Contents

[Breadcrumb 1 (7/5/2022 6:50 PM) 2](#_Toc110525561)

[Context: 2](#_Toc110525562)

[Inferences/Thoughts/Conclusions: 2](#_Toc110525563)

[Breadcrumb 2 (7/5/2022 7:23 PM) 2](#_Toc110525564)

[Context: 2](#_Toc110525565)

[Inferences/Thoughts/Conclusions: 3](#_Toc110525566)

[Case 1) 3](#_Toc110525567)

[Case 2) 3](#_Toc110525568)

[Case 3) 3](#_Toc110525569)

[Case 4) 3](#_Toc110525570)

[Corrections: 3](#_Toc110525571)

[Breadcrumb 3 (7/6/2022 3:06 PM) 4](#_Toc110525572)

[Context: 4](#_Toc110525573)

[Inferences/Thoughts/Conclusions: 4](#_Toc110525574)

[Breadcrumb 4 (7/6/2022 3:24 PM) 4](#_Toc110525575)

[Context: 4](#_Toc110525576)

[Inferences/Thoughts/Conclusions: 4](#_Toc110525577)

[Breadcrumb 5 (7/6/2022 7:00 PM) 5](#_Toc110525578)

[Context: 5](#_Toc110525579)

[Inferences/Thoughts/Conclusions: 5](#_Toc110525580)

[Breadcrumb 6 (7/6/2022 7:15 PM) 6](#_Toc110525581)

[Context: 6](#_Toc110525582)

[Inferences/Thoughts/Conclusions: 6](#_Toc110525583)

[Naive Bayes Algorithm (7/11/2022 9:15 PM) 7](#_Toc110525584)

[Breadcrumb 7 (7/11/2022 10:21 PM) 8](#_Toc110525585)

[Context: 8](#_Toc110525586)

[Inferences/Thoughts/Conclusions: 8](#_Toc110525587)

[Breadcrumb 8 (7/12/2022 12:16 PM) 9](#_Toc110525588)

[Context: 9](#_Toc110525589)

[Inferences/Thoughts/Conclusions: 9](#_Toc110525590)

[Breadcrumb 9 (7/19/2022 6:26 AM) 11](#_Toc110525591)

[Context: 11](#_Toc110525592)

[Inferences/Thoughts/Conclusions: 11](#_Toc110525593)

[Naïve Bayes Mini-Project (7/25/2022 8:10 PM) 13](#_Toc110525594)

[Into the code! 16](#_Toc110525595)

[Support Vector Machine (SVM) 7/28/2022 7:14 PM 20](#_Toc110525596)

[Intro: 20](#_Toc110525597)

[Breadcrumb 9 (7/19/2022 6:26 AM) 21](#_Toc110525598)

[Context: 21](#_Toc110525599)

[Inferences/Thoughts/Conclusions: 21](#_Toc110525600)

[SVM Mini-Project 23](#_Toc110525601)

[References 26](#_Toc110525602)

# Breadcrumb 1 (7/5/2022 6:50 PM)

## Context:

There’s a group of animals which then is divided in two groups based on if they are “*acerous*” or not, to us this property’s meaning is unknown. Then we are asked to classify a specific animal into one of both groups.

## Inferences/Thoughts/Conclusions:

In this case we are dealing with a set of objects, or events which must classified according to a specific property, this property may not have meaning to a computer, but that’s why in the process of learning, there are examples provided to the algorithm where the answers are also given, this with the purpose of train somehow this algorithm and to allow it to take the right decision. This classification may be evaluated with a Boolean response “True” or “False” to determine whether a given object (“*animal*”) satisfies the condition (be or not be “*acerous*”).

So, it’s evident there’s a pattern which dictates this classification and the effectiveness of the algorithm to do it, so there’s an error associated and here it is where statistics and probability enter the game.

# Breadcrumb 2 (7/5/2022 7:23 PM)

## Context:

Four situations are presented: 1) recognize someone from a picture, 2) analyze bank data and flag if there’s any fraud, 3) recommend someone a new song based on preferences such as genre, tempo etc. and 4) cluster Udacity students based on their learning styles. From all of them we must decide which is suitable for supervised learning

## Inferences/Thoughts/Conclusions:

So, the appliance of machine learning isn’t categorical, the models and algorithms must vary according to the context presented. so according to the previous construction, we must decide.

### Case 1)

I consider the facial identification is a good example, because in here there’s a label assigned to a human face, and from what I know human faces are “*unique*” even if some of them are similar there´s a way to differentiate them. “Facial Recognition”.

