4 Models

In this chapter, we present two alternative models applied to the Quotation Extraction task. We create one model using the ETL algorithm and one model using the Structured Perceptron algorithm. In our model for ETL we adopt a token classification approach. We decompose our task into subtasks and provide a baseline system for each subtask. In our model for Structured Learning we describe a prediction problem used to solve the Quotation Extraction task and we define its respective input-output mapping. For both models, we present the feature set used to solve the task.

4.1 Entropy Guided Transformation Learning

In our model for ETL, we adopt a token classification approach. We decompose the Quotation Extraction task into two subtasks: *quotation identification* and *association between quotation and author*. We further decompose the quotation identification subtask into three elementary identification subtasks: *quote beginning*, *quote end* and *quote bounds*. The latter corresponds to the match made between a quote beginning and a quote end. Since ETL is an error-driven learning algorithm, we must provide a baseline system for each subtask.

4.1.1 Quote Beginning Identification

We model this subtask for the ETL algorithm. We use POS and NE features to solve this subtask.

In the preprocessing step, we use a list of verbs of speech, e.g. *disse*, *falou*, *comentou*[1], and change their POS from V to $VSAY$. Moreover, we create three derived features. The first one is the *Bounded Chunk*, in which we assign a *1* to the three quotation marks ‘ “ - and also to all the tokens between them, whenever there are more than three tokens between the quotation marks. Otherwise, we assign a *0* to the token. The number of tokens between quotation

[1]said, spoke, commented

marks is an empirically set threshold. Table 4.1 shows an example of this feature.

Table 4.1: Derived feature Bounded Chunk example

Word	**’**	**Quem**	**manda**	**aqui**	**sou**	**eu**	**’**	**disse**	**Bernardinho**	**.**
BC	1	1	1	1	1	1	1	0	0	0

The second derived feature is the *Verb of Speech Neighborhood.* We assign a *1* to each verb of speech and also to its four closest tokens. Otherwise, we assign a *0* to the token. The neighborhood size is also an empirically set threshold. Table 4.2 shows an example of this feature.

Table 4.2: Derived feature Verb of Speech Neighbourhood example

Word	**’**	**Quem**	**manda**	**aqui**	**sou**	**eu**	**’**	**disse**	**Bernardinho**	**.**
POS	’	PRO-KS	V	ADV	V	PPES	’	VSAY	NPROP	.
SVN	0	0	0	0	0	1	1	1	1	1

The last derived feature is the *First Letter Upper Case.* We assign a *1* to a token, if it is a word and its first letter is in upper case. Otherwise, we assign a *0* to the token. Table 4.3 shows an example of this feature.

Table 4.3: Derived feature First Letter Upper Case example

Word	**’**	**Quem**	**manda**	**aqui**	**sou**	**eu**	**’**	**disse**	**Bernardinho**	**.**
FLUC	0	1	0	0	0	0	0	0	1	0

We use a baseline classifier with two classification patterns, both of them based on regular expressions. The baseline assigns a *S* tag to tokens after quotation marks, except when the quotation mark is preceded by a number. In Figure 4.1, we illustrate this kind of exception. Below we show the respective regular expression.

[ˆ**number**] [’ ” -] **.**

In Figure 4.2, we present an application example of this rule. Each *S* token is tagged with a *Start* subscript.

1 - Estados Unidos: 6m05s34
2 - Holanda: 6m07s22
3 - Romênia: 6m07s25

Figure 4.1: First rule exception example

> - É$_{Start}$ a primeira vez em o planeta que um time brasileiro paga sozinho quase 10 milhões de euros e não tem o atleta - lamenta$_{Start}$ o mandatário alvinegro.

Figure 4.2: First rule application example for the quote beginning identification subtask.

The second rule is to assign a S to tokens after a period or question mark followed by a person or organization and a colon, provided the token is not a person or organization. This last restriction treats cases such as soccer team squad lists. Below we show the respective regular expression.

[\. \?] [**Person Organization**] **:** [ˆ**Person Organization**]

In Figure 4.3, we present an application example of this rule. Each S token is tagged with a *Start* subscript.

