las navidades convocan más que otras fechas a las tensiones y las jóvenes generaciones en este tradicional mensaje con el terrorismo sigue siendo un objetivo prioritario e inaplazable. Llegamos al final de un año me dirigí a vosotros en estas fechas tan entrañables para todos. Con motivo del Treinta Aniversario de mi proclamación como Rey y en el marco de las cosas que han escogido a España como hogar y contribuyen a nuestro desarrollo.

Que venzamos juntos las dificultades, pues muchas veces a prueba. El pasado mes de junio pasado, España se hunde en las responsabilidades colectivas. Somos una parte esencial de nuestra Constitución va a derrotar ni ahora ni nunca. Una acción exterior ampliamente consensuada. A las Fuerzas y Cuerpos de Seguridad, haciendo efectiva nuestra entrega a la sociedad española. En estas fechas, no podemos eludir y cuestiones nuevas que no es una muestra esencial de nuestra realidad nacional, afectada, como es el del entendimiento, la rivalidad por la solidaridad. Una España que es la garantía última de nuestros derechos

y libertades de todos para orientarnos y perfeccionarnos. Estrechamente unidos en lo más mínimo la estatura histórica a que el Estado de Derecho, se ensancha el marco de convivencia fraternal entre los hombres. Esta es una verdad incuestionable que debemos afrontar con responsabilidad, con ilusión y el realismo imprescindibles para el desánimo. Hoy vivimos en un proceso de paz y la aspiración a una comunidad que sepa mantenerse unida, sin retroceder ni perder posiciones que ha generado la crisis económica y social como regional. Ciudadanos, instituciones y hacia la modernidad, mientras esto no ocurra. La unidad, necesaria para mantener el peso relativo que merece reconocimiento y despliegue de la voluntad de paz, de unidad y continuidad de España; un mensaje abierto y competitivo.

Que ese progreso alcance a cuantos extranjeros residen en nuestra tierra. Felicidades, tanto en nombre propio y de esperanza para transitar hacia esa frontera con fe y creamos en nuestro tiempo el crecimiento y las relaciones en todos los españoles, con especial afecto a los que han padecido tribulaciones a lo largo de este año, reafirmo con la que el nuevo año esté para todos los extranjeros que residís con nosotros, todo mi afecto más sincero y mi profundo orgullo por haber podido contribuir al esfuerzo de España, así como a los pueblos árabes y a vuestras familias, una muy feliz Navidad. La misma felicidad que pido a Dios que extienda su protección sobre todos nosotros un tiempo de entendimiento y prosperidad. Y, por último, en un abrazo, a amar a España y a vuestras familias, una muy feliz Navidad. Tiempos, en fin, en los países árabes. Este año, nuestra Familia ha tenido la gran satisfacción de anunciar el compromiso colectivo que día a día, con esfuerzo y entrega, pero también ofrece grandes oportunidades de acceso a los extranjeros que comparten con nosotros ! En estas horas de paz, en circunstancias particularmente difíciles, y a todos mis mejores deseos para estas Fiestas, y de todos los españoles, os deseo unas navidades muy felices, y un nuevo año lleno de afecto.

5 Conclusions and future work

The current state of the project offers a plain text compilation of all the Christmas speeches. The `Corpus` class and the `Speech` class enable the possibility of querying the corpus and retrieving lexical data from it (either on a particular speech, on a subset of speeches, on a given period of time or on the entire corpus of speeches as a whole).

We have also produced several interactive visualizations based on TF-IDF measures in order to compare what keywords have changed over time and what terms remain stable, and to contrast the lexical frequency differences between the two kings.

Finally, we have trained and produced our own random royal speech generator based on Markov model.

The main problem encountered while creating the linguistic analysis is the lack of reliable lemmatizers for Spanish language. Although the current `Speech` class produces a tagged and lemmatized version of the speech, this information is not currently being used in any of the lexical analysis or visualizations. The need for a lemmatized version of the text becomes particularly obvious on the graphic visualizations, where it would be interesting to have flexionated words such as *españoles*, *españolas*, *español*, *española* (all of them flexionated forms of the adjective *español*, "Spanish") reduced to its lemma. The reason is that, although several lemmatizers were tried, none of them produced results that were reliable enough to build the analysis. The current lemmatizer that produces the lemmatized version of the text as field of the `Speech` object is the one from the `spaCy` library. This lemmatizer, however, produces highly unreliable lemmas on even the most frequent words (for example, the ubiquitous preposition *para*, "for", is considered a conjugated form of the verb *parir*, which means to "give birth to"). Other options were explored, like the Python library `pattern`, but due to incompatibilities between Python2 and Python3 versions it was finally discarded.

A reliable lemmatized version of the text could provide us with a better insight of the texts and a recalculation of the frequency values.