### Case 2)

I doubt about this one, because detecting a fraud may not have a very specific pattern even though this is reflected in monetary terms, there’re several ways to commit it, and unlike case 1, the examples are not “easy to identify”. But deep down I think these kind of algorithms may involve a more complex pattern identification where the intention or method used is just neglected and the monetary mismatch is the focus, in which case we can say it’s an example of supervised learning.

### Case 3)

This one is suitable for supervise learning because there are specific patterns in music, such as tempo or instruments involved, or maybe they don’t need to specifically “*understand*” the music itself, but since the sharing of any music (in digital platforms or apps), it has defined properties which can then help algorithms to make this classification. Besides the features of music are already given so based on them it’s just matter of finding similar songs.

### Case 4)

I think this applies the same way the case 1 does, almost straight forward, because there’s a bunch of characteristics which define a “*learning style*”, so once identified, examples may be given for the algorithm to practice.[[1]](#footnote-1)

## Corrections:

Case 1): this is supervised because in the original statement there’s an album of tagged photos, so in here the faces or people in those photos already count with labels (a face corresponds to a name).

Case 2): this is unsupervised scenario because there’s no labeled data to identify what is a “*weird looking*” transaction. In here also applies the explanation of case 4)

Case 3): this is supervised

Case 4): this is unsupervised, because there’s no info about what type of learning styles exist, or about the students, besides it indicates clustering is not part of supervised learning. Even though this seems to be absolute, the reality is that there could be approaches to implement this kind of models in such scenarios.[[2]](#footnote-2)

# Breadcrumb 3 (7/6/2022 3:06 PM)

## Context:

The case of music recommendation is brought again, to illustrate how “supervised learning” works; There are two elements: features and labels.

## Inferences/Thoughts/Conclusions:

Part of this was already covered (showed: **above**), but instead of “***properties***” the convention uses “***features***” and instead the response is known as “***label***”. These labels are essentially the categories for classification.

# Breadcrumb 4 (7/6/2022 3:24 PM)

## Context:

If we want to analyze the given features, there must be a way to represent them, so computational operations send a result. For example, a graph with axes for each feature (Figure 1).

## Inferences/Thoughts/Conclusions:

There’s a way to translate abstract features into numeric values, in a way a spectrum is created taking the music example:



Does like

**Soaring**



**Light**

**Slow**

**Fast**



Don’t like



Figure 1: Example of features numeric-graph representation[[3]](#footnote-3)

This is a very simplified example, because in reality there could be more than two features, in which case the “graph” could have more than two axes, and as it gets complicated to represent, here it is where maybe tensors get involve representing these higher dimension’s objects (not sure). Besides the last graph showed a dataset from which each point represents a sample (song) and its feature’s components. Also, there’s an evident pattern, so assigning the label is easy, but **what would happen if the dataset was very disperse?** For example:

**Slow**



Does like

**Soaring**



**Light**

**Fast**

Don’t like

Figure 2: Same features but with different data sets

If I were to represent a dataset through a data structure, it would be in this case a matrix, where each column represents a feature component and in each line are the samples (this opens up the question: **how to organize data in matrices or multidimensional objects?**). Any way there’s an unclear label for the previous dataset.

# Breadcrumb 5 (7/6/2022 7:00 PM)

## Context:

This is an exercise that relates topics about “Scatter Plots” like the previous ones, but with terrain features instead of music ones.

## Inferences/Thoughts/Conclusions:

Scenarios (terrain) to graph (these are approximations and don’t count with numeric values):



**9**

**8**

**7**

**6**

**5**

**4**

**3**

**2**

**1**



**flat**

**7**

**5**

**6**

**8**

**9**



**4**

**1**

**2**

**3**



**smooth**

**steep**

**bad**

Figure 3: Terrain Scatter Plot. It seems in the real exercise they ask for a single example and not all of them

# Breadcrumb 6 (7/6/2022 7:15 PM)

## Context:

Imagine there’s a new Scatter Plot which counts with a dataset and a new sample (black dot Figure 4) is added to the group, if there are two labels available, based on antecedents (statistics) which label suits the best for this sample?

## Inferences/Thoughts/Conclusions:

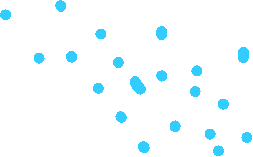
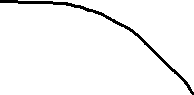


Figure 4:Scatter plot

In this scenario and by statistics it can be inferred that the new sample (black dot) must be blue. Again, this makes me think exactly how much statistics play a role in suggesting the identification (label assignment). But things get weird once a given new sample (purple dot) gets near the blurred boundary that separates both data groups. In this case statistics don’t help that much and the uncertainty increases.

