Named entity type classification: Using local and context features.

Sergio Duarte

1. Introduction

Named entity classification is the recognition and identification of linguistics elements that can be cataloged in several categories as names of organizations, persons, locations, time expressions, among others [1]. This classification represents an important subtask of more complex information extraction and linguistic applications, given that named entities represents the main content of a document [2].

Several rule based approaches have been used to work out this problem. Usually in these approaches finite state patters are built using specific linguistic information and word patters [3]. However, recently machine learning approaches have became more attractive given that such methods overcome many of the difficulties intrinsic in based rules approaches, as their adaptability to different application contexts and cost of maintenance [2].

Thus in the present work two different machine learning approaches are contrasted. The first method applied is Classification tree learning. This method was chosen because it is easier to understand and to interpret the results intrinsic in the algorithm, which is very useful since the main goal of this work is to analyze the impact of different features in the classification task. The second method is Bayesian learning and particularly naïve Bayes classifier. This method is studied because it has been used in a variety of practical linguistic problems. Naturally the aim of this work is to analyze which method is more convenient given a particular scenario and to study how the features behave for both approaches.

The predefined named entity types are taken from the hierarchy proposed by Magda Ševčíková [4].

2. Data Description

The annotated data used for the experiments were extracted from the Prague Dependency Treebank[1] which contains a subset of the Czech National corpus. Three different sets of data were used: training data, development data and test data. Table 1 summarizes the length of each dataset. For the training date 6109 samples were considered, for the test data 803 samples and for the development data 782, which correspond to the number of named entities in the data.

Table 1: Datasets length and number of named entities in each file

Data	*Sentences*	*Words*	*Named entities*
Training	1.608	41.710	6.109
Development	201	4.915	782
Test	201	5.296	803

The information provided in each data includes the lemma, form and morphological tag for each word. The information is shown in Table 2. The word form refers to the actual word that is found in the text, for example in Table 2 is shown the word "Krasnou" which is the Czech word for "*beautiful*" inflated into the fourth case (accusative). The lemma is a common representation for the set of all the forms that have the same meaning. Following the example, the lemma for the word form "Krasnou" is *krásný*, which is the word form without inflections. It is important to mention that the lemma was not considered in this study given that its selection or codification is somehow dependent of the data provider, for this reason its use could be not reproducible in studies using data from different sources.

Additionally, datasets provide morphological information of the words. The Prague Dependency Treebank use positional tags with 15 fields, which are summarize in Table 3. For the present study only three fields were considered: part of speech, gender and case. The fields 6, 7, 13, 14, 15 are undefined in 99% of the data, meanwhile the fields 8, 9, 10 and 12 are undefined in more than 90% of the data, thus these field were ignored. In addition the second field that correspond to *sub part of speech* was not considered given that the information provided is highly specific and it would difficult

[1] The data was provided by the lectures of the course Introduction to Machine Learning (2007)

the reproduction of the experiments performed in this work with data from different sources.

It is important to mention that it is not feasible to consider all the information provided by the tag given the enormous degrees of freedom that this introduce to the hypothesis representation, which is around 10^{10} solely for the morphological tag. In particular it was found that when it is considered more than 3 fields of the tag, the computational resources to process and train the models become very expensive.

Additionally, it is convenient to use only three fields since a bigger hypothesis representation requires more data, thus it is important to keep the number of features as small as possible given the size of the datasets (see Table 1).

Table 2: Information provided per word by the datasets

Element	*Description*	*Example*
Word form	Word as is found in the text	"Krásnou"
Lemma	Headword or form chosen by convention to represent the lexeme.	krásný
Morphological tag	Annotated morphological information of the word form	AAFS4----1A----

Table 3: Enumeration of the morphological tag elements

Position	**Description**
1	Part of speech
2	Sub part of speech
3	Gender
4	Number
5	Case
6	Possessive gender
7	Possessive number
8	Person
9	Tense
10	Grade
11	Negation
12	Voice
13	Reserve 1
14	Reserve 2
15	Variant

In the hierarchy classification provided in [4] there are two different levels of predefined named entities type. Table 4 illustrates the classes only considering the first level and Table 5 extends some of the classes given Table 4 to the second level.

Table 4 shows 11 types of named entities. 10 of these classes are suggested by Ševčíková hierarchy. The class "s" is annotated in the data and even if it does not belong to the hierarchy it was included in this study given than 7% of the named entity corresponds to this class.

Table 5 illustrates some named entity type classification considering two levels in the hierarchy. It is important to mention that each type in the first level can be expanded to more than 5 and up to 11 classes. For more details of these classifications refer to the reference [4].

