Report Homework 1 – Machine Learning

Denise Landini

1938388

Summary

Chapter 1: Project Overview

In this homework the objective is to provide a solution for the function classification problem described in the seminar "Machine Learning and Security Search".

1.1 Dataset

I have two types of dataset:

- Dataset with duplicate contains 14397 functions;
- Dataset without duplicate contains 6073 functions.

The dataset contains 14397 functions. In the dataset each function belongs to one class (n. functions in the first dataset):

1. Encryption: 2724 functions;
2. String Manipulation: 3104 functions;
3. Sorting Functions: 4064 functions;
4. Math Functions: 4504 functions.

The structure of the dataset is a JSON file and in particular each line is a JSON object.

I decide to read this JSON file with a specific python library called pandas. This library allows me to convert a JSON string to pandas object.
First of all, I import pandas library in my program and then I use its function called *read_json* to read the data. This function returns a variable consisting of a DataFrame type that has four labels: ID, semantic, lista_asm, cfg.

Each item is a dictionary. The keys of the dictionary are:

- ID: unique id of each function;
- Semantic: the label of each function in {math, sort, encryption, string};
- Lista_asm: the linear list of assembly instruction of each function;
- Cfg: the control graph encoded as a network graph.

Chapter 2: Data pre-processing

Data pre-processing is an important step in the data mining process and may affect the way in which outcomes of the final data processing can be interpreted. Data processing can be done in different ways. In this homework I consider two different way to do the data pre-processing.

2.1 Method 1: TfidfVectorizer

I decide to use the Vectorization because it is a technique by which I can make my code execute fast. Furthermore, vectorization is important in Machine Learning because in Machine Learning there is the concept of an optimization algorithm that tries to reduce the error and compute to get the best parameters for machine learning model. So, by using a vectorized implementation in an optimization algorithm I can make the process of computation much faster than the unvectorized implementation.

One of the prerequisites for using vectorization is to import Numpy library that is a library for Python which adds support for large arrays and multidimensional arrays along with a large collection of high-level math functions to be able to operate efficiently on these data structures.

We have three different types of Vectorization:

1. HashingVectorizer: instead of storing the tokens as strings, the vectorizer applies the hashing trick to encode them as numerical indexes. The advantage is that it is very low memory scalable because is no need to store a vocabulary dictionary in memory. The disadvantage is that the features' names can no longer be retrieved, once vectorized.
2. CountVectorizer: it is used to transform a given text into a vector based on the frequency (count) of each word that occurs in the document.

3. TfidfVectorizer*:* it balances out the term frequency (how often the word appears in the document) with its inverse document frequency (how often the term appears across all document in the dataset).

So, we have that Count Vectorizer and TfidfVectorizer are similar.
I choose TfidfVectorizer instead of Count Vectorizer because it turns out better performance for this problem. In fact, with CountVectorizer I can have some words that will appear many times and their counts are not very significant in the coded vectors.

2.2 Method 2

I use a different way of pre-processing the data in dataset.
Since the dataset is made up of four classes that represent the type of program that we have in the dataset, I think it is reasonable:

- to create classes (one for each type of program);
- assign the instructions that are most used in this program.

In particular, I consider that:

- Encryption program:
  - Is composed by a lot of nested for and if. So, I consider the instructions that we always have in a condition as all the jump instructions, i.e. “jmp”, “je”, “jne”, “jg” and so on;
  - But in a condition or loop we can also find a lot of “cmp”;
  - They use “xor” a lot and bitwise operations, so I considered instructions like “and”, “or, “, “xor” … .
- Sorting program:
  - Usually is composed by one or two nested for and some helper function. In assembly we have that to consider an external function we have to use the instructions “call”, so I consider this instruction. But, also in this case I consider the jump instruction because we have conditions and cycles;
  - Uses compare and moves instructions. For this reason, I consider “cmp” and “mov”.
- Math program:
  - Obviously uses a lot of arithmetic operations. So, I consider all the instructions for the addition, subtraction, multiplication, division like “add”, “inc”, “dec”, “mul”, “imul”, “div”, “cdq”, “idiv” and so on;
  - Sometimes it does operations on vectors and matrices: it usually uses floating point instructions, special registers xmm*. For simplicity, I consider all the instructions that start with “F” (all the floating-point instructions start with “F”), because the floating point instructions are for example “fidiv”, “fxch” and so on.
- String Manipulation:
  - Uses a lot of comparisons and swap of memory locations. So, I consider “mov”, “lea”, “pop”, “push” and all the jump instructions because we have comparison.
- Loops:
  - I decide that all the instructions that I have not considered until now, are label of loops.

