Universitat de Lleida

TREBALL FINAL DE MÀSTER

Estudiant: Gerard Farré i Gomez

Titulació: Màster en Enginyeria Informàtica

Títol de Treball Final de Màster: Twitter Sentiment Analysis

Director/a: Francesc Giné De Sola, Marc Solé Farré

Presentació

Mes: Setembre

2019

Contents

List of Figures

List of Tables

Listings

1 Introduction

In the last years, compute the sentiment detection it has become a challenge. The growing of machine learning techniques helped a lot to improve the accuracy of the sentiment prediction. In this project, we will deep on all the steps involved to deep learning, the different available techniques and the analysis of the results.

1.1 Sentiment analysis

When a person writes (or spokes) some opinion, there is a polarity associated with it: **Positive** or **negative** opinion.

Most of the times, is easy for a person to identify the polarity of the text, but how a machine, who doesn't know what a sentiment is, can identify it?

This process involves a research of mathematical algorithms and different techniques to match a sentiment of a text with the least margin of error. This is called sentiment analysis.

The machine learning is the most efficient technique to deal with this problem, because it is able to classify data (features) into different labels, two in this case, positive label and negative label (a binary classification problem). The success of this technique involves the quality of the input data, the chosen algorithm and its parameters.

1.2 Data source: Twitter social network

To make our system works, we need a lot of people's opinion to make the algorithm learn and validate it. In other words, we need a lot of data. Best site to get this opinions is a social network, which has a high amount of interactions between users.

The chosen social network is Twitter, created at 2006 in San Francisco, California. His popularity has grown quickly because allow to users express his opinion about things like politics, art, sports, etc. instead of his own life (family, friends, holidays, etc.) like users do on other social networks, for example Facebook. Actually Twitter has 321 million of monthly active users.

Besides, there is a limit of 280 characters for each message (140 chars. before November, 2017), forcing the user to make his opinion brief, clear and concise,

which is perfect for sentiment analysis.

Twitter provides an API[1], that allows to extract all the data to build our dataset in an easy way.

1.3 Project objectives

The objective of the project is to study, use and compare the different existing Machine Learning techniques to achieve the best accuracy of sentiment analysis.

This process consists of several phases:

- **Data extraction**: This phase is involved on get the input data from the source and build our dataset.
- **Data cleaning**: The extracted data can contain noise or irrelevant characters that can interfere on the model training. The data cleaning is the process to correct this data to better fit our model.
- **Model training**: This process use a Machine Learning algorithm to find patterns on the input data and builds a model that is capable to catch this patterns. This process is repeated for each framework and technique.
- **Model evaluation**: The evaluation consists in using the model with a hidden subset of our dataset and check the difference between the actual label and the predicted label. With this difference we can generate an analysis to see the goodness of the model.
- **Results comparison**: With the evaluation of all models, we can show and compare the results to analyze which model performs better with our kind of problem, the binary text classification.

[1] Application programming interface (API) is a set of subroutine definitions, communication protocols, and tools for building software.

2 Techniques for sentiment analysis

If we want to know if the sentiment of a text is positive or negative, clearly is a binary classification problem. There are different ways to classify the sentiment of a tweet. We need to go deep to the available techniques, to analyze which fits better in our use-case.

2.1 Neural networks

A neural network is a set of artificial neurons or nodes that have a directional connections (edges) between them like a directed graph (figure 1), making the analogy of the interconnected network of neurons that our brain has.

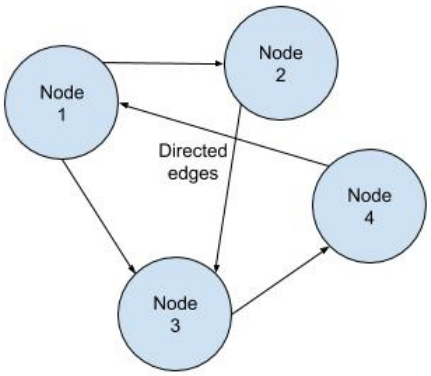


Figure 1: An example of a directed graph

2.1.1 The neuron (node)

A neuron receives the input from other connected nodes, and for each input it has an associated weight. Besides, it receives another input called bias, which is a constant value that helps the model fits better to the data.

First, as we can see in the figure 2, the node computes the weighted sum of its inputs, and then, applies the activation function to finally obtain the resulting output. For example, a neuron with two neurons connected as an input, will compute the following formula:

$$Y = f(X_1 \cdot W_1 + X_2 \cdot W_2 + bias)$$

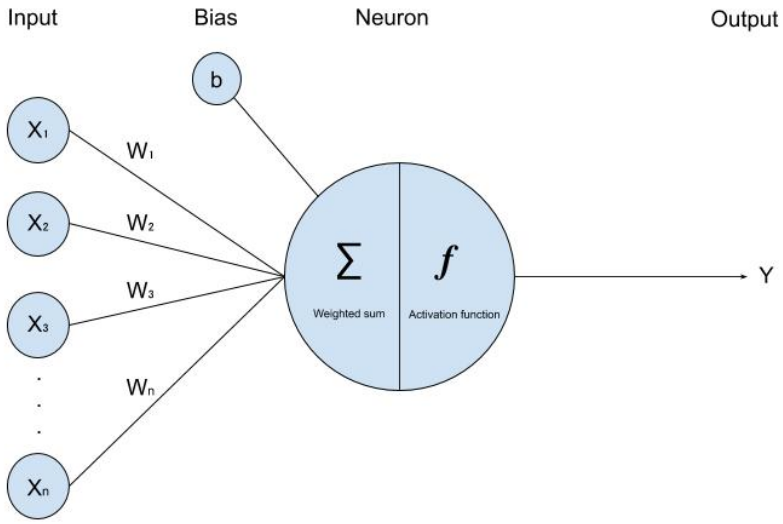


Figure 2: Model of a neuron

The activation function is a non-linear operation and his purpose is to introduce non-linear properties to the network. If we skip the activation function step, the output keeps linear and we obtain a linear regression model, which is too limited and performs bad in most cases.

Some of the most used activation functions are:

- **Sigmoid:** Sigmoid function transform the value into a range from 0 to 1.
$$S(x) = \frac{e^x}{e^x + 1}$$
- **Tanh:** Improves the sigmoid function expanding the range from -1 to 1.
$$tanh(x) = \frac{2}{1 + e^x} - 1$$
- **ReLU:** The Rectifier Linear Unit function changes all negative values to zero.
$$ReLU(x) = max(0, x)$$

2.1.2 Feedforward neural network

The Feedforward neural network is based on layers technique. Each layer has a set of nodes connected to the nodes of the adjacent layers (see figure 3).

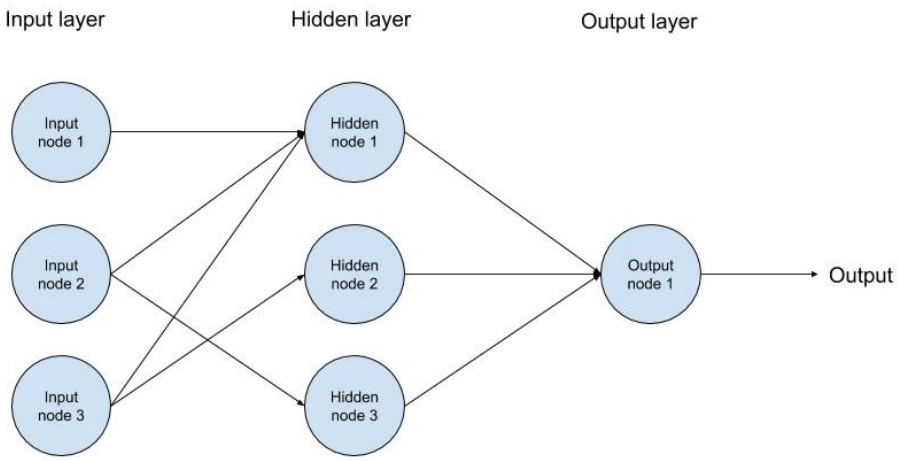


Figure 3: A feedforward neural network representation

A feedforward network has three types of layers:

- **Input layer:** The input nodes takes the information from outside and send it to the nodes of hidden layer.
- **Hidden layer:** The hidden nodes performs all the computations and send the information to the output nodes.
- **Output layer:** The output nodes collects the information computed by the last hidden layer and send it to outside.

The feedforward neural network can have from zero (single-layer perceptron) to multiple hidden layers (multi-layer perceptron). The flow of the feedforward neural network goes to one direction only, forward, from input layer to hidden layers, ending on the output layer.

2.1.3 Convolutional neural network

The convolutional[2] neural network is a variant of feedforward neural network. It has a fully connected[3] multilayer perceptron (at least one hidden layer). The convolutional neural networks makes two extra computations with the input data called feature extractors which are the input of the multi-layer perceptron, as show in the figure 4. This feature extractors are convolution and pooling.

[2]Convolution is a mathematical operation on two functions to produce a third function that expresses how the shape of one is modified by the other.

[3]Each node of a layer is connected to all nodes of the next layer.

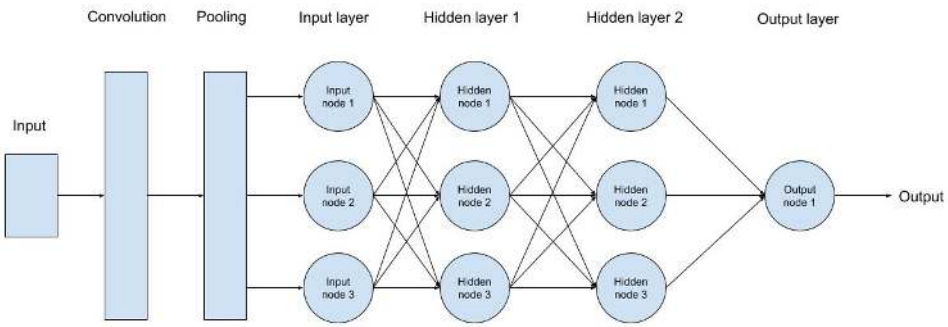


Figure 4: The convolutional neural network architecture

2.1.3.1 Convolution

The convolution is used to create a feature map which indicates where the feature is located. A convolutional network can have more than one convolution. If we imagine our data as a two dimensions matrix, the convolution process applies a filter or kernel what is a same dimension matrix with lower length.

First, we need to superpose the kernel on the top left of the data and make the product of the numbers on the same position and then add all the results. This addition will be the resulting number of the first position of the feature map. The next step is to move the kernel one column and repeat the same process. When the kernel has no columns to displace, we need to move one row down and start again to the left. The process is repeated until all the input data is computed by the kernel with a resulting matrix with the same length as the kernel.