Table 4: Named entity types for the first level in the hierarchy

Class	*Named entity*	*Example*
a	Number of addresses	356
c	Bibliographic items	3.2
g	Geographical names	Prague
i	Institutions	Cambridge
m	Media names	Gazette
n	Specific number usages	0-0
o	Artifact names	Christoslaus
p	Personal names	Santiago
q	Quantitative expressions	Fifth
t	Time expressions	2008
s	Abbreviations	SPP

Table 5: Some of the named entity types in the hierarchy when two levels are considered

Class	*Named entity*	*Example*
pd	Tittles	Wolf
g_	Underspecified	Mojunkumech
i_	Underspecified	IHS
or	Directive, norms	Listiny
tf	Feast	Silvestra
at	Phone numbers	57321068

In the current work only the first level is considered for the classification, which means that subtypes were merged into their main type. This approach is convenient given that many subtypes are not frequent in the data, producing a negative impact in learning algorithms. In Table 6 is shown some of the low frequencies named entity found in the data. In this table is possible to see that many entity types have less than 5 samples, which makes those types irrelevant to any machine learning algorithm. However after merging the second level into its first level, we obtain a significant better distribution of frequencies, as it is shown in Table 7.

Table 6 Named entity frequency without merging (two level) for some of the low frequency types

f	*Type*
1	Pd
2	g_
3	i_
3	Or
3	Tf
4	At

Table 7 Named entity frequency considering only the main level

f	type
0	n
0	c
23	a
79	m
297	o
385	s
451	i
731	t
1133	g
2632	p

Additionally all the words found in the data that does not belong to the hierarchy classification were ignored. Consequently the following types were ignored: segm, text,cap, lower,upper, ?, f

It is important to mention that for some samples there are two or more named entity type classification. Even if these samples reduce the accuracy of the machine learning method implemented, they were not ignored because it is considered that such situation represents better a real world scenario, where sometimes we don't have access to the test data to filter those kind of annotations, furthermore it can be relevant for specific applications to annotate a single sample with two different named entity types.

Also it was found that the percentage of these samples represents 12% of the training data and 16% of the testing data. Given these percentages, the elimination of these samples may bias the fairness of the experiments. Similarly it was not found strong criteria to choose a particular annotation for each one of the samples. For this reason it is convenient to analyze the methods including these samples.

3. Machine learning methods

3.1 Classification Trees

This method tries to find a set of rules to predict a dependent variable *Y* from *n* training samples X_i, which are called predictors. Each training sample has categorical and/or continuous values– measurable features for each observation- [5]. This prediction is performed building a decision tree which approximates the target function. A decision tree represents a set of if-then rules based on the feature values of the samples, this tree is built from the known data and it is used to predict the class for further samples with known features but unknown class. The Figure 1 shows an example of a Partitioning tree [6].

The leaves in the tree represent a partition of the space of all possible observations. It is important to note that in statistical problems the distribution of classes over *X* can overlap [6], for this reason the distribution of the class at each node can be represented as a probability distribution in which some criteria is used – for instance Bayes decision rule- to select the one that has the maximum probability or the one that minimize the error.

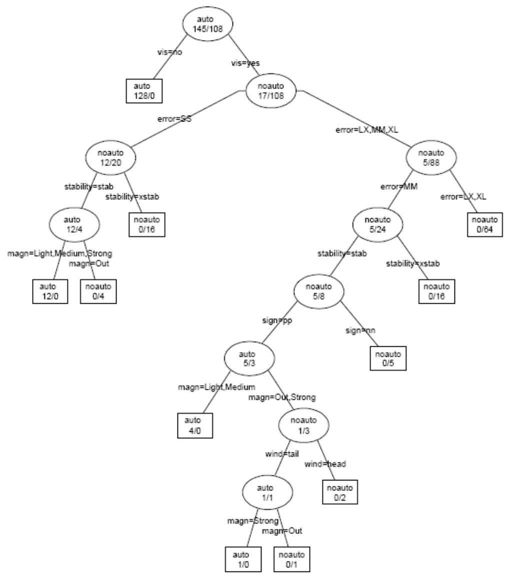

Figure 1: Partition tree example: The leaves show the proportion of class #1 over class #2 reached the given node [6]

The tree is built recursively starting with the root containing all the samples of the training data, then the algorithm choose the "best" feature to split the sample space, in this step all the features and its possible values are evaluated over some criteria function which measure the "purity" of the two children nodes (for binary splitting case that is the method used in this work), this function measure somehow the quality of the distribution in each node [7]. The *Gini index* was used as the splitting criteria [6]:

Equation 1: Gini index

$$\sum_{j!=k} p_{ij} p_{ik} = 1 - \sum_{k} p_{ik}^{2}$$

The value p_{ik} represents a multinomial distribution over the classes at the node *i*.

Once the best possible feature to split the current node is found, the algorithm assigns a class to the node. This task is done using the following formula [7]:

The partition and classification processes are done on every node until it has been reached a maximum level in the three –established as a parameter- or when there is only one observation in each new child created [7].