So, one thing that helps to gain certainty is by defining a decision surface in a way we force the identification and therefore the new sample (purple dot) is now classified under the red label whatever that means. Nevertheless, I wonder if the boundary problem keeps existing even if it’s now reduced significantly.



This Decision surface’s boundary can be defined by a curve or a line depending on the scenario. The simplest approach is by using a linear boundary, but the slope must be the right one that best generalize the plotted data, this is, no all slopes satisfy separation of the classes. The main idea here is the fact the **decision area** can be made with python given the training points. Also, in here the first python library is introduced: [**Scikit-learn.**](https://scikit-learn.org/stable/modules/naive_bayes.html)**[[4]](#footnote-4)**

# Naive Bayes Algorithm (7/11/2022 9:15 PM)

Basically, what we have been covering is how to implement Naïve Bayes theorem to classify an event/object based on a current dataset which provides a criterium of how data behaves assuming the features are independent between themselves.[[5]](#footnote-5) The reason for using the term “*algorithm*” instead of “*theorem*” is because there’s a special class of models or algorithms used in machine learning that may be of different kind but are fundamentally based in the “Naïve Bayes theorem”. The part of “independency” refers to the mixed influence of one or more features for an event to occur; in a vague notion, it tries to look for the probability of an event to occur based on the mixing of two “independent” features that influence this last probability.

Through the naïve assumption of independency, the creation of models with a great behavior turns out to be easier. They do it by calculating the “ probability ” of an event to occur based on the probabilities of previous events. Mathematically the Naïve Bayes theorem expression is as follows (Roman, 2019):

A great way to understand this theorem is through this video: <https://www.youtube.com/watch?v=ayQglkLE36I&ab_channel=Serrano.Academy>

# Breadcrumb 7 (7/11/2022 10:21 PM)

## Context:

From the previous “*breadcrumbs*” the intention now is to pass those concepts to a python script and apply them.

## Inferences/Thoughts/Conclusions:

The first thing to make is to get ready the needed libraries and before the classifier’s creation there´s a general method to consider for that. Basically, first we must have two datasets, one for **“*training*”** the model and the second for **“*testing*”** it, which in the bigger picture means to have “*prepare data*”. Then the classifier is created using the proper libraries, in this case scikit-learn, more specifically one of the methods it has: “GaussianNB”. Finally, through the classifier we then make predictions with the testing set of data and if necessary, plot it or find a graphical/numerical way to represent its effectiveness. Simplified[[6]](#footnote-6):

An important observation here is the data preparation step because it’s essentially the most delicate part, and also unique due to its dependency of a context or situation that guides how data behaves or what are the interest to identify.

The library also contains functions that facilitate the lecture of data and curve fitting as well as determining the effectiveness or score with the “.score()” method.

A truly SMALL example is the following (in terms of structure):

#Context:

#From an Udacity course (Machine Learning) the scikit-learn library there are some basic tools to make classifiers

#(supervised learning) take into account features/attributes and assign a label.

import numpy as np

features = np.array([[-1,-1],[-2,-1],[-3,-2],[1,1],[2,1],[3,2]])

labels = np.array([1,1,1,2,2,2])

from sklearn.naive\_bayes import GaussianNB

classifer = GaussianNB()

classifer.fit(features,labels)

print(classifer.predict([[.8,-1]]))         #result: [1]

# Breadcrumb 8 (7/12/2022 12:16 PM)

## Context:

Now let’s going on deeper into the Naïve Bayes algorithm and present what’s about the Bayes Rule. In here the concepts of prior and posterior probabilities are consider as well as their integration as a whole. Also, there’s a new term or method to use which is probabilities’ “***standardization***”, and why not “***sensitivity***” and “***specificity***”.

7/18/2022 10:57 PM

To achieve that, a cancer example is used.

The statement goes as follows: There’s a type of cancer which affects 1% of the population; also, there’s a test that indicates whether a person has cancer and according to the given data, there’s a 90% probability it´s really positive (*sensitivity*) and there’s a 90% probability it’s negative if the person doesn’t have cancer (*specificity*).