> (...) Copa do Mundo de 1958. GLOBOESPORTE.COM: O$_{Start}$ Garrincha foi reprovado em o exame psicológico, mas acabou sendo um de os destaques da Copa de 1958.

Figure 4.3: Second rule application example for the quote beginning identification subtask.

4.1.2 Quote End Identification

We model this subtask using a rule-based approach, since it has a good performance. We use the features word, POS, NE and quote start, generated in the previous subtask, to solve this subtask.

We use a baseline system with three classification patterns, all of which are based on regular expressions. Starting from a single quote followed by a token classified as quote start, we read all the tokens until we reach another single quote or the feed end. We assign a E to the token before the single quote or the feed end. It is similar for double quotes. We show the two respective regular expressions below.

’ QuoteStart .* [**’ FeedEnd**]
” QuoteStart .* [**” FeedEnd**]

In Figure 4.4, we present an application example of this rule. Each S token is tagged with a *Start* subscript. Similarly, each E token is tagged with an *End* subscript.

Roth elogia Roger e diz que armador
tem que 'ir$_{Start}$ para o sacrifício$_{End}$'.

Figure 4.4: First rule application example for the quote end identification subtask.

The second rule is as follows: starting from a dash followed by a token classified as a quote start, we read all the tokens until we reach another dash or the feed end. The last dash cannot be preceded by the word *ex* or followed by the POS *PPES*, which means a personal pronoun. Those restrictions treat cases where dashes are not used as quotation delimiters, e.g. prefixes and enclisis like *ex-jogador* and *procurá-lo* [2]. If those restrictions are satisfied, we assign a *E* to the token before the dash or the feed end. We show the respective regular expression below.

- QuoteStart .* [[ˆ**ex**]-[ˆ**PPES**] **FeedEnd**]

In Figure 4.5, we present an application example of this rule. Each *S* token is tagged with a *Start* subscript. Similarly, each *E* token is tagged with an *End* subscript.

- Se$_{Start}$ ele está desgastado, vamos ter que conversar.
Talvez seja melhor nem jogar. Mas acho que não foi
isso que ele quis dizer$_{End}$ - sorri Roth.

Figure 4.5: Second rule application example for the quote end identification subtask.

The third rule is: starting from a person or organization followed by a colon and a token classified as a quote start, we read all the tokens until we reach a period or question mark followed by a person or organization and a colon, or the feed end. We assign a *E* to the period or question mark, or to the token before the feed end. We show the respective regular expression below. *Organization* is represented by *Org*.

[**Person Org**] **: QuoteStart .*** [\. \?] [([**Person Org**] **:**) **FeedEnd**]

In Figure 4.6, we present an application example of this rule. Each *S* token is tagged with a *Start* subscript. Similarly, each *E* token is tagged with an *End* subscript.

[2] ex-player, look for him

GLOBOESPORTE.COM: Qual$_{Start}$ a importância da conquista da Copa do Mundo de 1958?$_{End}$
JOÃO HAVELANGE: (...)

Figure 4.6: Third rule application example for the quote end identification subtask.

4.1.3 Quote Bounds

We model this subtask using a rule-based approach, since it has a good performance. We use the features quote start and quote end, both generated in the previous subtasks, to solve this subtask.

Table 4.4: Rule application example for the quote bounds subtask.

Word	**Carlinhos**	**Bala**	**:**	**'**	**Enílton**	**fez**	**o**	**gol**	**de**	**o**	**título**	**'**
QS	_	_	_	_	S	_	_	_	_	_	_	_
QE	_	_	_	_	_	_	_	_	_	_	E	_
Quote	_	_	_	_	q	q	q	q	q	q	q	_

The baseline system assigns a *q* tag to tokens from the quote start to the quote end. It assigns a *O* to the remaining tokens. In Table 4.4, we present an application example of this rule.

4.1.4 Association Between Quotation and Author

We model this subtask for the ETL algorithm. We use features POS, quote and coreference to solve this subtask.