Equation 2: Assigning of classes to nodes [7]. ***c(i|j)*** **is the cost of misclassifying** ***i*** **as j,** ***π(i)*** **is the prior probability of** ***i*** **(estimated from the training data),** ***N_i*** **is the number of class** ***i*** **in the training data and** ***$N_i(t)$*** **is the number of samples with class** ***i*** **in the node. We select class if the inequality is satisfied.**

$$\frac{c(i \mid j)\pi(i)N_i(t)}{c(i \mid j)\pi(j)N_i(t)} > \frac{N_i}{N_j}$$

3.2 Naïve Bayes

This classification method is suitable when each instance of the problem can be represented as tuples of attributes of the form $\{a_1, a_2, a_3, ..., a_n\}$, and the target function evaluated on each instance, *f(x)*, can take any value from a finite set of values *V* [8].

The prediction of the target function for new instances is obtaining getting the most probable target value V_{max} as is described in equation 3 [8].

Equation 3: Most probable target value for a given sample

$$V\max = \underset{v_j \in V}{\arg\max}\, P(v_j \mid a_{1,} a_2, ...a_n)$$
$$= \underset{v_j \in V}{\arg\max}\, P(a_{1,} a_2, ...a_n \mid v_j) P(v_j)$$

The two probabilities involved in equation 3 are calculated assuming that the conditional probabilities of the features are independent. This assumption leads us to the Naïve Bayes classifier which is expressed by equation 4 [8].

Equation 4: Naive Bayes Classifier. Where a_i refers to the i-th attribute of the sample and ***v_j*** **denotes the j-th possible class value.**

$$Vnb = \underset{v_j \in V}{\arg\max}\, P(v_j) \prod_i P(a_i \mid v_j)$$

The probability *$P(v_j)$* is estimated counting the frequency of each possible class in the training data., The probability *$P(a_i \mid v_j)$* is estimated using equation 5 [8].

Equation 5: m-estimate of probability

$$\frac{n_C + mp}{n + m}$$

In equation 5, *n* correspond to the total number of time that v_j appears in the data and n_c is the number of times of v_j in the presence of the attribute value a_i, p corresponds to a prior estimation of the probability of for this case is set to *1/k* (where k is the number of possible values). [8]

4. Implementation

Given that both methods can use the same problem representation, that is a set of tuples in the form x_i ($a_1, a_2, a_3, ..., a_n$,) and classes c*i* that characterized the sample, a matrix representation is used to store the attributes values and classes calculated from the training and testing data.

In the present work the entire implementation was done in *R*. The library XML was used to extract the data, the library *rpart* was used to perform the classification tree and the library *e1071* to run naïve Bayes.

The implementation was organized in three different R script files: f_engine.R, training.R and testing.R, in the following sections is described the organization of each file and the main R functions used.

4.1 Feature extraction (Script f_engine.R)

First lines of this script define the following parameters:

- w1 and w2: Size of the window for the morphological features: w1 represents the number of words before and w2 the number of words after.
- ww1 and ww2: Size of the window for the word length feature
- www1 and www2 : Size of the window for the word class feature

The main function of this script is *getFeatures.* This function constructs the matrix with the feature values for each sample of the training or test data. For this purpose the function has too arguments: the path of the XML file from which we will extract the features and the vector with the named entity references.

Also this script defines several functions to extract the following features:

- Morphological information (Function morphoF): This feature is calculated in a window of size specified by the parameters *w1* (number of words consider before the name entity) and *w2* (number of words consider after the name entity). In fact for each word is generated the three following features:

    - POS (1)
    - Gender (3)
    - Case (5)

- Word Classification (Function wordClass): This function assigns a class to the word in evaluation and its context. The size of the window is specified by the parameters *www1* and *www2*. The possible classes are listed above:
    - L: all letter are lowercase
    - C: starts with capital
    - U: all uppercase
    - N: contains a number
    - A: all numbers
    - Z: other
- Hyphened (Function wordHyphen): This function determines in the word contains special characters or if it is hyphened. The word is classified in one of the following options:
    - H: constains hyphen
    - D: contains dot
    - M: constains $
    - F: contains ‘/’
    - P: contains ‘%’
    - Z: other
- Word length (Function wordLength): This function calculates the length of the word and its context. The window size is specified by the parameters *ww1* and *ww2*.

- Ne distance : This feature is calculated inside the function *getFeatures* and represent the number of words between the word in evaluation and the previous two named entities found – the value store correspond to the mean of the two distances-.