Based on this the question comes up: **What is the probability of having cancer if the test turns out to be positive?** (could be phrased differently, but this is one way)

## Inferences/Thoughts/Conclusions:

A great way to illustrate the above example is by using Ben’s diagrams. So, the groups show probabilities of having or not cancer considering the test’s results:

No Cancer and Negative

Cancer 1

Cancer and Positive 90

No Cancer and Negative

All population

They way to address this problem involves somehow the sensitivity and the specificity; so, at a first glance it’s obvious the probability of having cancer and a positive test isn’t that high as it is 90%, but lower. In the other hand it’s discarded saying the probability is 1%, because it neglects the “test status”, so this last one must be part of the “calculation”. In this case the independent features which influence the posterior probability are “having cancer” and “positive test”. From the last reasoning we can state ; but which one could be a more precise value?

Just a small parenthesis here: Bayes rule (again) this is arithmetically a multiplication but is interpreted as an addition:

By trying to substitute into equation 2, we come up with the following:

Prior probability (having cancer)

Prior probability (**not** having cancer)

sensitivity (positive test evidence and having cancer)

(positive test evidence and **not** having cancer)[[7]](#footnote-7)

specificity (negative test evidence and **not** having cancer)

To compute the posterior probability, we need to make computations for two hypothesis, the first scenario having cancer if the test is positive as wells as not having cancer if the same test is positive:

These calculations are right but, the issue is that the addition of probabilities doesn’t add up the unit, so we must standardize (simplified: make the addition to add up 1)

The first step is to add both probabilities then each result must be divided by this sum:

The red terms represent the “standardized” posterior probability of one event or the another to occur, and the blue one is the answer for the question of the example which in percentage is 8.333%.

# Breadcrumb 9 (7/19/2022 6:26 AM)

## Context:

It’s mentioned Naïve Bayes algorithm is frequently used in text learning, this is, learning or applying identification to documents. Emails are included here so maybe we expect to deal with spam identification. Anyway, another example is used as guide, there are two different people who use to write emails, and these emails have specific words which use with different frequencies, the list of words is restricted to 3 (this list applies to both of them).

The frequencies and words are the following:

|  |  |  |  |
| --- | --- | --- | --- |
| Person | Words’ frequency | | |
| Love | Deal | Life |
| Chris | 0.1 | 0.8 | 0.1 |
| Sara | 0.5 | 0.2 | 0.3 |

Now if we have the following phrase “***Love Life***” the prior probability suggests that (given data)

and

And the guiding question is: Based on the statistics who intuitively would write that phrase?

Same question if the phrase is now “***Life Deal***”

## Inferences/Thoughts/Conclusions:

Answering the question, it’s apparent Sara would write it even though the prior probability suggest it could’ve been anyone because of the fifty-fifty chances, but from the table both words are used more often by Sara.

In the second scenario the uncertainty increases, but despite this increment due to the word deal, it’s more probably to have been written by Chris. This is a fair guess but what about really calculating those probabilities!

Prior probability (typing the phrase)

Prior probability (typing the phrase)