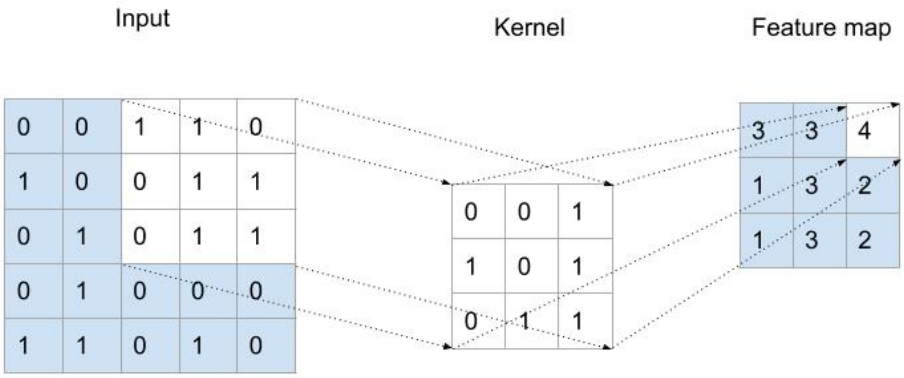


Figure 5: The convolution computation procedure with 2 dimension 5x5 matrix as input and 3x3 matrix on filter

In the example of the figure 5, to get the first number (3) on the position 0,0 of the matrix, first, the convolution process computes the following multiplications.

$$0 \cdot 0 = 0$$

$$0 \cdot 0 = 0$$

$$1 \cdot 1 = 1$$

$$1 \cdot 1 = 1$$

$$0 \cdot 0 = 0$$

$$0 \cdot 1 = 0$$

$$0 \cdot 0 = 0$$

$$1 \cdot 1 = 1$$

$$0 \cdot 1 = 0$$

Finally, computes the sum of the results.

$$0 + 0 + 1 + 1 + 0 + 0 + 0 + 1 + 0 = \mathbf{3}$$

2.1.3.2 Pooling

The pooling is used to reduce the amount of features and consists in applying a fixed operation to a region of the matrix. There are different operations we can use, like max-pooling or average-pooling.

- **Max-pooling:** Take the maximum value of the elements of the region (see figure 6).
- **Average-pooling:** Compute the average of all elements of the region.

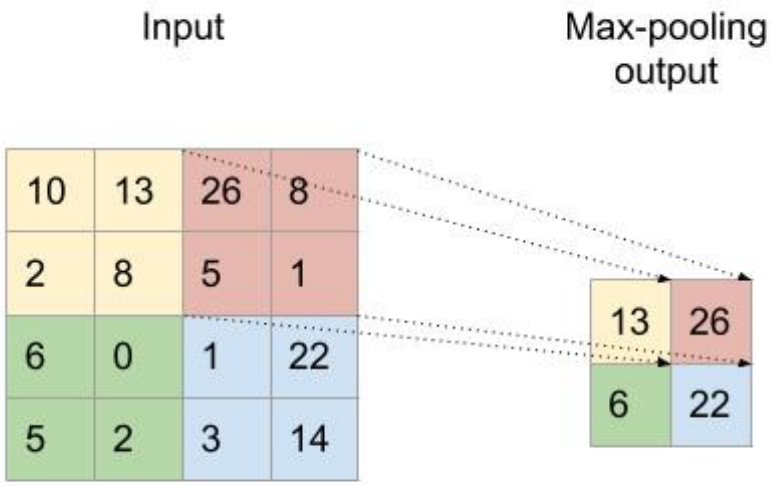


Figure 6: Example of pooling with max-pooling operation

2.1.4 Optimizers

Optimizers are algorithms that help us to reduce the loss and improving the accuracy modifying the internal parameters of the model, like weights (W) or bias (b), which updates the direction for an optimal solution on training process. How this internal parameters are modified it will depend on the used algorithm. The most popular optimization algorithms are:

- **Stochastic Gradient Descent (SDG)**: This algorithm modifies the parameters to find the minimum value of the loss function computing the derivative (gradient) for each parameter.

$$w = w - \eta \bigtriangledown f_i(w)$$

where w is the parameter and $\bigtriangledown f_i(w)$ is the gradient of loss function. There is a variant of SGD with momentum, wich adds a fraction of the vector of the last iteration to the current update vector. Calculating the weighted average of the last iteration can reduce the oscillations to a more sensitive direction and faster convergence.

$$w = w - \gamma V(t-1) + \eta \bigtriangledown f_i(w)$$

- **Adagrad**: The sum of the squares of the gradients are stored for each parameter. This is used to scale the learning rate. When the gradient has a high value, the sum will be high, and the division would make gradient accelerate slowly,

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} \cdot g_t$$

were v_t is the sum of squares of gradients.

- **RMSProp**: This algorithm is a variation of Adagrad but instead of accumulate the squared gradients keeps the average of it. This approach can avoid the problem of a big number on the squared sums and steps get smaller and smaller over the training,

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} \cdot g_t$$

were v_t is the average of squares of gradients.

- **Adam**: Adaptative Moment Estimation is a combination of SGD with momentum and RMSProp algorithms. It uses the squared gradients to scale the learning rate and computes the weighted average of the last iteration. Because we initialize averages with zeros, the estimators are biased towards zero. To correct this problem we can use the bias correction and it consists on compute the value of the first two moments. The $m(t)$ are the mean of the gradients of the first moment and $v(t)$ is the uncentered variance of the gradients of second moment.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

On the next moments, the parameter are updated with the following formula:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t$$

Adam is the most used algorithm because has all features of the other algorithms and it performs better in most cases. This is the optimizer we will use for our CNN model.

2.2 Logistic Regression

The logistic regression is a classification algorithm that uses sigmoid function [4] , seen in neuron activation function on previous section, to transform a linear to a non-linear function avoiding the problem with linear regression.

The figure 7 shows us how the linear regression can't fit all the data correctly. The sigmoid function introduces a curve to the function that allows the algorithm to take care of all the data.

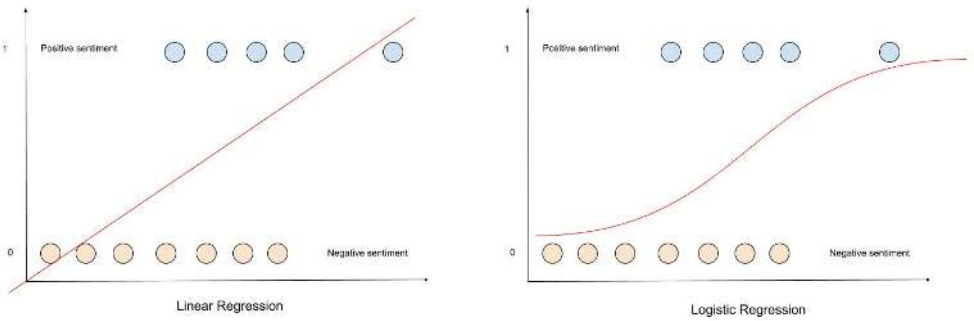


Figure 7: Difference between linear regression and logistic regression

The **linear** regression model can be expressed with the following function:

$$\beta_0 + \beta_1 \cdot x$$

Then we can transform the linear model to logistic model with the sigmoid function:

$$L(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 \cdot x)}}$$

2.3 Naive Bayes

Naive Bayes is a probabilistic algorithm based on Bayes theorem. The Bayes theorem computes the probability of event A occurring based on conditions

[4]Sigmoid function transform a real number into a range from 0 to 1: $S(x) = \frac{1}{1 + e^{-x}}$

of event B:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

To predict if some sentence is positive or negative we need to compute both classes and check which class has higher probability. For example, with the following dataset [1]:

Text	Sentiment
A good car	Positive
Great challenge	Positive
Bad luck	Negative
It's fine	Positive
A never-ending story	Negative
A great observation	Positive

Table 1: Dataset example.

Let's try "a great story" sentence, with the positive probability:

$$P(a\ great\ story|Positive) = \frac{P(Positive|a\ great\ story) \cdot P(a\ great\ story)}{P(Positive)}$$

And the same for the negative:

$$P(a\ great\ story|Negative) = \frac{P(Negative|a\ great\ story) \cdot P(a\ great\ story)}{P(Negative)}$$

The problem is if the dataset doesn't contains the sentence "a great story", the probability is zero, which isn't the behaviour we want. The naive technique consists on compute the probability of each word separately.

$$P(agreatstory|Positive) = P(a|Positive) \cdot P(great|Positive) \cdot P(story|Positive)$$

Now, for each word, we apply the Bayes theorem. Let us calculate the first one:

$$P(a|Positive) = \frac{P(Positive|a) \cdot P(a)}{P(Positive)} = \frac{\frac{2}{10} \cdot \frac{3}{15}}{\frac{10}{15}}$$

- P(Positive|a): How many times "a" is appearing in the positive sentences divided by the total amount of positive words.
- P(a): The number of times "a" appear in the dataset divided by all the words.
- P(Positive): The number of positive words, divided by all the words.

This process is repeated for all the words to obtain the positive probability. Then, as mentioned above, we need to compute the same for negative label and compare which of both results has better probability to finally classify the sentence to the "correct" label.

2.4 Linear Support Vector Machine

Support vector machine is a supervised learning model that consists in classifying all the data to one or the other class.

If we imagine each data entry like a point in a bi-dimensional space, the more similar points are closest. To split the points in two classes we can trace a line in the space called hyperplane, as it shows the figure 8. But, in most cases, we can draw many hyperplanes. Which is the best?

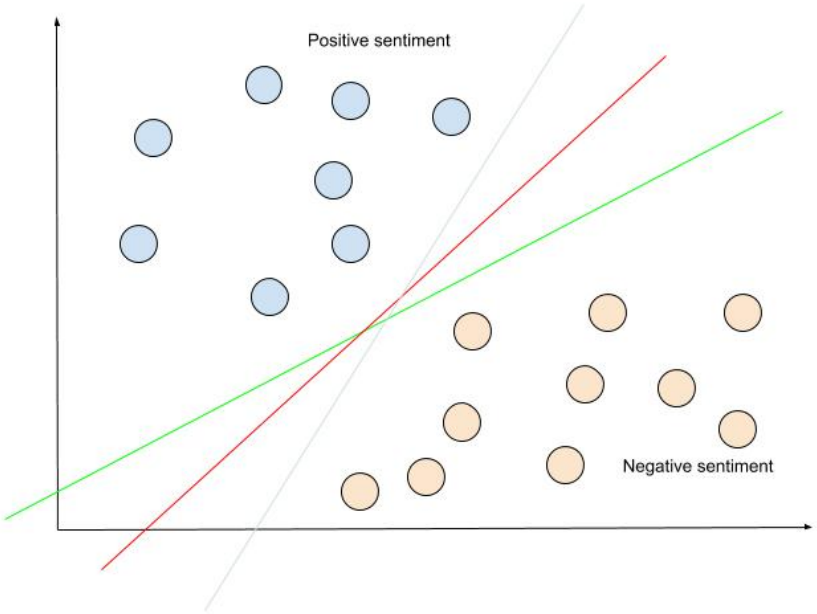


Figure 8: Example of multiple hyperplanes splitting the points in two classes

Each hyperplane has a distance to each class, the margin, so the best hyperplane is the one with maximum margin from both classes (figure 9). To compute the margin we can take the closest point of each class to the center, called support vector, that defines the end of the class region. With the highest margin, we can ensure that the prediction have more confidence because the classes are more distinguishable due the bigger distance.