- Position of the word in the sentence and length of the sentence are calculated inside the main function as well (getFeatures)

It is important to mention that numerical features were categorized using quintiles, which are quantiles with 5 regular intervals, each one having the same number of items. The motivation of this categorization is to let both machine learning methods to create more general rules, avoiding some over fitting. For this purpose the data was preprocessed in order to obtain the correct interval of values for each non categorical feature. The function *mycut* evaluates a given number and returns its category according to the intervals preprocessed. Each category is represented by a letter from the following set: {A,B,C,D,E}.

4.2 Training and testing script

The training and testing script files start including the script *f_engine.R* which contains the functions needed to extract and calculate the features from the training and test XML files. After this line it is necessary to specify three different paths:

- *path*: path in hard disk for the file **.ne.oneword.xml*

- *path2*: path in hard disk for the file **.m.xml*
- *file_*= path where the user wants to save the table with the features value matrix (and its classes for the training case).

Then, the script construct two vectors with the classes and references from the **.ne.oneword.xml* files, these vectors are called *type* and *id* respectively. In this part of the script the methods *xmlTreeParse, getNodeSet* and *xmlAttrs* are used to access to the XML hierarchy, extract the nodes and access the fields of the node.

Once we obtain the references of the named entities samples – and its classes-, we call the function *getFeatures* which is defined in the file *f_engine.R*. This function construct the matrix with the features calculated from the training or text data.

The training script defines the function *train*, which receives two parameters: an integer value specifying the method that the user wants to use, and the matrix with the features and class values of the samples – the matrix returned by the method *getFeatures*-. The integer value for the current implementation can have only two values: 1 for decision trees and 0 for naïve Bayes. This function returns in a variable the model obtained with the chosen training method.

For decision trees method it was used the function rpart -with the method "class"- using the following features:

- Morphological tag (in a window defined as: w_{i-2},w_{i-1}, w_i, w_{i+1}, w_{i+2})
- Word class,
- *ne* distance

It is important to mention that other features were also used in the experiments but the features mentioned above produced the best results – a further analysis can be found in the next section of this document-.

The previous script chooses automatically the complexity parameter associated with the smallest cross validation error [9]. A further analysis concerning cross validation and the pruning of the tree will be covered in the results section.

The function *naiveBayes* is used to perform the Naïve Bayes method. This function receives two arguments: the matrix with the training feature values and classes, and the function with the attributes to use in the training. The following features are specified in the attributes expression –which correspond to the best combination of features found-

- Word class
- Word length : with window context of size 1: ,w_{i-1}, w_i, w_{i+1}
- Morphological tag: window of size 1 : w_{i-1}, w_i, w_{i+1}
- Ne distance

Similarly the script test.R defines the function *test* which has three arguments: the kind of model to use for the test – Naïve Bayes (0) or decision trees (1)-; the model itself – returned by the method *train*- and the matrix with the features values, which is generated once the test.R script is loaded. The script shows the accuracy of the model chosen with the test data provided.

5. Experiments and Results

Several experiments were run for both machine learning algorithms to understand the behavior of the features in each model and evaluate their suitability for the two algorithms. Also it was explore the convenience of some features applied in context using different window sizes. Another aim of the experiments was to analyze the importance of specialized linguistic information as features, case of the morphological tag, which let us discern at the same time which simple features can lead to decent predictor models.

The different settings of the models explored in following sections were tested using the development data.

It is important to mention that the baseline of the problem studied in this work is 44%, which correspond to the frequency in the data of the type *p*. This percentage gives a intuition of the difficulty nature of the problem of named entity classification.

5.1 Classification tree experiments

For the classification tree algorithm, several window sizes were set in order to discern the most convenient one given the limited amount of training data.

The experiments were carried out using the *word class* feature and increasing the size of the window steadily for the feature that is being studied. It is important to mention that the accuracy for the evaluation data obtained using only the feature mentioned was 52.5%, which is high considering its simplicity. However this can be explained by the fact that the most frequent named entities are *ps*, *pf*, *gc* and *gu*, which correspond to 54% of the named entities in the data and follow patters discern by the word class feature (for instance the words that starts with capital letters).

Table 8 summarized the results obtained. The first experiment showed that adding the features POS, gender and case increase the accuracy by 17.3%. Furthermore, increasing the size of the window of these features showed an augmentation in the accuracy of 4.24% (considering one word before and one word after). This result shows that using morphological features lead to significant improvements in the accuracy of the model. However, it was found that bigger windows don't produce better results. This can be explained by the fact that using bigger windows implies using bigger training data.

The same experiment was carried out to find the most suitable size window for the features *length of word* and *hyphen classification*. However these attempts did not produce better results. This result is understandable since there are not many samples with the special characters that are matched with the feature *hyphen*. Also there is not a clear trend in the word size according its *ne* type, for this reason this feature does not provide information to the method.