Table 4.5: Derived feature Coreference Indicator example

Word	**Caio**	**Junior**	**:**	**'**	**O**	**Juan**	**foi**	**o**	**melhor**	**de**	**o**	**jogo**	**'**
Coref	ref00	ref00	_	_	_	ref01	_	_	_	_	_	_	_
CI	ref	ref	_	_	_	ref	_	_	_	_	_	_	_

In the preprocessing step, we use the list of verbs of speech and change their POS to *VSAY*. Moreover, we create one derived feature which is the *Coreference Indicator*, in which we assign a *ref* to the token, if its coreference feature value is different from underline. Otherwise, we assign an underline to the token. Table 4.5 shows an example of this feature.

In addition, we concatenate tokens, resulting in one new token per quote, and one new token per coreference. In Table 4.6, we present an example of a

Table 4.6: Sentence before concatenation

Word	**Caio**	**Junior**	**:**	**'**	**O**	**Juan**	**foi**	**o**	**melhor**	**de**	**o**	**jogo**	**'**
Quote	_	_	_	_	q	q	q	q	q	q	q	q	_
Coref	ref00	ref00	_	_	_	ref01	_	_	_	_	_	_	_

Table 4.7: Sentence after concatenation

Word	**Caio/Junior**	**:**	**'**	**O/Juan/foi/o/melhor/de/o/jogo**	**'**
Quote	_	_	_	q	_
Coref	ref00	_	_	_	_

sentence before concatenation. We show the same sentence in Table 4.7 after concatenation.

Table 4.8: Sentence before elimination

Word	**Cuevas**	**chega**	**e**	**avisa**	**:**	**'**	**Não/me/considero/ salvador/de/a/pátria**	**'**
POS	NPROP	V	KC	VSAY	:	'	O	'
Quote	_	_	_	_	_	_	q	_
Coref	ref00	_	_	_	_	_	_	_

Furthermore, we eliminate tokens with POS different from a period and *VSAY*, quote equal to *O* and coreference equal to *O*. In Table 4.6, we present an example of a sentence before elimination. We show the same sentence in Table 4.7 after elimination.

Table 4.9: Sentence after elimination

Word	**Cuevas**	**avisa**	**Não/me/considero/ salvador/de/a/pátria**
POS	NPROP	VSAY	O
Quote	_	_	q
Coref	ref00	_	_

We use a baseline classifier with two classification patterns, both of them based on regular expressions. The first rule is: the baseline assigns the *r1+* tag to the quote, if it is followed by a saying verb and a coreference. We show the respective regular expression below.

Quote SayingVerb Coreference

In Figure 4.7, we present an application example of this rule. We show the original sentence to facilitate reading. The quote is in *italic*, the verb of speech is underlined and the coreference is in **bold**. The baseline system assigns *r1+* to the quote.

> - *Não vamos aceitar. Guiñazu tem contrato de mais dois anos e meio, é um ídolo, um jogador maravilhoso. Este não é o preço de ele* - diz **Piffero**.

Figure 4.7: First rule application example for the association between quotation and author subtask.

The second rule is as follows: the baseline assigns the tag *rd-* to the quote, if in the previous sentence there is a coreference. We link the quote to the first coreference in the previous sentence. The number of coreferences from the quote to the coreference is expressed by *d*. We show the respective regular expression below.

[\.]? .* Coreference .* \. .* Quote

In Figure 4.8, we present an application example of this rule. We show the original sentence to facilitate reading. The quote is in *italic* and the coreferences are in **bold**. The baseline system assigns *r2-* to the quote.

> (...) afirma. **Caio Júnior** diz que a ausência é prejudicial para **o jogador**, que abre espaço para outros atletas. - *Ele dá o espaço para outro jogador, futebol é assim. O Éder entrou bem, o Maxi entrou bem* - diz.

Figure 4.8: Second rule application example for the association between quotation and author subtask.

If those two classification patterns are not applicable, the baseline assigns the *r1+* tag to the quote, since this is the most frequent classification.

In this section, we presented our model to ETL, the baseline system used by ETL and the decomposition of the Quotation Extraction task. In the next section, we describe our model to Structured Perceptron.