In this way I create a matrix in which I have:

- Each row that corresponds to one program;
- Each column represents how many times there are instructions of that type of program. In particular:
  - Column 0: how many times we have instruction of the encryption program;
  - Column 1: how many times we have instruction of the sorting program;
  - Column 2: how many times we have instruction of the math program;
  - Column 3: how many times we have instruction of the String program;
  - Column 4: how many times we have a loop label.

Example: the first row that I have by printing the matrix is: [[66, 79, 5, 77, 6], …]

It means that in the program 0 I have 66 instructions that refer to the encryption program, 79 instructions that refer to the sorting program, 5 instructions that refer to the math program, 77 instructions that refer to the string program, 6 labels that refer to the loop and so on for the other rows that represent other programs.

Chapter 3: Split Data

For both method 1 and method 2, I use the function *train_set_split* to split the dataset. The parameters that are passed to these functions are:

1. X_all that represent:
    - For the first method is the result of vectorization by TfidfVectorizer;
    - For the second method is the matrix that I have created.
2. y_all that take the labels;
3. Test size that allows me to decide the size of the data that has to be split as the test set. In this case, I choose to split the data for 20% test and 80% training (that is the general rule);
4. Random state: is the seed for the random number generator during the split

Chapter 4: Create models

I test different models such as:

1. Bernoulli Naïve Bayes Model;
2. Multinomial Naïve Bayes Model;
3. Random Forest classifier;
4. Gaussian Naïve Bayes.

I decided to use the same models for both the first method and the second method pre-processing. I don't use other models like Logistic Regression and Support Vector Machines (SVM) because these models are designed for binary classification and do not natively support more than two classes.

4.1 Bernoulli Naïve Bayes Model

Bernoulli is a Naïve Bayes classifier for multi-variate Bernoulli models. BernoulliNB works with binary random variable, so it has outcome 1 if word appears in the text, 0 otherwise. This model is popular for document classification tasks, in which term occurrence features are used rather than term frequencies.

4.2 Multinomial Naïve Bayes Model

MultinominalNB implements the Naïve Bayes algorithm for multinomially distributed data, and is used in text classification, where the data are represented as word vector counts.

4.3 Random Forest classifier

Random forest is an algorithm based on Decision Trees. It ensemble method that generates a set of decision trees with some random criteria and integrates their values into a final result. I decide to use Random forest instead of Decision tree because is less sensitive to overfitting.
To use this method, I import RandomForestClassifier from sklearn.ensaemble.

4.4 Gaussian Naïve Bayes Model

In Gaussian Naïve Bayes we have continuous values associated with each feature that are assumed to be distributed according to a Gaussian distribution.
To use this method I import GaussianNB from sklearn.naive.bayes.

Chapter 5: Evaluation

For both method 1 and method 2 I choose to show the data that I obtained with:

- Confusion matrix;
- Classification report.

5.1 Metrics

For each class in the dataset, I compute:

- Precision;
- Recall;
- F1-score.

5.1.1 Accuracy

Accuracy is one metric for evaluating classification models. Accuracy can show us if a model is being trained correctly and how it may perform generally. However, in some cases, accuracy only is not enough to assess the performance of a classification method. In fact, it does not do well when we have an unbalanced data set.

The accuracy is given by:

$$Accuracy = \frac{number\ of\ correct\ prediction}{total\ number\ of\ predictions}$$

For binary classification, accuracy can also be calculated in terms of positives and negatives as follow:

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

Where:
TP = True Positive
TN = True Negative
FP = False Positive
FN = False Negative
This value of accuracy is the value that I find in confusion matrix.

5.1.2 Precision

Precision can avoid false positives (1 if FP = 0)

$$Precision = \frac{|true\ positives|}{|predicted\ positives|} = \frac{TP}{TP + FP}$$

5.1.3 Recall

Recall can avoid false negatives (1 if FN = 0)

$$Recall = \frac{|true\ positives|}{|real\ positives|} = \frac{TP}{TP + FN}$$

5.1.4 F1-score

F1-score is the best metrics because it is a measure of the accuracy of a test that takes into account precision and recall of the test.

$$F1\ score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

5.2 Confusion Matrix

Confusion matrix is a specific table that allows visualization of the performance of an algorithm. In a classification problem with many classes, we can compute how many times an instance of class Ci is classified in class Cj. So, each row of the matrix represents the instances in a predicted class while each column represents

the instances in an actual class and vice-versa.
The main diagonal contains accuracy for each class. All the other cells of the matrix contain the errors.