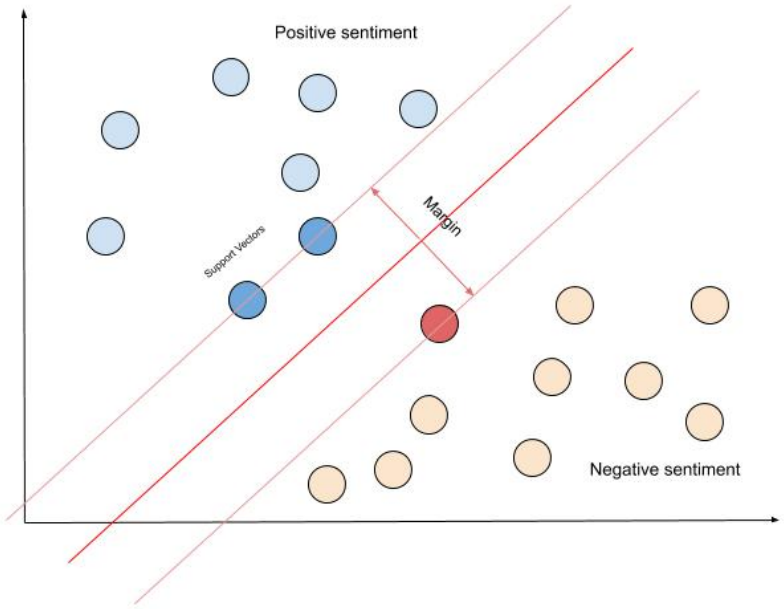


Figure 9: Example of margin from the hyperplane to the support vector

The hyperplane can be defined as $w \cdot x - b = 0$ were w is the normal vector to the hyperplane, x is the point vector and b is the bias[5]. The positive vectors can be explained as $w \cdot x - b \geq 1$ and negative vectors as $w \cdot x - b \leq -1$.

[5]bias is a constant value that helps the model fits better to the data

3 Machine learning frameworks

A machine learning framework[6] is a tool to allow building machine learning models easily, efficiently and faster. There are numerous frameworks to use, but we will focus on two of the most powerful frameworks, Tensorflow and Spark MLlib.

3.1 Tensorflow

TensorFlow is an end-to-end open source platform developed by Google Brain team for building machine learning models. It uses the neural network technique, composed by nodes and his directed edges. Each node has zero or more inputs and outputs. It uses a python interface to provide a client while executes the application in high-performance C++.

Figure 10: Tensorflow logo

The main components of Tensorflow are:

- **Tensors:** A tensor is a generalization of vectors and matrices to potentially higher dimensions. Internally, TensorFlow represents tensors as n-dimensional arrays of base datatypes.

  A tensor have a datatype (int32, float32, string) and a shape. The shape is the number of dimensions and the size of each dimension.

- **Variables:** A variable stores a shared persistent tensor. To create a variable we can call *get_variable()* function:

```
v = tf.get_variable("v", shape=[1, 2, 3], dtype=tf.int32)
```

Listing 1: Tensorflow variable creation.

  The example of listing 1 creates a 3 dimensional tensor with 1 element in the first dimension, 2 elements in the second dimension and 3 elements in the third dimension. The tensor name is v and its elements are integer type.

[6] A framework is an abstraction which provide generic functionality to build a specific application.

- **Graphs:** TensorFlow uses graphs to represent dependencies between individual operations. In a graph the nodes represents units of computations and the edges are the data input or output of the node. This division of work between nodes can help for the system to identify operations that can execute in parallel. In the same way, thanks to the edges we can benefit from a distributed execution using each partition across different computers. Tensorflow provides a default graph and all the elements are added to the graph when are created.

- **Sessions:** Sessions is an interface to connect the client program with the c++ runtime, providing access to devices of the machine and caching information about the graph.

```
with tf.Session() as sess:
  # ...
```

Listing 2: Tensorflow session creation.

The Tensorflow framework has a big amount of algorithms and functions ready to use, which provides us an easy tool to build our Convolutional Neural Network.

3.2 Spark MLlib

The second framework we will use for training the machine learning algorithms is Apache Spark, a data processing engine, with MLlib, a library wich runs the ML algorithms over Spark.

Figure 11: Spark logo

3.2.1 Spark

Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming.

It provides a lot of functions to read and analyze the data from different data sources like Hadoop, HDFS, Cassandra, HBase, Amazon S3, or any text file. The data is read as RDD (Resilient Distributed Datasets) format, which is a collection of elements that can be operated in parallel.

```
# Creates a Spark session
conf = SparkConf().setAppName(appName).setMaster(master)
sc = SparkContext(conf=conf)

# Manual RDD creation
data = [1, 2, 3, 4, 5]
data = sc.parallelize(data)

# Create RDD from file
data = sc.textFile("data.txt")
```

Listing 3: Spark RDD creation.

RDD supports two type of operations, the transformations which creates a new dataset modifying the existing one, and the actions, which returns a value after a computation.

```
# Returns the number of elements of the dataset
count = data.count()

# Returns the first element of the dataset
first = data.first()

# Returns the first 10 elements of the dataset
tenReg = data.take(10)
```

Listing 4: Spark basic actions.

The operations includes the Map-Reduce programming paradigm, and consists on the *map* operation, that applies a function to all elements of the dataset and builds a new dataset with the results. The *reduce* operation aggregates all elements of the dataset using a function and returns the result.

```
# Creates a dataset with a pairs of the element and number 1
pairs = data.map(lambda s: (s, 1))

# Counts how many times the element appears
# adding the 1 number for each occurrence
counts = pairs.reduceByKey(lambda a, b: a + b)
```

Listing 5: Spark basic actions.

3.2.2 MLlib

The MLlib library provides a parallel and scalable high level tools to use common learning algorithms (like classification and regression), feature extraction and transformation, and other utilities.

There are a lot of different algorithms for feature extraction, for example the CountVectorizer that convert text to vector of counts.

```
cv = CountVectorizer(inputCol="words", outputCol="features")
model = cv.fit(data)
result = model.transform(data)
```

Listing 6: MLlib feature extractor.

Or for the feature transformation, like the StringIndexer, which encodes a column of string labels to a label indices.

```
indexer = StringIndexer(inputCol="category",
                        outputCol="categoryIndex")
indexed = indexer.fit(data).transform(data)
```

Listing 7: MLlib feature transformer.

To train a Machine Learning model with the prepared features, for example Logistic Regression, and finally do the evaluation with the test set, it's as easy as:

```
lr = LogisticRegression(maxIter=10, regParam=0.3)
model = lr.fit(training)
results = model.transform(test)
```

Listing 8: MLlib model training and testing.

4 Input data

The most important part of building a model is the dataset. A good quality dataset can achieve high accuracy models, so we will need to pay attention to all the steps to get the input data prepared for training the model, as the diagram of figure 12 shows: extraction, cleaning and transformation.

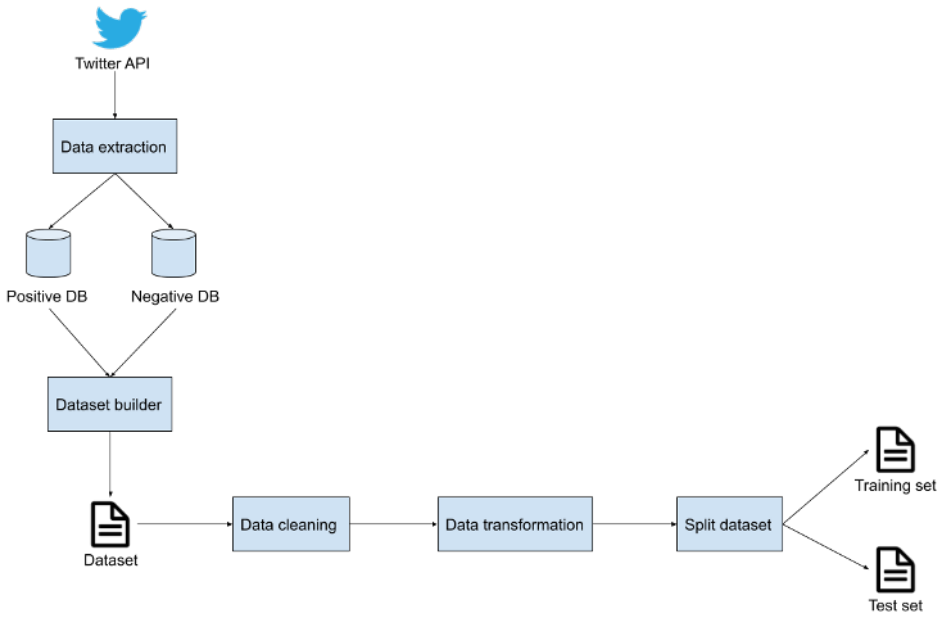


Figure 12: Data steps diagram

The dataset is required to have:

- **Features**: The data we want to use as entry data to obtain a prediction. In our system will be the content of the tweet.
- **Label**: Is the result of the problem we want to solve. If the goal is to predict if the text sentiment is positive or negative, each input needs to be labeled with "positive" or "negative", or more exactly, as 1 or 0 respectively.

4.1 Data extraction

The extraction of the data involves the social network Twitter wich has an API[7] that allows to get the tweets in real-time with specific filters (see listing 9).

The filters used to obtain the data are the following:

- Language: Spanish.
- Keywords: A set of specific words and emoticons.

The function of the extractor is to get the all the tweets that match this filters and save it to the database. But the data is not labeled yet, we need to know if this tweet is positive or negative. The approach followed by the extractor are the keywords. The words and emoticons are divided in positive and negative keywords in order to classify each tweet to the right sentiment and save it to the appropriate database.

```
twitterStream = new TwitterStreamFactory(...);
filterQuery = new FilterQuery();
filterQuery.track(negativeList.toArray(new String[0]));
filterQuery.language("es");
twitterStream.filter(filterQuery);
```

Listing 9: Code of tweet extraction using Java Twitter API

Given the two databases, the positive and the negative, with a big amount of data, we are able to take the data from both and build a single dataset, labeling the sentiment to 0 if come from the negative database and 1 if come from the positive database, as shown in listing 10.

```
for(Object tweet: negativeTweetDatabase.getTweets()){
    tweet.sentiment = 0;
    tweets.add(tweet);
}
for(Object tweet: positiveTweetDatabase.getTweets()){
    tweet.sentiment = 1;
    tweets.add(tweet);
}
saveTweetsToFile(tweets);
```

Listing 10: Code of building dataset from database

[7] API is a programmatic interface consisting on exposed endpoints to a defined request–response message system, typically expressed in JSON or XML.

Finally, we have the output dataset with the following attributes for each tweet[2]:

- **Id**: Original id of the tweet in twitter API.
- **Sentiment**: The label that indicates if the tweet is positive or negative.
- **Source**: The software that extracted the tweet.
- **Text**: The content of the tweet.

The most important columns of the dataset are the *Text* (the features) and the *Sentiment* (the label), which will be used to train the model. The *id* and *source* columns are only for informative purposes and will be omitted on the read of the file in the next phase.

Id	Sentiment	Source	Text
907783668227342336	0	Tweet extractor	Horrible music

Table 2: Example of a row of the dataset.

Two different size datasets have been built for this project to compare how the amount of data can improve or not the accuracy of the model. The smaller dataset contains 428,688 registers and the vocabulary size is 122,252 words. The bigger dataset contains 1,306,478 registers and the vocabulary size is 284,290 words.

4.2 Data cleaning

The data cleaning is a critical phase for the success of building a ML model. It consists in finding and correcting all the errors, inconsistencies and duplicity's of the data. Avoiding noise and removing the parts that disturb or does not contribute to anything improves the training process, reducing the amount of data to be processed and obtaining better explanatory data that helps to take better decisions.

Because we have built the dataset, we are sure that the data of "label" is consistent and we can avoid the cleaning of this column. The data that we will clean is the "text" column, which is the data that we have no control over it on the dataset creation.

All the tweets of the dataset are cleaned according to the following steps:

- Remove all single character words.
- Replace users, links and hashtags with the tags <NAME/>, <LINK/> and <HASHTAG/> respectively.
- Correct HTML characters to the right symbol. Ex: '&' to &.
- Change common abbreviations to the complete word or sentence. Ex: 'btw' to 'by the way'.
- Remove all punctuation marks.
- Split joined sentences for users they have been left out of characters and write a sentence like 'VeryGood'.
- Use a spell corrector to correct all mispelled words. The name of the tool used is *Hunspell*.