**Chris**

**Sara**

**Posterior Probabilities[[8]](#footnote-8)**

**Standardization:**

**Same example but with the phrase: “Love Deal”:**

**Chris**

**Sara**

**Posterior Probabilities[[9]](#footnote-9)**

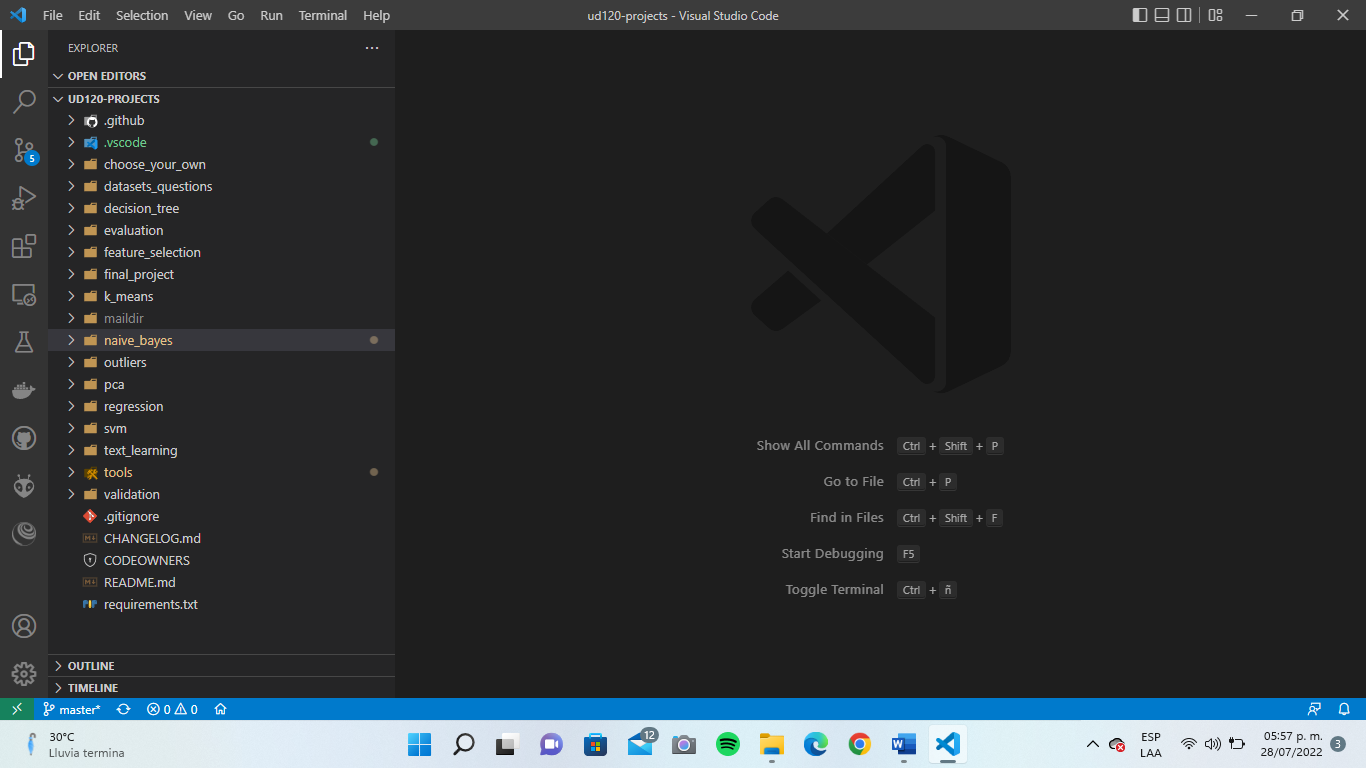
**Standardization:**

A very remarkable characteristic of this algorithm is that it doesn’t consider the order of the words (at least for these examples).

# Naïve Bayes Mini-Project (7/25/2022 8:10 PM)

Essentially the project consists of coding an email classifier, for this a GitHub repository was provided (<https://github.com/udacity/ud120-projects>)

Once the repository is downloaded, it looks like this:



Even though there are many files and folders, only a few of them were actually used, the first one made sure all the proper libraries were downloaded as well downloaded files with needed information for this project. There were two files, first one contains emails and the second one the authors (labeled with numbers, authors 0 and 1), and the algorithm must distinguish each email from its author according to the word content (like in previous examples).

The code of that file is this one:

#!/usr/bin/python3

print("Checking for nltk")

try:

    import nltk

except ImportError:

    print("You should install nltk before continuing")

print("Checking for numpy")

try:

    import numpy

except ImportError:

    print("You should install numpy before continuing")

print("Checking for scipy")

try:

    import scipy

except:

    print("You should install scipy before continuing")

print("Checking for sklearn")

try:

    import sklearn

except:

    print("You should install sklearn before continuing")

print("Downloading the Enron dataset (this may take a while)")

print("To check on progress, you can cd up one level, then execute <ls -lthr>")

print("Enron dataset should be last item on the list, along with its current size")

print("Download will complete at about 1.82 GB")

import requests

url = "https://www.cs.cmu.edu/~./enron/enron\_mail\_20150507.tar.gz"

filename = "../enron\_mail\_20150507.tar.gz"

with open(filename, "wb") as f:

    r = requests.get(url)

    f.write(r.content)

print("Download Complete!")

print("Unzipping Enron dataset (This may take a while)")

import tarfile

tfile = tarfile.open("../enron\_mail\_20150507.tar.gz")

tfile.extractall(".")

tfile.close()

print("You're ready to go!")

Despite the “*low*” relevance it has for the problem resolution, what’s interesting here is how to download documents from code as well as the good practice of using the “*try-except*” reserved words to debug errors when importing libraries.

Author’s file:

Graphical user interface, application

Description automatically generated with medium confidence

Email’s file:

Calendar

Description automatically generated with low confidence

## Into the code!

The complete code to solve the mini project is the following:

##!/usr/bin/python3

"""

    This is the code to accompany the Lesson 1 (Naive Bayes) mini-project.