4.2 Structured Perceptron

We need to define a prediction problem to solve the Quotation Extraction task. We also need to define a mapping from an input $\boldsymbol{x}$ to an output $\boldsymbol{y}$. Each input-output pair belongs to a piece of news and corresponds to all quotations identified in that piece of news. Moreover, we need to define a feature set to be used by the Structured Perceptron algorithm. In the next sections, we outline the prediction problem used to solve the task, its respective input-output mapping and the feature set.

4.2.1 Prediction Problem

We model our task as a well-known optimization problem called weighted interval scheduling (WIS). In the WIS problem, we have a set of n tasks which have a start time s_i, an end time e_i and a weight w_i. We want to find the maximum-weight subset of nonoverlapping tasks. That problem is efficiently solved by dynamic programming.

```
 1. sort tasks by increasing end times.
 2. compute $p_i$ for i from 1 to n.
 3. $M_0 \leftarrow 0$; $S_0 \leftarrow \{\}$
 4. for i from 1 to n
 5.    if $w_i + M_{p_i} \geq M_{i-1}$
 6.       $M_i \leftarrow w_i + M_{p_i}$; $S_i \leftarrow \{i\} \cup S_{p_i}$
 7.    else
 8.       $M_i \leftarrow M_{i-1}$; $S_i \leftarrow S_{i-1}$
 9.    endif
10. endfor
```

Figure 4.9: Linear time algorithm for WIS

We present linear time algorithm for WIS in Figure 4.9. We define M_i as the maximum weight of any set of compatible tasks, all of which end by e_i. Moreover, we define S_i as the tasks involved in schedule M_i. We also define p_i as the task k with the largest index less than i and $e_k < s_i$. If there is no task that satisfies that condition, $p_i = 0$.

The WIS solution is progressively built. At each iteration i of n, we determine if task i will be part of the solution, or the task $i - 1$. By the end of the iterations, we have the maximum weight of all the compatible tasks in M_n. We also have the tasks involved in the solution in S_n.

4.2.2 Feature Set

We use thirteen basic features for each quotation-coreference combination (q_i, c_j). We present an illustrative example to exemplify the features in Figure 4.10. Coreferences are in **bold**, quotations are in *italic* and verbs of speech are underlined. Each coreference has an integer subscript that identifies the candidate. Similarly, each quotation has an integer subscript that identifies the candidate. In total, there are eight coreference candidates and two quotation candidates.

> **Nélio Machado**$_1$, que defende **Daniel Dantas**$_2$, considerou '*estranha*'$_1$ a acusação de que **Dantas**$_3$ teria cogitado subornar **o juiz**$_4$. '*Isso é o fim da picada. Completamente sem fundamento e bem no dia em que o* ***Daniel***$_5$ *vai prestar depoimento. Estou inclinado a pedir suspeição* ***dele***$_6$ *[**Fausto de Sanctis**$_7$]. Acho muito estranho, tem conteúdo de mais armação do que qualquer outra coisa*'$_2$ <u>disse</u> **ele**$_8$.

Figure 4.10: Example to illustrate the input features for Structured Perceptron

The first one is *Distance*, which contains the number of coreferences between q_i and c_j. In Figure 4.10, for the combination $(1, 2)$, we assign a *distance0*. For the combination $(1, 5)$, we assign a *distance2*.

The second one is *Direction*, which indicates whether c_j is on the left side or on the right side of q_i. In Figure 4.10, for the combination $(1, 1)$, we assign a *directionLeft*. For the combination $(1, 3)$, we assign a *directionRight*.

The third one is *Verb of Speech in Between*, which indicates whether there is a verb of speech between q_i and c_j, based on a list of verbs of speech. In Figure 4.10, for the combination $(2, 8)$, we assign a *verbOfSpeechInBetween*. For the combination $(1, 1)$, we do not assign any tag.

The fourth one is *Number of Verbs of Speech in Between*, which contains the number of verbs of speech between q_i and c_j. In Figure 4.10, for the combination $(2, 8)$, we assign a *numVerbsOfSpeechInBetween1*. For the combination $(1, 1)$, we assign a *numVerbsOfSpeechInBetween0*.