When this process is finished, all the dataset is cleaned and ready for the next step.

4.3 Data transformation

All our features are categorical values[8], but manage string data is very inefficient computationally speaking, so machine learning algorithms require to operate with numerical values. The process to convert categorical values to numerical values is called encoding. The classical string encoding consists in assigning a number to each word.

Given a dataset with the following sentences (table 3):

Text
How are you
It is awesome
You are a hero

Table 3: Dataset example.

The string encoding builds the table 4.

[8]Categorical values are variables that contain strings rather than numbers

Word	Index
how	0
are	1
you	2
it	3
is	4
awesome	5
a	6
hero	7

Table 4: Example of string encoding.

The string encoding has some issues because some algorithms assumes when the value is higher, better is the category. To solve this problem we can use the One Hot Encoding technique which consists in converting the value to a binary number, setting 1 in the column that has the value and 0 otherwise (see table 5).

how	are	you	it	is	awesome	a	hero
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

Table 5: Example of One hot encoding.

The last step is to assemble all the features (the encoded data) and its label (0 for positive, 1 for negative) into a vector to have the dataset ready to train.

4.4 Splitting the dataset

Before the model training, first we need to split the dataset in two parts: the training set and the test set.

- **Training set**: is the data used to train the model.
- **Test set**: is used after the model is trained to evaluate how the model performs with unseen data.

Never use as test set a data used on the model training, because the model learned how to classify this exact data and has a very high probability to guess the prediction, which may cause the results to be false.

Because the test set has the label, the framework can compare the predicted result with the label to throw some metrics about the success of the model.

Usually, the percentage of training set go from 70% to 90% and the rest goes to the test set, which is from 10% to 30%.

In this project, the percentage of the training set is 80% and the test set is 20% (figure 13).

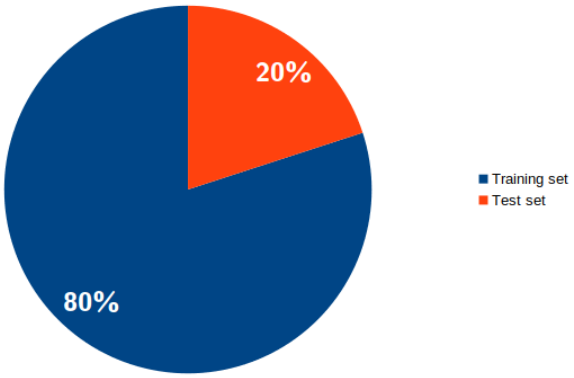


Figure 13: Train and test set percentage used for the project

5 Machine learning models

A machine learning model is the result of training some data using a machine learning algorithm with a specific configuration. After the model is fitted, it is able to get new data as input and predict the label of this data, explained in figure 14. This is what the model evaluation does, get the predictions of the test set (unseen labeled data) and compare if is equal to the label.

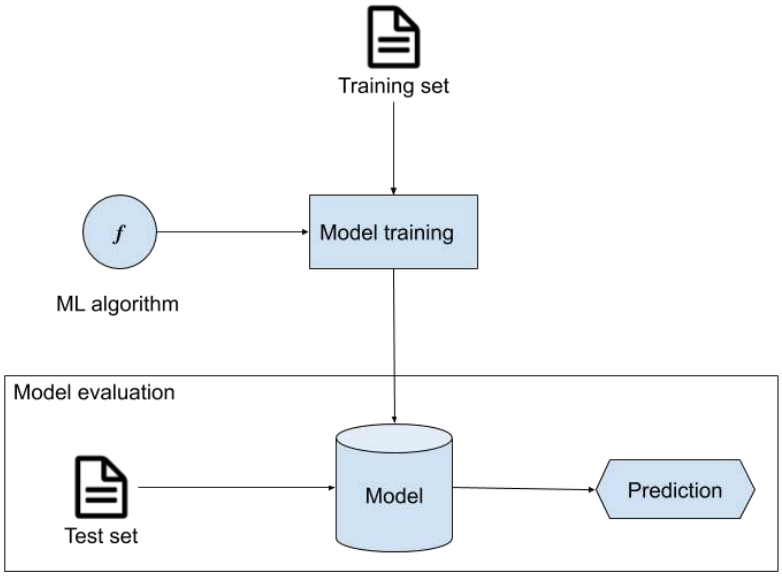


Figure 14: Machine learning model phases

5.1 Model training

Before training the model, first we need to define the configuration of the machine learning algorithm or technique. Each technique/algorithm has his own parameters, so we will need to define it for the Convolutional Neural Network (CNN), Logistic Regression, Naive Bayes and Support Vector Machine.

5.1.1 Convolutional Neural network

For the CNN there are some decisions to do to build the network. The network has the following properties:

- **Embedding layer**: The input layer. His purpose is to convert the input data into dense vectors of fixed size. We have a dictionary of a lot words, let's say n. To compute the weight of each feature we need a

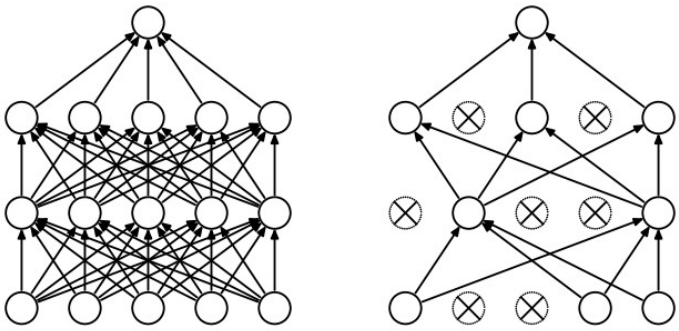
n-dimensional vector, which is very memory costly. The transformation to a dense vectors[9] helps to reduce the amount of dimensions to a fixed size. This size, introduces a new variable, the embedding size, and the model will be trained with a value of emmbedding size of 16, 32, 64, 128 and 256 to compare the results.

- **Neuron activation function**: The used function is ReLU [section 2.1.1], which changes the negative values to 0, $ReLU(x) = max(0, x)$. The number of neurons, called the number of filters, is a variable and will be trained with 16, 32, 64, 128 and 256.

- **Convolution layer**: Used to reduce the number of the vectors preserving his relations [section 2.1.3.1].

- **Pooling layer**: His job consists in reduceing the number of parameters [section 2.1.3.2]. The function used is Max-pooling, which takes the max. value of the region. The max-pooling has better performance and is better extracting the important features rather than the average-pooling, which extracts the features more smoothly, but may fail in some cases because all features are used for feature mapping which is a very generalist approach.

- **Dropout layer**: The dropout layer [13] consist in deactivating some random neurons during the training, as shown in 15. This technique forces the neurons connected to the disabled neuron to learn the same thing. The use of this layer is to prevent over-fitting[10] because can minimize the co-dependency between the neurons.



(a) Standard Neural Net (b) After applying dropout.

Figure 15: Standard and dropout layer comparison

- **Output layer**: The output layer uses the softmax function [2]. For the neural networks the labels mutually exclude themselves, if is posi-

[9] A dense vector is when most of the values in the vector are non zero.

[10] Over-fitting occurs when a model captures the noise of the data and minimize the ability to generalize.

tive cannot be negative or vice versa. This is not true for a sentiment detection, which can be neutral, in other words, 50% positive and 50% negative. The softmax algorithm calculates the probability for each class, determining what is the predominant class.

- **Optimization algorithm**: The optimization algorithm is Adam, seen on section 2.1.4. The optimizer tweaks the weights of the model to minimize the loss function, making the prediction more precise. There are some papers [9] comparing different optimization algorithms on a neural network demonstrating that Adam is the most effective algorithm, achieving the best results and being one of the most recommended choices, as we can see in figure 16.

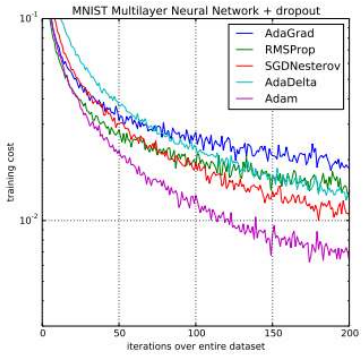


Figure 16: Optimization algorithms comparison

- **Epoch number**: An epoch is one iteration of all the dataset samples that has updated the internal model parameters. We set the number of the epochs to 8 because given past tests it has been verified that this is the best value to minimize the error of our model.

5.1.2 Spark MLlib

The classic Machine Learning algorithms has less parameters to decide than CNN. The Logistic Regression and Support Vector Machine takes two parameters, the max iterations and the L2 regularization parameter.

The max iterations define, like his name says, the maximum number of iterations made by the optimization algorithm which is an iterative method.

L2 regularization is used to avoid over-fitting and it computes the sum of the squared of the coefficients.

$$\lambda \cdot R(\beta) = \lambda \cdot \frac{1}{2}\|\beta\|_2^2 = \lambda \cdot \frac{1}{2}\sum_{i=0}^{n}\beta_i^2$$

Where λ is the parameter of our choice ($\lambda \in (0, \infty)$) to control the impact of the regularization. Too high value can cause under-fitting and too low value may can't avoid the over-fitting.

We will train and compare the models with the following values:

- **Max iterations**: From 10 to 100 in parts of 10 (i.e. 10, 20, 30, ...).
- **Regularization parameter**: From 0.1 to 1 in parts of 0.1 (i.e. 0.1, 0.2, 0.3, ...)

On Naive Bayes you can define the model type and the smoothing for multinomial model type.

MLlib supports two kind of event model types, multinomial and Bernoulli, which are the most used for classification.

On multinomial event model the distribution is parametrized by vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ for each class y, where n is the size of the vocabulary and θ_{yi} is the probability $P(x_i \mid y)$ of feature i appearing in a sample belonging to class y [12]. This model ignores the non-occurring feature.

The additive smoothing is the regularization parameter of Naive Bayes algorithm. This technique consists in not letting any probability to be zero, as could be the case of a feature that never happens in the training set, because on the multiplication operation the other probabilities are removed.

On the Bernoulli event model the features are the inputs described as a binary variables. The decision rule for the model is based on

$$P(x_i \mid y) = P(i \mid y)x_i + (1 - P(i \mid y))(1 - x_i)$$

penalizing the non-occurrence of the feature rather than multinomial.

On text classification it makes no sense to penalize the non-occurrence of a word and it's proved that multinomial model performs better for this kind of problem [10], specially with large datasets (figure 17).

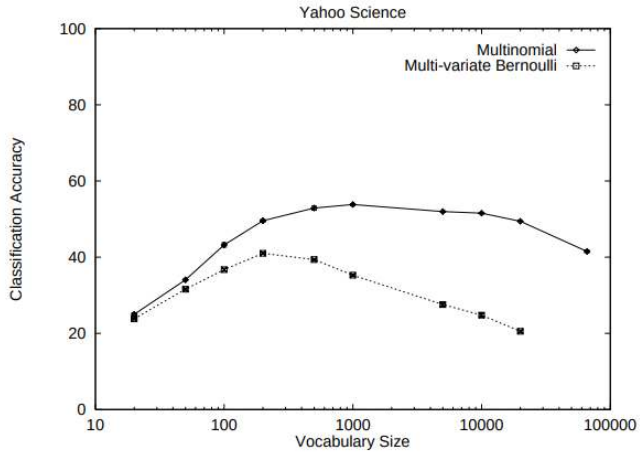


Figure 17: Multinomial and Bernoulli model types comparison

Then, we will use the multinomial model and different values for smoothing:

- **Model type**: Multinomial.
- **Smoothing**: From 0.1 to 1 in parts of 0.1 (i.e. 0.1, 0.2, 0.3, ...)