After, the tree was trained using all the features describe in section 4.1 – with windows of size 2 for context features-. This experiment is useful because classification tree learning involves well defined criteria to select the best attributes at each step, then the method is able to discard irrelevant features leading to the best combination of them. The tree obtained using this approach is shown in Figure 2. The method was training using the features: Morphological tag (with window size = 2), word class, *ne* distance, hyphen class, length of the sentence, length of the word, position in the sentence. The accuracy obtained was 74.55%. However Figure 2 shows that only the features gender, case and POS (for w_i); gender and POS for w_{i-1}; gender for w_{i+1}, *word class* and *ne distance* were used.

A better understanding of the learning algorithm behavior can be obtained through the confusion matrix of the prediction. Table 9 illustrates three different confusion matrices: Table 9.1 shows the confusion matrix for the prediction using only the word class feature, Table 9.2 shows the confusion matrix for the prediction using the word class feature and POS, and the last table shows the matrix for the best combination of features found (which are enumerated above). These matrices were chosen given the significant accuracy increase obtained by the different predictions as can be seen from Table 8. Also, this selection eases the analysis of the features impact and its behavior.

Table 9.1 shows that types *t* and p are discerned correctly by the learning algorithm when only the feature *word classification* is used. However it misclassifies the types *g,s,i* and *o* since they are classify as *p*. These results follow the nature of the word classification feature which detects numbers (classifying type *t*). Nonetheless, it is not able to distinguish between the types *g,s,i* and *p* because the patters considered for this feature are the same for the name of people, places and institutions.

Table 9.2 illustrates how the accuracy increase is obtained when morphological features are added.

Morphological information let the decision tree to classify quite well the type *s* and discern partially between the types q and p. A better classification of these types leads to a considerable accuracy increase (in this case 18.7%) given their high frequency in the data (types *p,q* and *s* sum up around the 70% of the training data). However types *m,o,i* are still misclassified

Table 9.3 shows that if we consider morphological information in context (using a window of size two) and the named entity feature, type *p* and *q* are better discerned. Nonetheless this table also shows that around 7.07% of the names with type *p* and *q* are still misclassified. Similarly the algorithm remains unable to recognize types *m,o* and performs poorly classifying *i*. However special emphasis should be stress in the design of features for the recognition of name entity types *p* and *q* because these two are the most frequent in the data and slight improvements in its detection represents high increase in the accuracy of the entire prediction.

Additionally, it was examined the cross entropy errors results, which are shown in Figure 3. This graph is useful to decide where to prune the tree since it relates the relative error with the size of the tree. For our particular case it turns out that cutting the tree at 9 (the size is given in number of nodes) it is obtained a equivalent tree in terms of relative error but a more general one giving that smaller trees represent more general models that may behave better in future test data, or in other words the prune of the tree helps to reduce the overfitting. The tree can be pruned using the R method *prune(model, cp,..)*, where model is the trained tree and *cp* is the complexity parameter to cut the tree. The complexity parameter with the smallest cross-validation error is assigned –according Figure 3 the value 0.013 is chosen, which means that the tree was cut at size 9. This can be done with the following script [9]:

cp= mode$cptable[which.min(mode$cptable[,"xerror"]),"CP"]

The accuracy obtained when the tree is pruned was 73.27%. Table 9.4 shows the resulting confusion matrix after pruning the tree. The pruning of the tree decreases slightly the accuracy of the prediction of the type *p* and *i*. However the pruned model seems to predict more accurately type s. It is interesting to note that most of the changes occur because the model stop attempting to classify the instances as type *i* since the corresponding row has only 0 values. This suggests that the prune process cut some of the rules where the type *i* was considered.

Table 8: Results obtained by the Classification tree method. Word C stands for word classification feature and tag(*n*) for morphological information (case, POS, gender) applied if a window of size *n*

Features	*Accuracy*
Word C	52.5%
Word C, tag	69.8%
Word C, tag(1)	74.04%
Word C, tag(2)	74.55%
Word C, tag(3)	74.55%

Table 9: Confusion matrices for the prediction of the decision tree: 4.1 Matrix when word classification attribute is used; Table 4.2 Matrix when morphological information is added; Table 4.3: Matrix for the best combination of features; Table 4.4: Matrix for the best combination of features after the tree is pruned. Bold values show the changes. *Note: Columns represents the predicted values and rows the real values.*

9.1)

T\P	a	g	i	m	o	p	s	t
a	***0***	0	0	0	0	0	0	0
g	0	***0***	0	0	0	0	0	0
i	0	0	***0***	0	0	0	0	0
m	0	0	0	***0***	0	0	0	0
o	0	0	0	0	***0***	0	0	0
p	0	178	72	8	54	***334***	38	7
s	0	2	1	1	0	0	***4***	0
t	6	0	0	0	4	0	0	***73***

9.2)