    Use a Naive Bayes Classifier to identify emails by their authors

    authors and labels:

    Sara has label 0

    Chris has label 1

"""

import sys

from time import time

sys.path.append(r'D:\Desktop\Material de investigacion\ARTIFICIAL INTELLIGENCE\TF Práctico\Práctica\_TF#0\ud120-projects\tools')

from email\_preprocess import preprocess

### features\_train and features\_test are the features for the training

### and testing datasets, respectively

### labels\_train and labels\_test are the corresponding item labels

features\_train, features\_test, labels\_train, labels\_test = preprocess()

##############################################################

# Enter Your Code Here

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

t0 = time()                                                 #training time

classifier.fit(features\_train,labels\_train)

print("Training Time:", round(time()-t0, 3), "s")

t0 = time()                                                 #predicting time

print(classifier.predict(features\_test))

print("Testing Time:", round(time()-t0, 3), "s")

print("Accuracy",classifier.score(features\_test, labels\_test))

The relevant findings here are the two main issues take a huge part of the time to solve. The first one is about importing functions from another file’s directory (folder). So, to overcome this first issue, let say, when we import functions from files, there’s a repertory of directories from where to pick, for example the prebuilt non-external python functions are stored in such paths, or if the function is located in a different file but in the same folder, there’s no problem. But if the path is not located in such repertory, then the module cannot be imported.

That’s why we see the “**sys**” import, it calls the “system” functions, and one of them is precisely “***path***”. If were to print that, in terminal we’d get a list with available directories from where to make imports. Such list may be modified with the “***append***” command, and finally adding the file’s directory, once the list is updated, the imports can be done, but this effect is not global, instead it only applies when the current file is executed, so, if the same function is imported in a different file, where its directory hasn’t been appended, the import will fail (Bailey, s.f.).

The ”***r***” before the file’s location string tells the Python Interpreter to use the backslashes are to be interpreted as literal characters and not escape sequences.

The last error has a deep background which I won’t cover in this document, anyway it has something to do with “**line endings *LF and CRLF***”, the correction was obtained from stack overflow, and essentially it suggested to change the default type of line ending “CRLF” to “LF”, this was possible in vs Code, even though this may also be done through git. There’s a blog that explains this better: <https://www.aleksandrhovhannisyan.com/blog/crlf-vs-lf-normalizing-line-endings-in-git/>

After executing the file, the results in terminal are the following:

Text

Description automatically generated

From here we can conclude that the efficiency is relatively high getting a 97% in accuracy for identification, and from the times measured for each operation training (fit) and testing (predict), the one that takes greater time to complete is the training. Another thing to consider is the amount of data available, the higher the greater, and for data in reality it must be already clear, filtered and analyzed to be used, else the noise or incompatibility will disturb the capacity of prediction as well as the efficiency.

The code for the processing is this one (this was given from **Udacity**):

##!/usr/bin/python3

import joblib

import numpy

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.feature\_selection import SelectPercentile, f\_classif

def preprocess(words\_file =r"D:\Desktop\Material de investigacion\ARTIFICIAL INTELLIGENCE\TF Práctico\Práctica\_TF#0\ud120-projects\tools\word\_data.pkl", authors\_file=r"D:\Desktop\Material de investigacion\ARTIFICIAL INTELLIGENCE\TF Práctico\Práctica\_TF#0\ud120-projects\tools\email\_authors.pkl"):

    """

        this function takes a pre-made list of email texts (by default word\_data.pkl)

        and the corresponding authors (by default email\_authors.pkl) and performs

        a number of preprocessing steps:

            -- splits into training/testing sets (10% testing)

            -- vectorizes into tfidf matrix

            -- selects/keeps most helpful features

        after this, the feaures and labels are put into numpy arrays, which play nice with sklearn functions

        4 objects are returned:

            -- training/testing features

            -- training/testing labels

    """

    ### the words (features) and authors (labels), already largely preprocessed

    ### this preprocessing will be repeated in the text learning mini-project

    authors\_file\_handler = open(authors\_file, "rb")

    authors = joblib.load(authors\_file\_handler)

    words\_file\_handler = open(words\_file, "rb")

    word\_data = joblib.load(words\_file\_handler)

    ### test\_size is the percentage of events assigned to the test set

    ### (remainder go into training)

    features\_train, features\_test, labels\_train, labels\_test = train\_test\_split(word\_data, authors, test\_size=0.1, random\_state=42)

    ### text vectorization--go from strings to lists of numbers

    vectorizer = TfidfVectorizer(sublinear\_tf=True, max\_df=0.5, stop\_words='english')

    features\_train\_transformed = vectorizer.fit\_transform(features\_train)

    features\_test\_transformed  = vectorizer.transform(features\_test)