Now, we are able to start the model training on tensorflow framework:

```
model.fit(features_train, labels_train)
```

Listing 11: Tensorflow start training

Or Spark MLlib framework:

```
model = classifier.fit(train_dataframe)
```

Listing 12: Spark MLlib start training

5.2 Model evaluation

The model evaluation is the phase when the model is trained and the test set is used to throw some metrics to evaluate the goodness of the model. Because the test set is labeled, the model can make the prediction and compare if is equals to the label.

Each frameworks computes slightly different metrics except accuracy, that exists in both. The metrics returned by Tensorflow are accuracy and loss [13].

```
loss, accuracy = model.evaluate(test_data, test_labels)
```

Listing 13: Tensorflow test set evaluation

The metrics computed by Spark MLlib are accuracy, precision and recall [14].

```
predictions = model.transform(test_dataframe)

eval = MulticlassClassificationEvaluator()
accuracy = eval.evaluate(predictions,
                         {metricName: "accuracy"})
precision = eval.evaluate(predictions,
                          {metricName: "weightedPrecision"})
recall = eval.evaluate(predictions,
                       {metricName: "weightedRecall"})
```

Listing 14: Spark MLlib test set evaluation

To understand these metrics, first we need to know some concepts about the classes and the predictions of binary classification.

There are two classes, the positive and negative, which corresponds with 1 (positive sentiment) and 0 (negative sentiment) respectively. Depending on the class and its prediction can be categorized in different types:

- **True Positive**: The TP are the positive sentences predicted as positive, i.e. label = 1 and prediction = 1.
- **False Positive**: The FP are the positive sentences predicted as negative, i.e. label = 1 and prediction = 0.
- **True Negative**: The TN are the negative sentences predicted as negative, i.e. label = 0 and prediction = 0.
- **False Negative**: The FN are the negative sentences predicted as positive, i.e. label = 0 and prediction = 1.

5.2.1 Accuracy

The accuracy is the most important metric and the one will be used to compare the goodness of the model between Tensorflow and Spark MLlib.

The accuracy is the proportion of true results (both true positives and true negatives) among the total number of cases examined. The result we obtain is the percentage of success predictions of the test dataset.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Which is the same as:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

There is a problem with this metric, and this is why we need the support of the other metrics to validate if the model is working correctly.

For example, if we have a dataset with 100 entries, 95 are negative and the other 5 are positive. Imagine our model isn't working fine and for all cases predicts the negative class. The accuracy of this model will be 95%, a very good mark, but false, because with the opposite classes, 95 positive and 5 negative, the accuracy will be 5%. With the other metrics explained below, we can assure that the metric are not lying us.

5.2.2 Loss

The loss value implies how better or worse our model performs. The range of loss go from 0 to ∞, the loss is not a percentage. The lower the loss value, the better the model predictions, so we want to minimize the loss the maximum as possible.

$$Loss = -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

It works multiplying the logarithm of the predicted probability by the real class, which penalize a lot the false positive and false negative classifications.

If the real class is 1 the second part of the function will disappear and if the real class is 0, the first part of the function goes away because it multiplies the logarithm by zero.

For example, if the real class is 1 and the predicted class is 0.1, the loss function computes:

$$Loss = -(1 \cdot \log(0.1) + 0 \cdot \log(1 - 0.1)) = -log(0.1) = 1$$

If the real class is 1 and the prediction is 0.9, very close to the real class, the result is:

$$Loss = -(1 \cdot \log(0.9) + 0 \cdot \log(1 - 0.9)) = -log(0.9) = 0.0458$$

We can observe that when the prediction have bigger distance from the real class, the loss is bigger.

5.2.3 Precision

The precision metric is, for those predictions the model said as positive, how many of them are correct. With the precision we obtain the proportions of positive results that are true positive results. The ideal value, with a perfect test, is 1 (100%).

$$Precision = \frac{TP}{TP + FP}$$

5.2.4 Recall

The recall metric is, for those which are actually positive, how many of them the model can label correctly. In this metric we can see the efectivity of the model labeling the positive data.

$$Recall = \frac{TP}{TP + FN}$$

6 Results

In this section, we will analyze and compare the results obtained from different models. We will compare if the same model trained with the bigger dataset (1,306,478 registers) is an improvement or not over the smaller one (428,688 registers). We will select the parameter configurations with the best accuracy for each technique/algorithm to confront them.

6.1 Convolutional Neural Network

The two parameters of the CNN are the embedding size, which is the size of the dense vectors, and the number of filters, which defines the number of neurons of the network. For both parameters the values 16, 32, 64, 128 and 256 are used. The goal of this test is to identify how this two parameters impacts on the accuracy of the model.

Small dataset

The accuracy of the dataset with 428,688 registers, as it shows the figure 18, has the better result, a 77.3%, on the most high values, so we can observe that, in most cases, a high value of both parameters can improve the accuracy. The impact of the embedding size is bigger than the number of filters, because comparing a high size of embedding with low number of filters has better accuracy than low embedding size with high number of filters. Taking the best result, it can be extracted that out of every 100 predictions, 77 are labeled correctly.

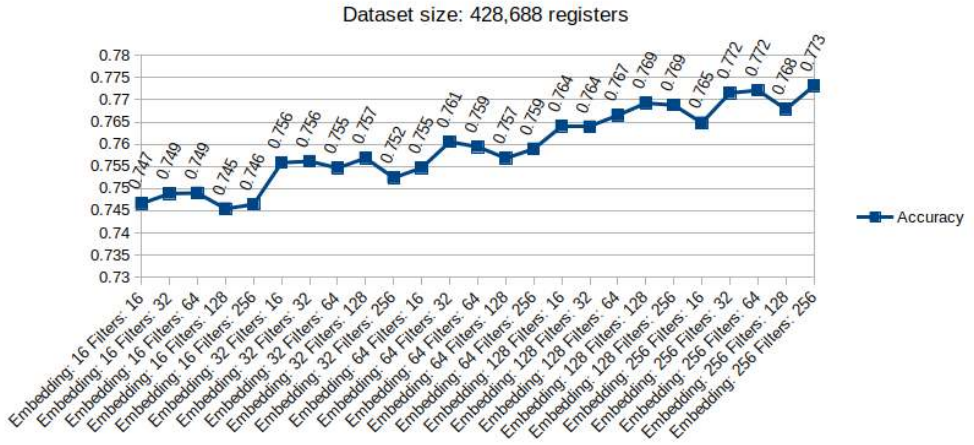


Figure 18: Chart of accuracy of the CNN with the 428,688 registers dataset.

The loss of the small dataset, which is depicted in figure 19, shows what is

expected, when loss is lower, better is the accuracy, reaching the minimum result (the best) on the best accuracy.

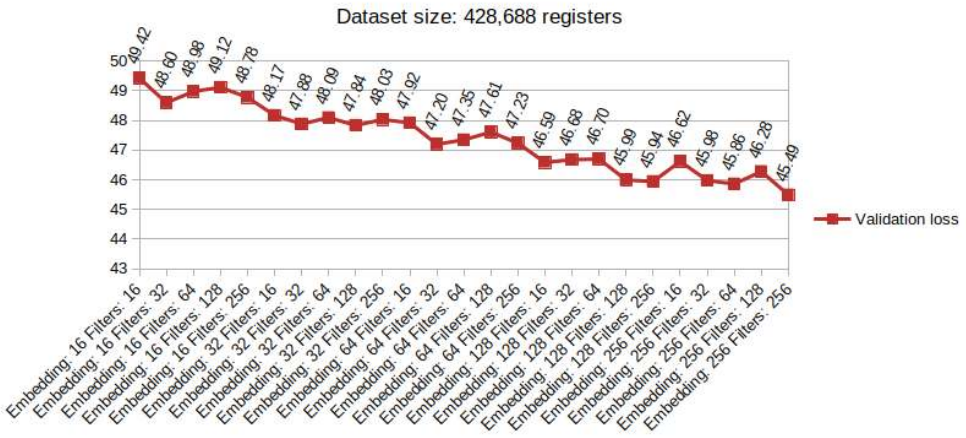


Figure 19: Chart of loss of the CNN with the 428,688 registers dataset.

Big dataset

The accuracy of the big dataset (1,306,478 registers), in figure 20, shows the same behaviour as the smaller one, when bigger is the embedding and filters, better is the accuracy. The maximum accuracy of the big dataset is a 76.3%, so it can predict correctly 76 out of 100 sentences. There is something that catch our attention, this best accuracy is a bit lower than the best accuracy of the smaller dataset, which is the proof that a larger dataset does not imply better accuracy. Anyways, the difference between both accuracies are very close, this can be because the data are random tweets, and this randomness can add better or worse data into the dataset.

The loss of the big dataset, as it shows the figure 21, has the same tendence as the small dataset, decreasing his loss when the accuracy increases. Comparing the values of the accuracy that matches the ones of small dataset, the loss are a little bit bigger, showing us that the small dataset model is performing better.

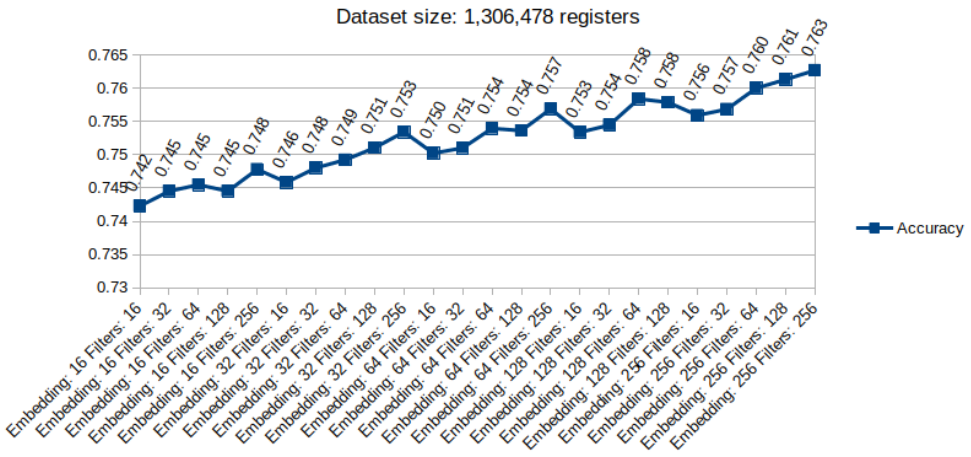


Figure 20: Chart of accuracy of the CNN with the 1,306,478 registers dataset.