The fifth one is *Coreference in Between*, which indicates whether there is a coreference between q_i and c_j. In Figure 4.10, for the combination $(1, 1)$, we assign a *coreferenceInBetween*. For the combination $(1, 2)$, we do not assign any tag.

The sixth one is *Quote in Between*, which indicates whether there is a quote between q_i and c_j. In Figure 4.10, for the combination $(2, 1)$, we assign a *quoteInBetween*. For the combination $(2, 3)$, we do not assign any tag.

The seventh one is *BLS Choice*, which indicates whether the baseline system proposed in Section 4.1 selects the combination (q_i, c_j). Supposing the baseline system selects the combinations $(1, 2)$ and $(2, 8)$ in Figure 4.10, we assign a *blsChoice* tag only to those combinations.

The eighth one is *Coreference POS Window*, which contains the coreference POS tag and the POS tags of its nearest ten tokens. In Figure 4.10, for the combinations that contain the coreference 1, we assign the tags *corefPOSWin-5=None*, *corefPOSWin-4=None*, *corefPOSWin-3=None*, *corefPOSWin-2=None*, *corefPOSWin-1=None*, *corefPOSWin0=NPROP*, *corefPOSWin0=NPROP*, *corefPOSWin1=COMMA*, *corefPOSWin2=PRO-KS-REL*, *corefPOSWin3=V*, *corefPOSWin4=NPROP* and *corefPOSWin5=NPROP*.

The ninth one is *Quotation POS*, which contains the quotation POS tag. In Figure 4.10, for the combinations that contain the quotation 1, we assign a *QuotationPOS=ADJ*.

The tenth one is POS In Between, which contains the POS tags of the tokens between q_i and c_j. In Figure 4.10, for the combination $(1,2)$, we assign the tags *POSInBetween=COMMA* and *POSInBetween=V*.

The remaining ones are *Bounded Chunk*, *Verb of Speech Neighborhood* and *First Letter Upper Case*, presented in Section 4.1.

4.2.3 Input-Output Mapping

We define a mapping from an input $\boldsymbol{x}$ to an output $\boldsymbol{y}$. We model input $\boldsymbol{x}$ as a quotation candidate set. All quotation candidates are generated by the baseline system explained in Section 4.1. For each quotation, there is a coreference set, which are quotation author candidates. The coreference set has the eight nearest coreferences, two after the quotation and six before it, and a dummy corefence to be selected when the quotation is invalid. We choose this configuration based on the distribution of distances from quotations to their authors in our corpus, as presented in Figure 5.3.

> **Nélio Machado**$_1$, que defende **Daniel Dantas**$_2$, considerou '*estranha*'$_1$ a acusação de que **Dantas**$_3$ teria cogitado subornar **o juiz**$_4$. '*Isso é o fim da picada. Completamente sem fundamento e bem no dia em que o* ***Daniel***$_5$ *vai prestar depoimento. Estou inclinado a pedir suspeição* ***dele***$_6$ *[**Fausto de Sanctis**$_7$]. Acho muito estranho, tem conteúdo de mais armação do que qualquer outra coisa*'$_2$ <u>disse</u> **ele**$_8$.

Figure 4.11: Example to illustrate the construction of x

We present an example to illustrate the construction of x in Figure 4.11. Coreferences are in **bold** and quotations are in *italic*. Each coreference has an integer subscript that identifies the candidate. Similarly, each quotation has an integer subscript that identifies the candidate.