    ### feature selection, because text is super high dimensional and

    ### can be really computationally chewy as a result

    selector = SelectPercentile(f\_classif, percentile=10)

    selector.fit(features\_train\_transformed, labels\_train)

    features\_train\_transformed = selector.transform(features\_train\_transformed).toarray()

    features\_test\_transformed  = selector.transform(features\_test\_transformed).toarray()

    ### info on the data

    print("No. of Chris training emails : ", sum(labels\_train))

    print("No. of Sara training emails : ", len(labels\_train)-sum(labels\_train))

    return features\_train\_transformed, features\_test\_transformed, labels\_train, labels\_test

# Support Vector Machine (SVM) 7/28/2022 7:14 PM

## Intro:

Essentially, we deal again with graphs like the introductory ones, in the sense we try to separate a dataframe into two groups, and again a simple way to do it is with a line, anyway in here the distance between both groups must guarantee the maximum distance to nearest point, this is what’s call a “***margin***”. This because the condition of “separating” may be satisfied, but how good is at doing it could be ambiguous so that’s why the need of this term. And as it happened in previous examples cases are possible where data is dispersed; in such cases the separation will be done the best possible. Like this example:

Chart, scatter chart

Description automatically generated

The best separating line is the one with a positive slope (assuming the convention right-positive, up-positive).

Chart, scatter chart

Description automatically generated

In this scenario the best separating line is the one with negative slope (assuming the convention right-positive, up-positive)

# Breadcrumb 9 (7/19/2022 6:26 AM)

## Context:

As expected SVM is another method from the sklearn module, and what’s interesting about it is how easy the code was made to remain almost intact with the exception of using a different type of classifier; instead of Gaussian, SVM, while the rest of the structure/syntax stays the same (in terms of code).

## Inferences/Thoughts/Conclusions:

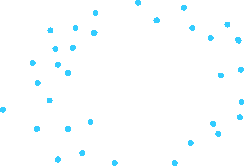
The fact there’re many models mean each one of them has advantages and disadvantages against particular conditions, so the problem will suggest which candidate will be the best, even though almost all of the work to solve it.

A picture containing graphical user interface

Description automatically generated

In the last image the importation of that algorithm is done but look at the SVM function’s parameter “*kernel*”, it is linear, a guess here is that maybe there are other types, such as polynomial or exponential, essentially other type of paths different from rectilinear. The documentation web page is added here: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>.

The guess was somewhat valid, in the current session, we deal with simplified problems which normally count with two independent features, but a new case is presented:



The best way to separate both groups is within a “circle”, this from an intuitive way of thinking, but this can also be represented in a different way, by adding a new feature “” which represents the radius of a circle of non-negative distance to the plane’s origin. The relevance of this radius emerges when we compare it for both groups; it’s smaller for red dots and greater for blue dots. So, representing this last projection on a new plane, we have the following:



(8/1/2022 10:28 PM)

In this new graph it’s obvious the separation, and it can be done with a regular line. This effect is due to the kernel used; it acts as a trick. What the kernel does is to create a new feature from the given features (because are not enough to make a valid separation), and then such new feature helps to create a valid separation. In the last example each graph represents a “space” with dimensions associated, which implies this trick can be used in higher dimensions (number of features). Kernels are predefined built-in functions, but customized ones may exist.

**SVM Parameters:**

* C=1.0
* kernel='rbf'
* degree=3
* gamma='scale'
* coef0=0.0
* shrinking=True
* probability=False
* tol=0.001
* cache\_size=200
* class\_weight=None
* verbose=False
* max\_iter=- 1
* decision\_function\_shape='ovr'
* break\_ties=False
* random\_state=None

So far, we have discussed the effect of the kernel for the boundary surface, but some other commonly used are the “***gamma***”, this parameter defines how far or close the influence of a single training example reaches, with low values meaning “*far*” and high numbers “*close*”. This influence relates the closer points of any group to the decision boundary, so when this influence is “***far***” this means the “***further points also influence***” the decision boundary’s shape, while a “***closer***” influence implies “***only closer points***” influence its shape.