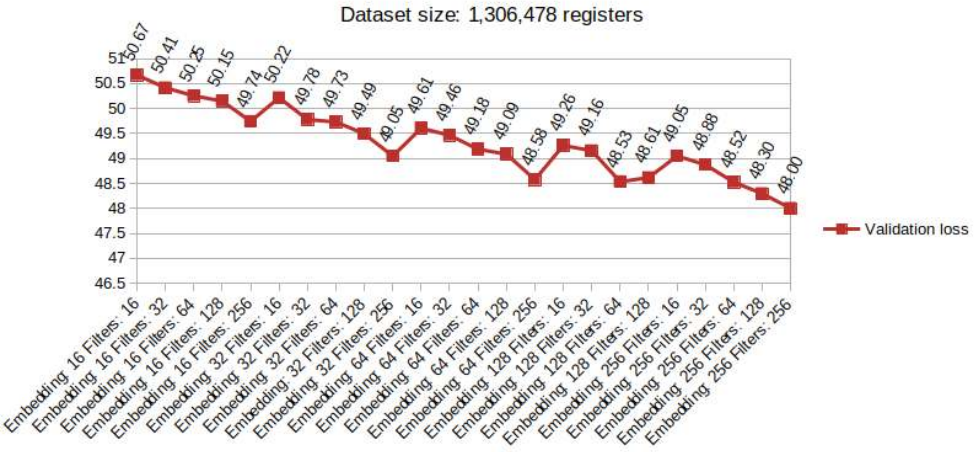


Figure 21: Chart of loss of the CNN with the 1,306,478 registers dataset.

The Tensorflow framework has a feature that supports GPU acceleration instead of CPU, on a graphic card with CUDA[11] cores, which is faster and has a big improvement on the training time. These tests have been carried out with the GPU Nvidia GeForce GTX 980 Ti graphic card with 2,816 CUDA cores and the CPU Intel Core i7-3770 @ 3.40GHz with 4 cores. The figure 22 shows that the small dataset training time with the CPU is 1 hour and 52 minutes and with the GPU go down to 28 minutes . The big dataset using the CPU it took 5 hours 42 minutes and with the GPU 1 hour 23 minutes. As we can see, there is a great improvement and it is worth to acquire a CUDA compatible graphic card if you have plans to train a big amount of models.

[11]CUDA is a technology developed by Nvidia that allow the processors of the GPU to run in parallel.

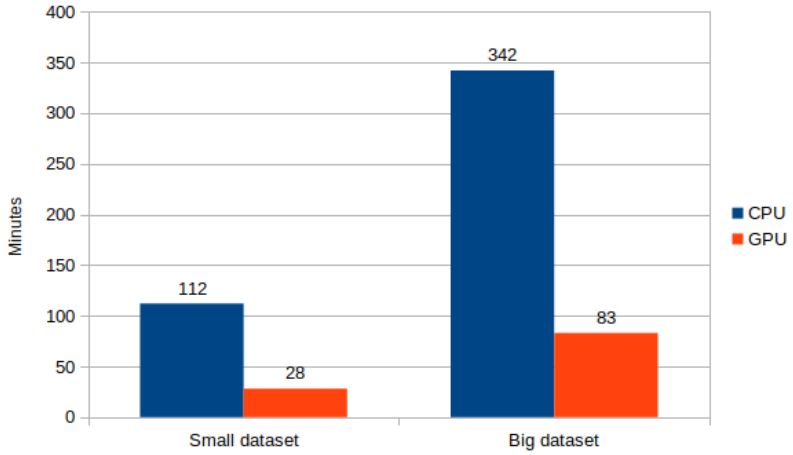


Figure 22: Difference of training time between CPU and GPU.

It is interesting to note that in a dataset approximately three times greater, it takes three times to do the training of the model, which indicates that the time scales in equal proportion to the size of the dataset.

6.2 Logistic Regression

The Logistic Regression algorithm has two parameters to play with, the number of iterations and the regularization parameter. Let's analyze the impact of this variables to the metrics of the model.

Small dataset

On the results of the small dataset (figure 23) we can observe a very pronounced pattern, when lower is the regularization parameter, better is the accuracy, being 0.1 and 0.2 those that have the better results. On the other hand, the number of iterations has a low incidence on the accuracy of the model but is slightly better when the value is lower, telling us that the optimization algorithm has enough with 10 iterations, and doing more iterations is unuseful or counterproductive. The model achieves the best accuracy with 10 iterations and 0.1 regularization with a 78.8% of accuracy. The 10 iterations and 0.2 regularization is the second one with a 78.7%.

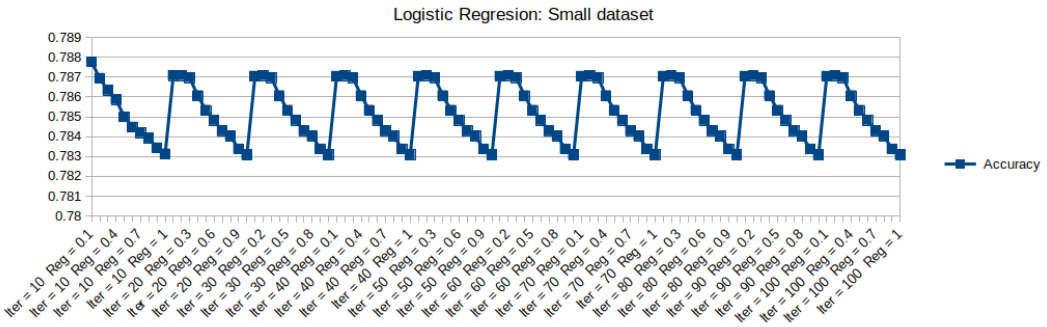


Figure 23: Accuracy of Logistic Regression on small dataset.

The precision of the small dataset, seen in figure 24, has the same shape as the accuracy, but gets the best value since 20 iterations and 0.2 or 0.3 regularization parameter. Given that with more than 10 iterations there is no improvement, we can save iterations and discard the bigger ones setting the best precision on 20 iterations and 0.2 or 0.3 RP with a 79%.

The recall of the small dataset (figure 24) has the exact same values as the accuracy. This means that our model has the same true rate labeling the positive examples and the negative examples which is very good.

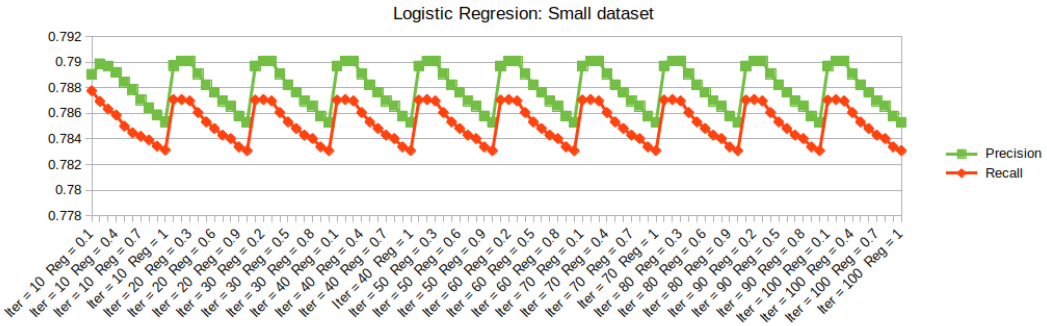


Figure 24: Precision and Recall of Logistic Regression on small dataset.

Two of the three metrics has the best result with 10 iterations and 0.1 PR and the other one on 20 and 0.2. If we take the 10/0.1 model we will lose 0.1% of precision, and for the model 20/0.2 we will lose 0.1% of accuracy and recall. Given that accuracy is the most important metric, we will prioritize maximize it and we will take the 10 iterations and 0.1 PR model as the best of the small dataset.

Big dataset

The same happens for the large dataset, which the results are shown in figures 25 and 26, the accuracy and the recall share the same results, the greater

the regularization parameter, the worse are the metrics and the number of iterations does not influence too much in the results, although it reaches the highest value in 80 iterations in all metrics. The best model of the big dataset is the 80 iterations and 0.1 regularization parameter, with an accuracy and recall of 74%, and a precision of 73.8%.

The difference between the results of the large and small dataset becomes more pronounced than in CNN, a 5% instead of 1%, showing that the classical algorithms penalize more the quality and the amount of data.

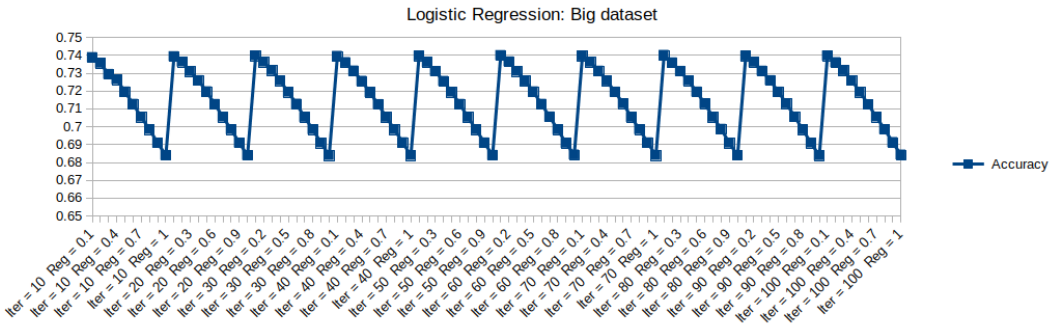


Figure 25: Accuracy of Logistic Regression on big dataset.

Figure 26: Precision and Recall of Logistic Regression on big dataset.

The training time of this and the following algorithms of the section are not compared because the training time is a few minutes (2 for the small dataset and 5 for the big dataset).

The training time of logistic regression algorithm (see figure 27) has a big difference if we compare with CNN. This reduction of time in comparison of neural networks is because the neural network is a more complex model with different layers to compute. The training time of the small dataset is 2.1 minutes and the big dataset is 5.4 minutes.

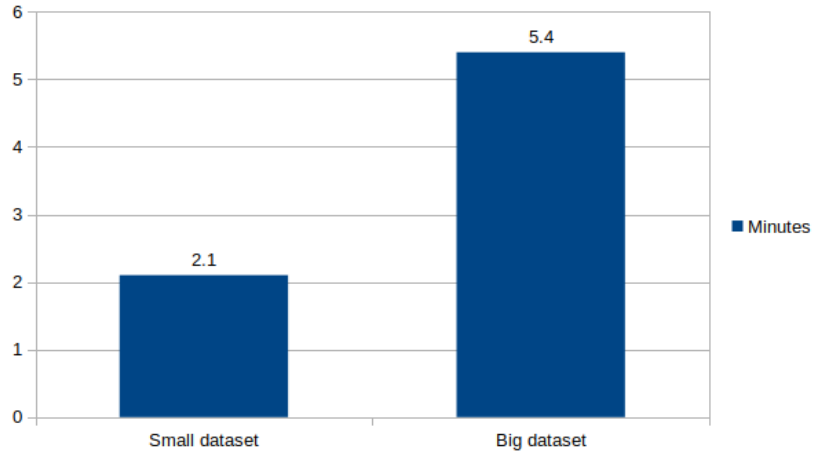


Figure 27: Logistic Regression training time.

6.3 Naive Bayes

The multinomial Naive Bayes has the smoothing parameter to adjust the probability to avoid 0 when a word of the dictionary is not occurring. Let's check how the different values for smoothing affects on the evaluation of the model.

Small dataset

The accuracy of the small dataset (figure 28) shows clearly when bigger is the smoothing, better is the precision. There is a regression between 0.3 and 0.6, but the accuracy reach the highest levels when smoothing is 1 with a 76.98%.