T\P	a	g	i	m	o	p	s	t
a	***0***	0	0	0	0	0	0	0
g	0	***166***	41	5	19	58	1	6
i	0	0	***0***	0	0	0	0	0
m	0	0	0	***0***	0	0	0	0
o	0	0	0	0	***0***	0	0	0
p	0	4	10	1	7	***265***	3	1
s	0	10	22	3	29	11	***38***	0
t	6	0	0	0	3	0	0	***73***

9.3)

T\P	a	g	i	m	o	p	s	t
a	***0***	0	0	0	0	0	0	0
g	0	***149***	36	5	5	18	2	6
i	0	2	***11***	3	13	5	7	0
m	0	0	0	***0***	0	0	0	0
o	0	7	0	1	***16***	2	0	0
p	0	14	15	0	5	***303***	2	1
s	0	8	11	0	16	6	***31***	0
t	6	0	0	0	3	0	0	***73***

9.4)

T\P	a	g	i	m	o	p	s	t
a	***0***	0	0	0	0	0	0	0
g	0	***149***	36	5	5	**24**	2	6
i	0	**0**	***0***	0	**0**	**0**	**0**	0
m	0	0	0	***0***	0	0	0	0
o	0	7	0	1	***16***	2	0	0
p	0	14	15	0	5	***297***	2	1
s	0	**10**	**22**	**3**	**29**	**11**	***38***	0
t	6	0	0	0	3	0	0	***73***

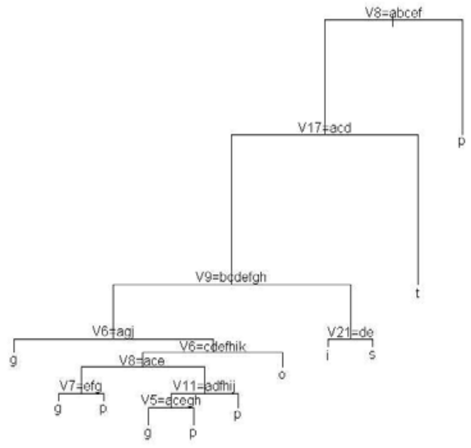

Figure 2: Decision tree obtained: In the tree is shown that only the features V8(Gender w_i), V16 (word class), V9 (Case w_i), V6(POS w_{i-1}) , V17(ne distance), V5 (Gender w_{i-1}), V11(Gender w_{i+1}) were used.

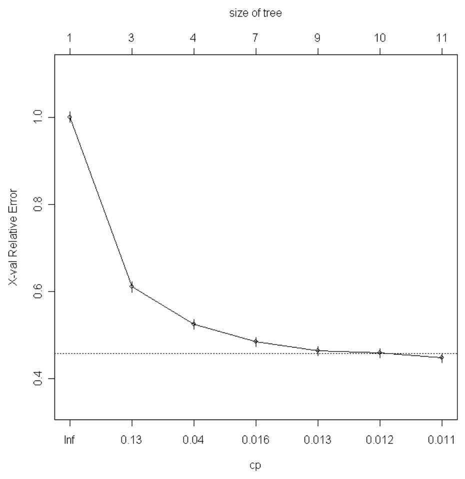

Figure 3: Cross validation error

5.2 Naïve Bayes experiments

The approach followed to study the Naïve Bayes method was to run a series of experiments were each feature was tested independently to see its impact in the model, similarly the test were done in first place on the development data.

Additionally several experiments were run in order to find the most suitable windows size for the training data. The results are summarized in Table 10. These results are for training the method with the feature specified in the table plus the *word class* and *ne distance* features.

Given these results, it is possible to state that the features *hyphened classification* don't provides a significant improvement in the accuracy, which is consistent with the result found for classification trees.

Additionally, the results of the Table 10 show that the features that improves the accuracy for Naïve Bayes are word length, length of the sentence, case, gender, POS, and word classification The best results were obtained using windows size of one (for word length and morphological information) and two for word classification. Note that previously the feature word length was not considered by the classification tree method and for this method represents a 5.68% increase of accuracy.

Further experiments were carried out to combine the features that represented improvements in the accuracy (from the analysis done in Table 10). These experiments shown that the best combination of features for this method is: word classification (applied in window of size 2), morphological information and word length, both applied in window of size 1. The accuracy obtained applying these features was 74.8% (on development data). The features chosen are similar to the ones found for the classification tree algorithm; however for Naïve Bayes the sizes of the windows were different and the feature word length becomes important in the model.

Additionally, an important difference is the higher accuracy obtained without considering morphological information, which is 65.7% (with the features word length, word class and length of the sentence) against 52.5% obtained by the previous method.

In addition, it was found that adding several of the features that increase the accuracy not necessarily lead to better results, contrary to classification trees where the agglomeration of features did not decrease the accuracy. This can be explained by the fact that the estimated probabilities for some features can be very low, result that is propagated in equation 4 creating some biased in the sense that the probability will be underestimated. This case was not present in decision trees, where not relevant features are ignored by the method.