We represent x as

$$\begin{aligned}\boldsymbol{x} =&((11,11,\{distanceNone, directionNone,\\ &verbsOfSpeechInBetweenNone, numVerbsOfSpeechInBetweenNone,\\ &coreferenceInBetweenNone, quoteInBetweenNone\},\\ &\{distance1, directionLeft, numVerbsOfSpeechInBetween0,\\ &coreferenceInBetween\},\\ &\{distance0, directionLeft, numVerbsOfSpeechInBetween0,\\ &blsChoice\},\\ &\{distance0, directionRight, numVerbsOfSpeechInBetween0\},\\ &\{distance1, directionRight, numVerbsOfSpeechInBetween0,\\ &coreferenceInBetween\}),\\ &(25,72,\{distanceNone, directionNone,\\ &verbsOfSpeechInBetweenNone, numVerbsOfSpeechInBetweenNone,\\ &coreferenceInBetweenNone, quoteInBetweenNone,\\ &boundedChunk, verbOfSpeechNeighborhood, firstLetterUpperCase\},\\ &\{distance3, directionLeft, numVerbsOfSpeechInBetween0,\\ &coreferenceInBetween, quoteInBetween, boundedChunk,\\ &verbOfSpeechNeighborhood, firstLetterUpperCase\},\\ &\{distance2, directionLeft, numVerbsOfSpeechInBetween0,\\ &coreferenceInBetween, quoteInBetween, boundedChunk,\\ &verbOfSpeechNeighborhood, firstLetterUpperCase\},\\ &\{distance1, directionLeft, numVerbsOfSpeechInBetween0,\\ &coreferenceInBetween, boundedChunk, verbOfSpeechNeighborhood,\\ &firstLetterUpperCase\},\\ &\{distance0, directionLeft, numVerbsOfSpeechInBetween0,\\ &boundedChunk, verbOfSpeechNeighborhood, firstLetterUpperCase\},\\ &\{distance0, directionRight, verbOfSpeechInBetween,\\ &numVerbsOfSpeechInBetween1, blsChoice, boundedChunk,\\ &verbOfSpeechNeighborhood, firstLetterUpperCase\}))\end{aligned}$$

where each tuple corresponds to a quotation with its coreference candidates; each coreference is decomposed by its basic features and is in curly brackets; $(11,11)$ represents the start token and end token of the first quotation; $(25,72)$ represents the start token and end token of the second quotation; the coreference candidates of the first quotation are the dummy one, 1, 2, 3 and 4; and the coreference candidates of the second quotation are the dummy one, 1, 2, 3, 4 and 8. We ommit the features *Coreference POS Window*, *Quotation POS* and *POS In Between* in order to facilitate reading.

We formalize $\boldsymbol{x}$ as

$$\boldsymbol{x} = ((x_{qstart1}, x_{qend1}, x_{coref00}, \ldots, x_{coref0L_0}), \ldots,$$

$$(x_{qstartM}, x_{qendM}, x_{corefM0}, \ldots, x_{corefML_M}))$$

where $x_{qstarti}$ with $i \in \{1, \ldots, M\}$ is the i-th quotation start token, x_{qendi} is the i-th quotation end token, $x_{corefij}$ with $j \in \{0, \ldots, L_i\}$ is the combination of the j-th coreference of quotation i with quotation i, producing the basic features, and $corefi0$ is the i-th dummy coreference.

We model output $\boldsymbol{y}$ as a vector of author indexes.

$$\boldsymbol{y} = (y_1, \ldots, y_M)$$

where $y_i \in \{0,\ \ldots,\ L_i\}$ for $i = 1, \ldots, M$.

We define a loss function that penalizes each mispredicted quotation.

$$\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{\imath=\mathbf{0}}^{M} 1_{[y_i \neq ypred_i]}$$

We define the feature vector $\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y})$ and its decomposition along $\boldsymbol{y}$ as

$$\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\imath=\mathbf{0}}^{M} \boldsymbol{\Phi}(\boldsymbol{x}, y_i)$$

where each $\boldsymbol{\Phi}(\boldsymbol{x}, y_i)$ is composed by F features and defined as

$$\boldsymbol{\Phi}(\boldsymbol{x}, y_i) = (\phi_1(\boldsymbol{x}, y_i), \ldots, \phi_F(\boldsymbol{x}, y_i))$$

We combine each quotation candidate $(x_{qstarti}, x_{qendi})$, with $i \in \{1, \ldots, M\}$, with its coreference candidates $x_{corefi0}, \ldots, x_{corefiL_i}$ 1 by 1 and we call the combined elements *tasks*. Each task has start time $x_{qstarti}$ and end time x_{qendi}. Furthermore, we assign a weight to each task, which is the sum of features associated to the task. We want to find the maximum weight subset of non-overlapping tasks. This can be seen as the well studied weighed interval scheduling problem which can be efficiently solved by the algorithm presented in Section 4.2.1.