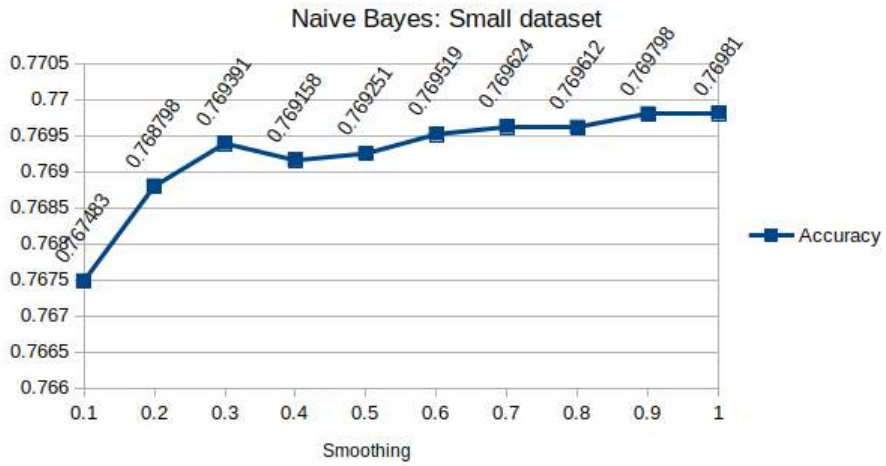


Figure 28: Accuracy of Naive Bayes on small dataset.

Observing the precision of the small dataset, as we can see in figure [29, we

realize that there isn't too much variance, although the 0.3 smoothing is the highest value, with a result of 78.0254%. Except the values 0.1, 0.2 and 0.5, the rest are too close values to be a decisive metric to choose the best model.

As we can see in Logistic regression, the recall (figure 29) behaves equal than accuracy showing the equitable balance between both classes. The best result is when smoothing is 1 with a recall of 76.98%.

Following the same criteria, the most balanced model between all the metrics, is when the smoothing value is 1.

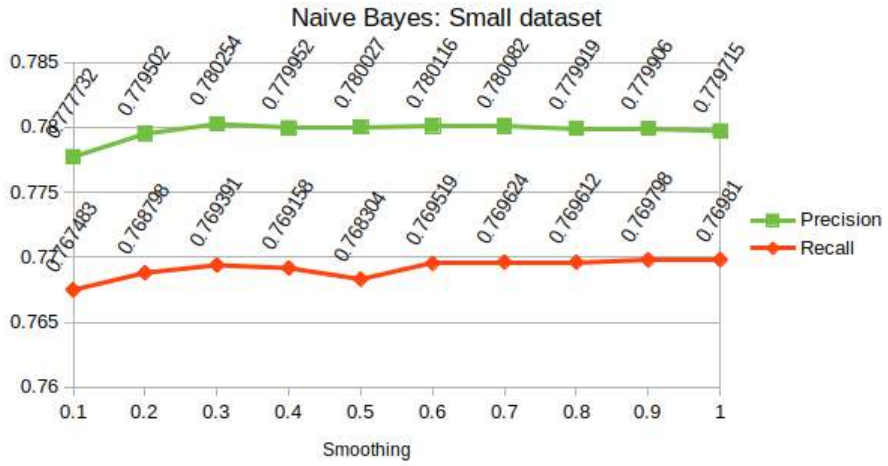


Figure 29: Precision and Recall of Naive Bayes on small dataset.

Big dataset

The results of the large dataset follows the same behaviour seen in the previous algorithm, reducing its accuracy by 3%, obtaining the best result with smoothing 1 with 73.8%.

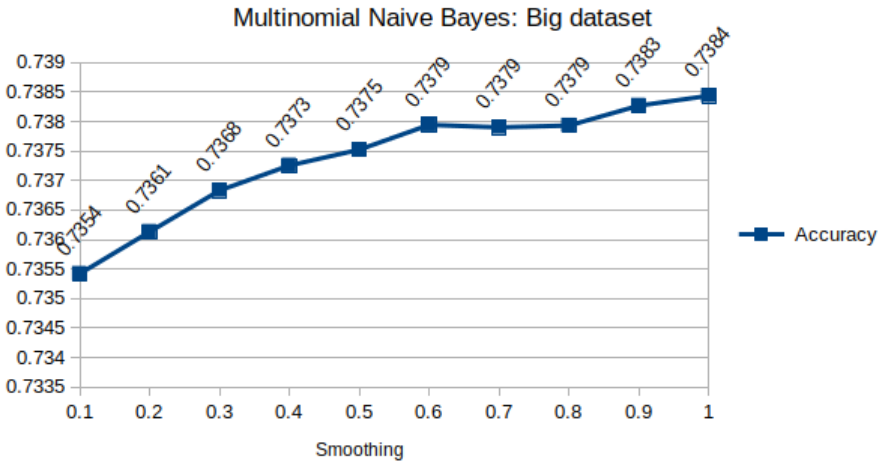


Figure 30: Accuracy of Naive Bayes on big dataset.

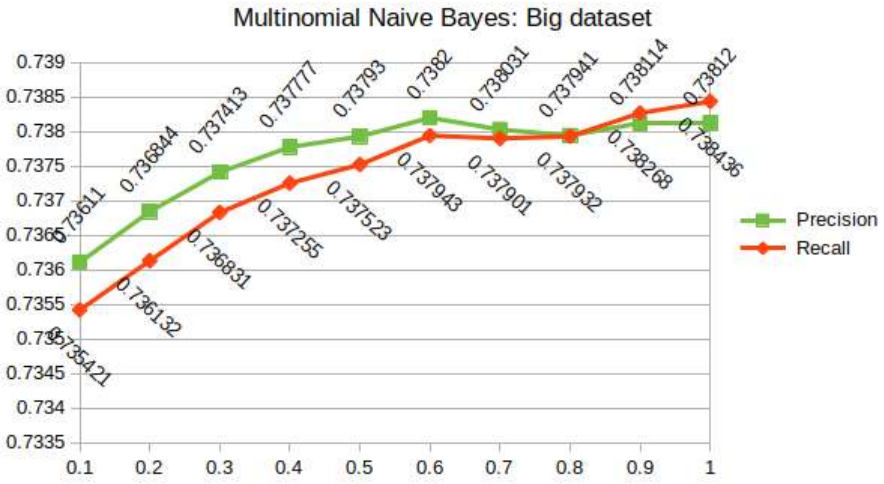


Figure 31: Precision and Recall of Naive Bayes on big dataset.

The training time of Naive Bayes algorithm (see figure 37) has the same behaviour as Logistic Regression, and it has a similar training time. The training time of the small dataset is 1.89 minutes and the big dataset is 5.1 minutes.

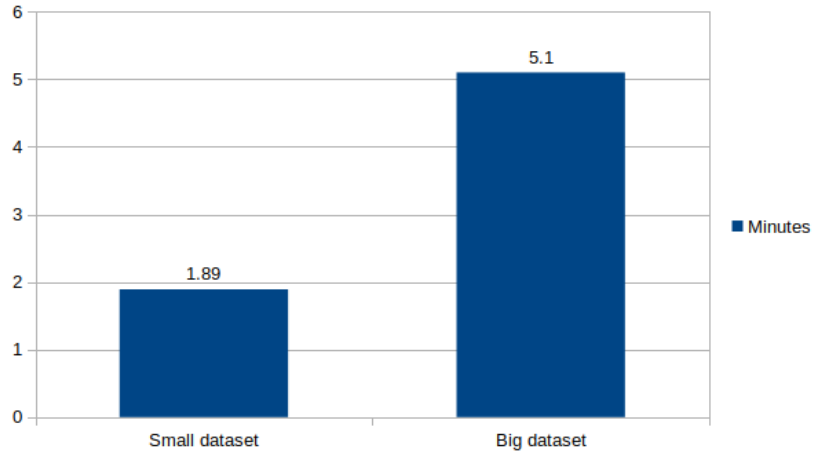


Figure 32: Naive Bayes training time.

6.4 Support Vector Machine

The Support Vector Machine algorithm supports the same parameters as Logistic Regression, the number of iterations and the regularization parameter. We train the different models with the same values as LR.

Small dataset

Examining the results of the small dataset, shown in figures 33 and 34, we observe the same behavior as the LR models, the regularization parameter has a greater impact on the metrics, making it worst when the value is higher. In the same way, the number of iterations has little effect on the result, although in general the metrics are slightly lower when the iterations are higher.

Taking into account that the best results of the accuracy and the recall are with 10 iterations and 0.1 of regularization parameter and the result of the precission with these parameters is very similar to the highest ones with 79.37%, the best model of the small dataset is the one with 10 iterations and 0.1 RP.

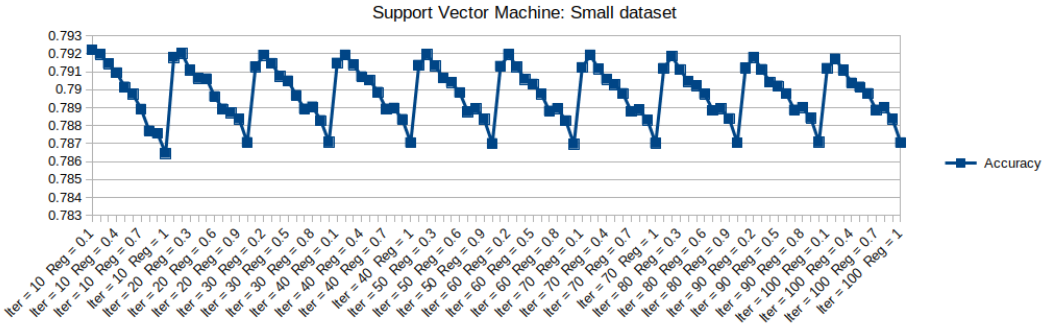


Figure 33: Accuracy of Support Vector Machine with small dataset.

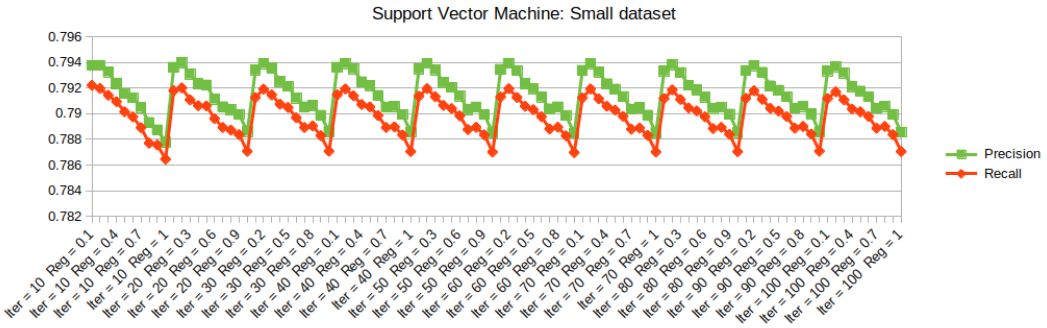


Figure 34: Precision and Recall of Support Vector Machine with small dataset.

Big dataset

On the side of the large dataset (figures 35 and 36), we can see how the regularization parameter have less incidence in the results when the number of iterations is low. Starting from 30 iterations, we have the same behavior that we have observed throughout this section, better metrics when the regularization parameter is lower. We again have a big difference between the large and small dataset, with a 5% difference, with an accuracy of 74.2% with 10 iterations and 0.1 of regularization parameter.

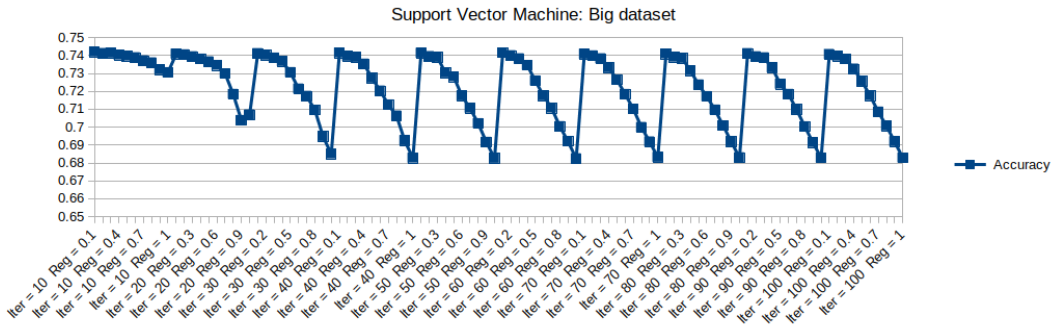


Figure 35: Accuracy of Support Vector Machine on big dataset.