Table 11 summarizes the results of these experiments. From this table we can see that when the features word class, morphological tag and length of the sentence are combined the performance of the system decrease, even if the features increase the accuracy of the system when are tested independently.

Also a confusion matrix was built for the best combination of features found for this method; the results are shown in Table 12. From this table it is possible to state that the name entities types classified correctly are very similar to the ones classified for the classification trees method (*g*,*p*,*s*,*t*). However the system confuses more often the type *p* and *q* which are very frequent in the data and consequently have a bigger impact in the performance of the algorithm. It is also interesting to note that this algorithm discern the name entity type *o* (Artifact names) which is highly misclassified by the classification trees algorithm. Even if words with this name entity type only represent 5% of the training data, it would be more convenient to use Bayesian learning when the detection of Artifact names is more relevant or are more important in a specific application.

It is important to mention that several of the experiments described so far were repeated including Lagrange smoothing but it was not found better results.

5.3 Algorithms comparison

Finally, a *paired test* [8] was performed on both algorithms in order to compare their performance and be able to determine the confidence interval in which one algorithm outperforms the other. For this purpose the training data was divided into 10 disjoint subsets of 570 elements each one.

The parameter to be estimated is the expected error difference of both algorithms on a data sample which is subscript to the instance distribution. The estimators are shown in equation 6 [8]:

Equation 6: Paired t estimators

$$\bar{\delta} = \frac{1}{k} \sum_{i=1}^{k} \delta_i$$

$$s = \sqrt{\frac{1}{k} \sum_{i=1}^{k} (\delta_i - \bar{\delta})^2}$$

And the interval is calculated using equation 7.

Equation 7: Paired t estimators

$$\bar{\delta} \pm t_{N,k-1} s_{\bar{\delta}}$$

Each value δ_i in equation 6 correspond to the error difference of the two algorithms when the training data provided correspond to the union of *k-1* subsets and the test data is the *i-th* set. The results obtained for a confidence level of 95% were: 0.039 ± 0.029. This means that the error of Naïve Bayes is bigger than the error of decision trees between 1% and 6.8% given 95% as confidence level.

Additionally, a bootstrap algorithm was performed. For this purpose it was trained a model for each algorithm choosing the best combination of features found. Then each model was tested using 1000 subsets of the evaluation data. Each subset was of size 100 and it was built randomly from the evaluation data.

Two different measures were taken. First, the error measure for each one of the 1000 experiments was store in a vector. After this vector was sorted in order to obtain the elements 25th and 975th of the vector which correspond to the interval with confidence level 95%. For the classification tree model the interval obtained using the method was: [12%, 32%]. For Naïve Bayes the interval was: [16%, 42%].

Also it was calculated the interval using equation 7. The estimator applied were the mean of the vector containing the errors and the standard deviation. The results for both methods are summarized in Table 13.

6. Conclusions

The best results were obtained using similar features in both methods. For classification trees the best combination of features was: morphological tag considered in a window of size 2, name entity distance and word classification. The features that provide the best results for the Naïve Bayes method are word classification (applied in window of size 2), morphological information and word length, both applied in window of size 1. The accuracy obtained by the decision tree using the evaluation data was 78.49% and for Naïve Bayes 74.7%

Nonetheless, classification trees tend to be more convenient for the *ne* type classification with the features considered, since it was obtained better accuracy results. More precisely, the paired test performed to compare both algorithms showed that with a 95% confidence level, decision trees outperform Naïve Bayes by a value between 1% and 6.8%. However the bootstrap method shows that even if the classification has a smaller and lower error interval, not always it can outperforms Naïve Bayes, since the intervals overlap.

Additionally it is handier to evaluate the impact of potential features in classification trees since the method employs a measure to quantify "how good" is a feature to split the hypothesis space, contrary to Naïve Bayes in which features with low probability can bias the class estimation, making more difficult the analysis.

However, promising results were obtaining with Naïve Bayes when morphological features were ignored. This result is interesting in the sense that in some real applications it is possible to do not have access to such kind of information.

Furthermore, the accuracy obtained by both methods can be improved merging some of the named entity types. The classification tree generated shows that no rules were generated for the types *a*, *m* and *f*, which occurs given its low frequency in the data. However, this merging can be conditioned to the situation in which the system is implemented. Similarly, ignoring the double annotations for some of the samples in the training data can produce better results. Nonetheless ignoring these samples or some of their annotations may be dependent the context of the language application as well.

Table 10: Naïve Bayes accuracy results: s_length refers to length of the sentence, w_position to word position in the sentence, w_length to the length of the word, tag to the gender, case and POS of the word, class to the *word class* feature. Parenthesis state the size of the window used.