In the other hand “***C***” represents the trades off correct classification of training examples against maximization of the decision function’s margin. This is, at ***high values***, the margin will be smaller (the said distance between points of different groups to the boundary surface line), therefore the algorithm is ***better*** at classifying ***points correctly***. In contrast for ***low values*,** we get a larger margin (smoothness), and the consequence is simpler decision with a ***worse correct points classification****.*

# SVM Mini-Project

Essentially, it’s the same case as in Naïve Bayes mini-project, there are small differences in code. The code used is the following:

# """

#     This is the code to accompany the Lesson 2 (SVM) mini-project.

#     Use a SVM to identify emails from the Enron corpus by their authors:

#     Sara has label 0

#     Chris has label 1

# """

import sys

from time import time

sys.path.append(r'D:\Desktop\Material de investigacion\ARTIFICIAL INTELLIGENCE\TF Práctico\Práctica\_TF#0\ud120-projects\tools')

from email\_preprocess import preprocess

### features\_train and features\_test are the features for the training

### and testing datasets, respectively

### labels\_train and labels\_test are the corresponding item labels

features\_train, features\_test, labels\_train, labels\_test = preprocess()

#features\_train = features\_train[:int(len(features\_train)/100)]    #less data

#labels\_train = labels\_train[:int(len(labels\_train)/100)]          #less data

#########################################################

### your code goes here ###

from sklearn.svm import SVC

classifier = SVC(kernel='linear')

t0 = time()                                                 #training time

classifier.fit(features\_train,labels\_train)

print("Training Time: ", round(time()-t0, 3), "s")

t0 = time()                                                 #predicting time

print(classifier.predict(features\_test))

print("Testing Time: ", round(time()-t0, 3), "s")

from sklearn.metrics import accuracy\_score

print("Accuracy: ", accuracy\_score(classifier.predict(features\_test), labels\_test))

At least the *structure* is the same, the change occurs in the function imported as well as the parameter called “*kernel*” as well as the accuracy, once executed the result in terminal is:

Text

Description automatically generated

Then when we add a line (less data) to reduce the dataset to only use the 1% to minimize the training and prediction times it also has an effect over the efficiency:

Text

Description automatically generated

Both scenarios use a linear kernel but now let’s see what happens when it’s set to “*rbf*” and using only 1% of the whole dataset:

Text

Description automatically generated

The accuracy increases so does it the computing speed compare with “*linear kernel*” (in thE Udacity site it gives an error message…)

Another modification is to change the “*C*” parameter:

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Maybe something is wrong because the result seems to be same for all cases…

(Create an example with invented data to play around)

# Decision Trees (8/4/2022 5:14 PM)

Check out this vid, it literally develops a decision tree as well as offers technical data, details, and references for more. Besides in one of those references, there’s a playlist of videos, all of them dealing with very useful and needed topics such as artificial intelligence, statistics, and probability.

# References

Bailey, C. (s.f.). *The Module Search Path*. Recuperado el 28 de july de 2022, de realpython.com: https://realpython.com/lessons/module-search-path/

Roman, V. (25 de april de 2019). *Algoritmos Naive Bayes: Fundamentos e Implementación*. Recuperado el 11 de july de 2022, de https://medium.com/datos-y-ciencia/algoritmos-naive-bayes-fudamentos-e-implementaci%C3%B3n-4bcb24b307f: https://medium.com/datos-y-ciencia/algoritmos-naive-bayes-fudamentos-e-implementaci%C3%B3n-4bcb24b307f

1. In reality there’s no need for typing deep responses in each case, but I guess they act as evidence there was a thinking process behind and no just random choices. [↑](#footnote-ref-1)
2. It seems those scenarios are compared to real examples [↑](#footnote-ref-2)
3. This example was taken from Udacity’s Machine Learning Course\* [↑](#footnote-ref-3)
4. This library was already installed (personally) [↑](#footnote-ref-4)
5. Even though learning more about probability and statistics, the tip here is to investigate deep down the theorems and concepts rather than dive into a wide book. That while a take a lot of time and topics won’t be covered. [↑](#footnote-ref-5)
6. A very important observation here is the fact of using **test datasets** because the first inputs help the algorithm to train and build the identification process; and to really evaluate the results, tests with unknown data helps to **determine the effectiveness** of the model. [↑](#footnote-ref-6)
7. From the Udacity course it showed the value doesn’t come from sensitivity but rather from the “opposite”, specificity. The first term is the conditioning feature. [↑](#footnote-ref-7)
8. The cases considered for the standardization are the ones in which both subjects type the phrase. As a result, the prior probability for both is 50% and the probabilities of typing the phrase according to the frequency they use some words, adds up a bit to this first probability. [↑](#footnote-ref-8)
9. The cases considered for the standardization are the ones in which both subjects type the phrase. As a result, the prior probability for both is 50% and the probabilities of typing the phrase according to the frequency they use some words, adds up a bit to this first probability. [↑](#footnote-ref-9)