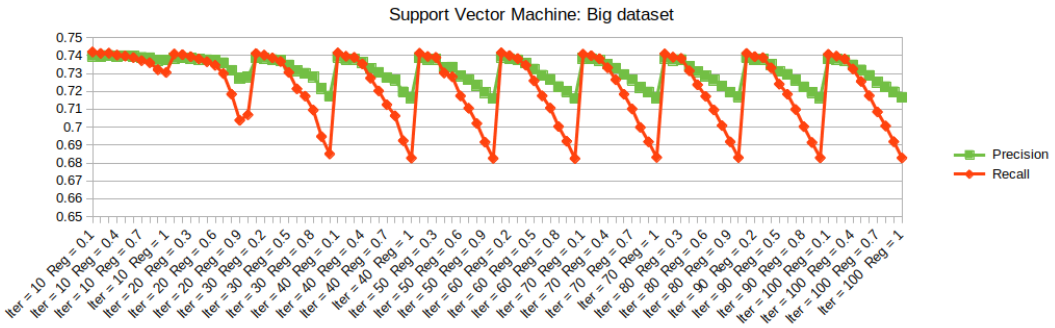


Figure 36: Precision and Recall of Support Vector Machine on big dataset.

The training time of Support Vetor Machine algorithm, which shows the figure 37, is a little bigger than Logistic Regression and Naive Bayes but it's also a few minutes. The training time of the small dataset is 3.4 minutes and the big dataset is 7.98 minutes.

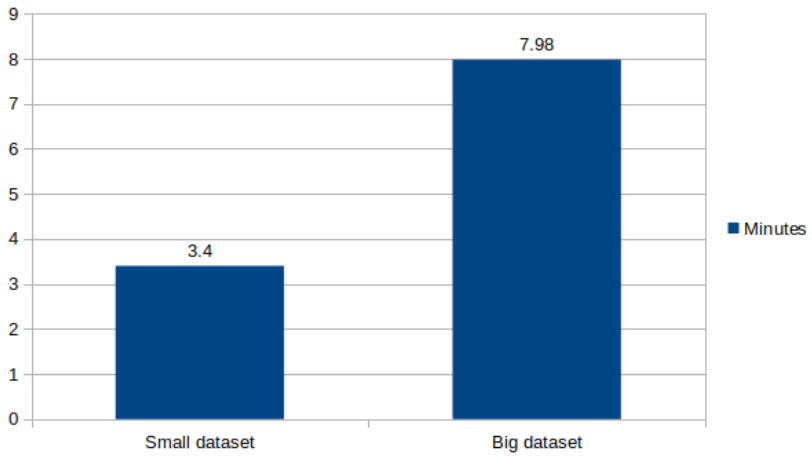


Figure 37: Support Vector Machine training time.

6.5 Comparison between them

Having identified the best model of each algorithm in each of the dataset sizes, it is time to do the comparison. As we have already verified with the other metrics the reliability of accuracy, in this section we will only compare this one.

Small dataset

Starting with the small dataset we can see in the figure 38 that all accuracies are very close having a 2% of variation between the best and the worst. This means that there are not too much difference between them and all this

models are perfectly valid to use them.

It is striking that CNN, which is theoretically the most powerful technique and should give the best results, ends in third place. This could be because the built network may not be the most optimal option and the addition or substration of layers should be evaluated in order to optimize the network.

The best accuracy is achieved by the Support Vector Machine with 79,2 correct predictions out of 100.

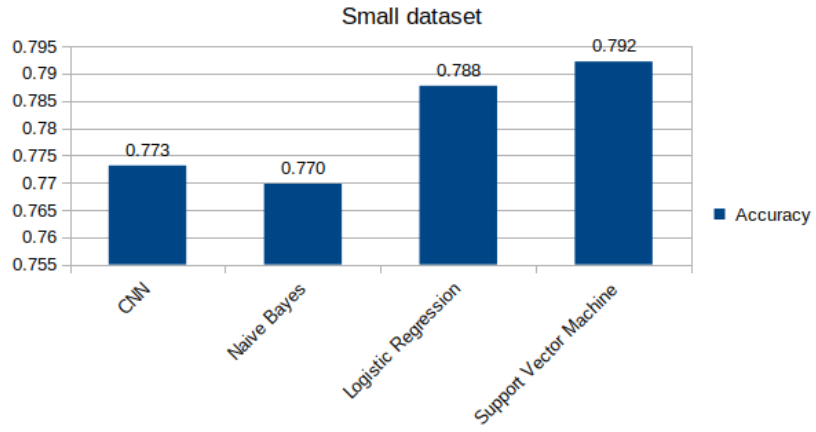


Figure 38: Comparison between the accuracy of all algorithms with small dataset.

All the metrics performs better with the small dataset, but can be interesting to compare the results of the big dataset to see the difference between the neural network and the classic algorithms when the dataset is bigger.

Big dataset

In the accuracy of the large dataset of figure 39 can be observed how the neuronal network is undoubtedly the winner. During this section, we have been able to see how CNN's technique is more stable and offers less difference when using a seemingly less-qualitative and bigger dataset. This can be a decisive factor to choose this technique when you have a changing and unforeseeable data source like the comments from a social network, ensuring less difference on predictions when you have a bad data extraction. For example, if we use sensor data, we always retrieve the expected data (after cleaning the data) making the model have better predictions, whereas with unforeseeable data such as user comments, it can be fashionable to write meaningless words for a period of time, that would add noise the model and reduce accuracy.

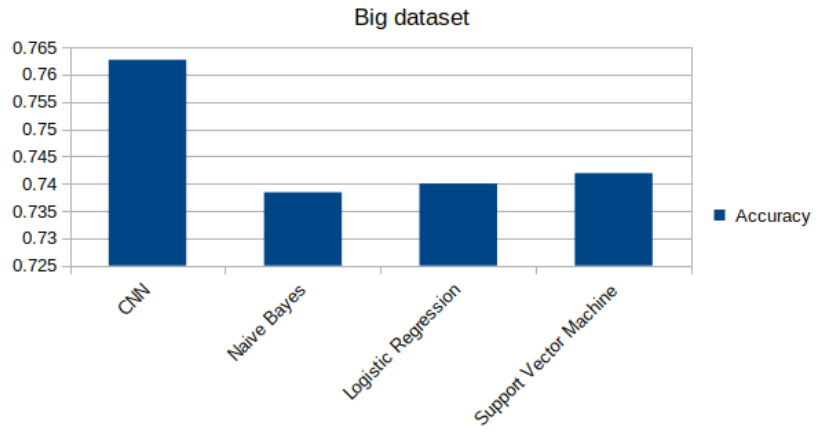


Figure 39: Comparison between the accuracy of all algorithms with big dataset.

7 Conclusion

In this section, I will explain the personal conclusions that I have extract from the realization of this project and the lines that remain open so that the project can continue growing and improving in the future.

7.1 Conclusions

The objective of this project was to deepen in all the steps of the creation of a ML model to classify the sentiment of a tweet.

First, we analyzed the different techniques or algorithms that could solve this type of problem. One with more advanced complexity, such as the Convolutional Neural Network, others, such as Naive Bayes, Logistic Regression and Support Vector Machine.

We have selected the framework that most fit for each of the algorithms, thus allowing the model to be created, trained and used efficiently.

We have trained each of the models with the different parameters that allowed the framework and we have analyzed different metrics to compare them and observe the impact they have on the result.

The best models have between 77% and 79% of accuracy, which is a good result for a complex problem, such as the analysis of sentiment in a real context. The algorithm that have the best accuracy score is the Support Vector Machine using the small dataset. Anyways, the model that have the lowest difference in accuracy between the two datasets is the Convolutional Neural Network, which is great for our type of data. In this way, the basis has been established in order to implement future improvements and optimize the processes in order to obtain a success rate greater than the obtained.

The extraction and cleaning of data has been a challenge, since when using data from real users, their way of writing and expressing themselves is conditioned by their demographic characteristics and personal experiences, making the task of cleaning all these data one of the most important to achieve the success of this project.

We have been able to verify how the quality of the data is more important than the quantity, making in some cases counterproductive to use a larger dataset, as it has happened to us.

The realization of this project has been very interesting for my formation

because has allowed me to obtain the basic knowledge in order to develop a machine learning project, a technique that is very important and popular in recent times and that we can find in many different areas of our daily life, such as in the selection of ads, spam detection, virus detection, voice recognition and so on.

7.2 Future works

Here are some of the possible points with which the project could be improved and expanded, in order to obtain a more accurate results or used used in other contexts.

- **More dataset sizes**: Due to time restrictions, this project has used two dataset sizes. In order to improve the analysis of the amount of data, more dataset sizes could be used to check their incidence in more detail.
- **Data extraction from other sources**: As a source of data for the project, the Twitter social network has been used. The data sources could be expanded using content from more platforms, like other social networks or forums, in order to extend the diversity of the data.
- **Improve the data cleaning**: There are some approaches that have been used to carry out data cleaning. Additionally, it could be added other decisions in this phase to facilitate the task of learning and building a more robust model.
- **Improve the Convolutional Neural Network**: As we have seen, the results of CNN have been lower than expected in comparison with other algorithms. We can try different types of configurations and number of layers for the network.
- **Auto retrain models with new data over the time**: Throughout this project, we have trained models with a static dataset. One of the benefits of machine learning is the ability to re-learn automatically as new data is extracted, making the model to improve over time.
- **Prediction for different contexts**: The aim of the project is to predict the text sentiment, but can be extended to other areas and other context like classify users according to their interests, for example political tendency prediction.

References

[1] Apache. Spark official webpage. `https://spark.apache.org`.

[2] Guillaume Bouchard. Efficient bounds for the softmax function and applications to approximate inference in hybrid models. *Xerox Research Center Europe*, 2008.

[3] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Kluwer Academic Publishers, Boston*, 1995.

[4] George Forman. An extensive empirical study of feature selection metrics for text classification. *Hewlett-Packard Labs*, 2002.

[5] Nikolas P. Galatasanos and Agelos K. Katasaggelos. Method for choosing the regularization parameter and estimating the noise variance in image restoration and their relation. 1992.

[6] Alfons Juan and Hermann Ney. Reversing and smoothing the multinomial naive bayes text classifier. *UPV and UoT Aachen*, 2000.

[7] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *University of Oxford*, 2014.

[8] Yoon Kim. Convolutional neural networks for sentence classification. *New York University*, 2014.

[9] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[10] Andrew McCallum and Kamal Nigam. A comparison of event models for naive bayes text classification. *Just Research and Carnegie Mellon University*, 1998.

[11] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *University of California, Berkeley*, 2002.

[12] Scikit-learn. Scikit-learn naive bayes documentation. `https://scikit-learn.org/stable/modules/naive_bayes.html`.

[13] Nitish Srivastava. Dropout: A simple way to prevent neural networks from overfitting. *Machine Learning Research*, 2014.

[14] Tensorflow. Tensorflow official webpage. `https://www.tensorflow.org`.