Features	*Accuracy*
Word Class	52.50%
S_length	57.80%
w_position	58.00%
w_length	55.11%
w_length(1)	58.56%
w_length(2)	58.18%
w_length(3)	58.18%
Hyphen	52.55%
tag	71.09%
tag(1)	72.25%
tag(2)	71.86%
class(1)	55.11%
class(2)	55.37%

Table 11: Naïve Bayes accuracy results: s_length refers to length of the sentence, w_position

Features	*Accuracy*
class(2),w_length(1)	64.32%
class(2),w_length(1),tag(1)	74.8%
class(2),tag(1)	73.4%
class(2),Ne,tag(1)	72.89%
class(2),wlength(1),s_length	65,72%
class(2),tag(1), s_length	72,63%
class(2),tag(1), wlength(1),S_length	74,29%
tag(1), wlength(1),S_length	73.65%

Table 12: Naïve Bayes: confusion matrix for the best combinations of features found (using development data): ***Note: Columns represents the predicted values and rows the real values.***

T\P	a	g	i	m	o	p	s	t
a	***2***	0	0	0	0	0	0	0
g	0	***124***	24	2	0	13	0	6
i	0	16	***13***	3	1	13	1	0
m	0	1	1	***0***	0	1	1	0
o	0	6	0	1	***36***	6	1	1
p	0	26	16	0	8	***299***	1	0
s	0	7	19	3	13	1	***38***	0
t	4	0	0	0	0	1	0	***73***

Table 13: Bootstrap analysis: Results of the bootstrap algorithm on both algorithms. The confident interval of the error has a confidence of 95%.

	Mean	Standard deviation	Interval
C. Trees	24,9%	3.79%	[17.3%, 32.4%]
Naïve Bayes	31.9%	5.3%	[21.4, 42.3%]

Table 14: Confusion matrices using evaluation data on both methods. Table 14.1 shows the confusion matrix for the classification tree model. Table 14.2 shows the confusion matrix for Naïve Bayes. ***Note: Columns represents the predicted values and rows the real values.***

14.1)

T\P	g	i	m	o	p	s	t
g	***125***	27	1	4	18	1	4
I	6	***11***	5	13	3	5	0
m	0	0	***0***	0	0	0	0
o	5	3	0	***7***	0	0	0
p	10	4	0	9	***327***	0	3
s	4	8	7	8	4	***43***	0
T	0	0	0	5	0	0	***88***

14.2)

T\P	g	i	m	o	p	s	t
g	***101***	22	1	4	25	1	4
i	27	***5***	0	3	8	0	0
m	0	1	***1***	1	0	1	0
o	5	4	0	***22***	10	1	2
p	11	5	0	5	***306***	0	3
s	6	16	11	11	3	***46***	0
t	0	0	0	0	0	0	***86***

According the confusion matrices, the features showed similar behavior in both learning algorithms since it is possible to discern between the classes *p*,*q*,*t* and *s*. However it is important to state that there is still significant place for improvement if the distinction of these classes, particularly future work should focus in the design of features able to classify the name entities that belong to the types p and q because these are the types that have the biggest impact in the accuracy given it is high frequency in the data. Also new features are needed to classify the name entity types *m* and *i* which are almost always misclassified by the methods explored.

Interestingly, Naïve Bayes was able to classify instances that belong to the type *o* contrary to Decision trees. This suggests that it would be more convenient to apply this method in scenarios where this entity type is more common.

Finally, the experiments performed in both methods show that simple features as the nature of the word characters – if it is capital, numerical, etc – provide significant information to the classifier. Additionally, morphological information represents an important source of information to improve the quality of the models and more if it is analyzed in the context of the words evaluated. However the size of the window seems to be conditioned to the size of the data. Training and test with bigger data would be helpful for future work.

References

[1] http://www.cisuc.uc.pt/lct/view_project.php?id_p=70
[2] Named Entity Recognition using an HMM-based Chunk Tagger. GuoDong Zhou Jian Su, Kent Ridge Digital Labs, Heng Mui Keng Terrace, Singapore 119613
[3] Beyond Named Entity Recognition Semanticlabelling for NLP tasks Centro Cultural de Belem
LISBON, Portuga 25th may 2004 In Association with 4th International conference of language resources and evaluation,LREC2004. Main conference 26-27-28 May 2004
[4] http://ufal.mff.cuni.cz/~hladka/project.html
[5] http://www.statistics.com/resources/glossary/c/cart.php6] Modern Applied Statistics with S Fourth edition by . N. Venables and B. D. Ripley, Springer (mid 2002)
[7] An Introduction to Classification and Regression Tree (CART) Analysis. Roger J. Lewis, M.D., Ph.D. Department of Emergency MedicineHarbor-UCLA Medical Center. Torrance, California
[8] Machine Learning, Tom Mitchel, 177
[9] http://www.statmethods.net/advstats/cart.html