I decide to use confusion matrix to represent the performance of first and the second method used for the pre-processing of the data, because in this way it is possible to see the classes that are more confused. I represent this matrix with color-maps.
To use this representation I import confusion_matrix from sklearn.metrics

5.3 Classification Report

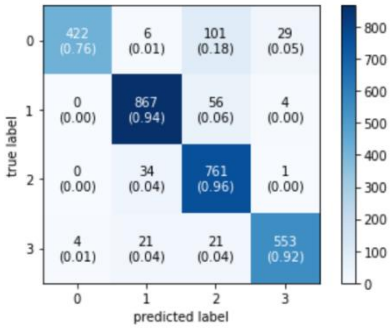
I think is also useful to represent the performances with the classification report, because it allows me to see very precisely, not only the accuracy value of the model used, but also Precision, Recall, F1-score. More specifically, with this representation I can see for class (Encryption, String Manipulation, Math, Sorting) precision, recall and F1- score.
The Support column indicates the number of examples of each class.
Furthermore, for each metrics (precision, recall, F1-score) there is the macro average, and weighted average.
To use this representation, I import classification_report from sklearn.metrics.

Chapter 6: Comparisons

6.1 Comparison between the first and the second method used for data pre-processing

I decide to compare the performances between the first method and the second method. So, I want to see how performance changes between one model and another taking into account each model that I have used.

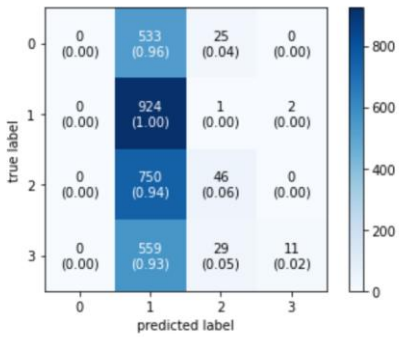
6.1.1 Bernoulli



a) Confusion matrix using the first method

	precision	recall	f1-score	support
encryption	0.99	0.76	0.86	558
math	0.93	0.94	0.93	927
sort	0.81	0.96	0.88	796
string	0.94	0.92	0.93	599
accuracy			0.90	2880
macro avg	0.92	0.89	0.90	2880
weighted avg	0.91	0.90	0.90	2880

b) Classification Report using the first method



c) Confusion matrix using the second method

	precision	recall	f1-score	support
encryption	0.00	0.00	0.00	558
math	0.33	1.00	0.50	927
sort	0.46	0.06	0.10	796
string	0.85	0.02	0.04	599
accuracy			0.34	2880
macro avg	0.41	0.27	0.16	2880
weighted avg	0.41	0.34	0.20	2880

d) Classification Report using the second method

Using the first method, I can see that all values are good enough, so this method works well, in fact I have an high accuracy.

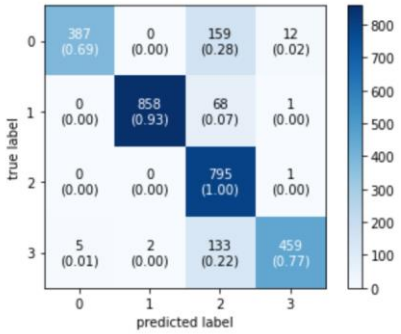
Using the second method, I can see that:

- The functions in encryption class are never found, so we have precision, recall, and F1-score equal to zero;
- I find many functions in math class; in fact, I can see the recall equal to one and I have the highest F1-score among the four classes considered;
- In Sorting class, I found functions but not a lot;
- In String class, I have the highest precision, but I have also the lowest recall among the classes.

So, almost all functions are classified as math functions.
As shown in figure a) and c) I can already notice a problem in using the second method, because I have the samples only in the second column instead of the main diagonal. (I remember that in the main diagonal I have the value of accuracy, and the other cell represent the errors).

In conclusion, how I can see considering figure b) and figure d) is that among the first method, which uses Tfidf vectorizer for pre-processing data, the accuracy is much higher than what I can have using the second method. So, in Bernoulli method is better use the Tfidf Vectorizer instead of the second method to pre-process data.

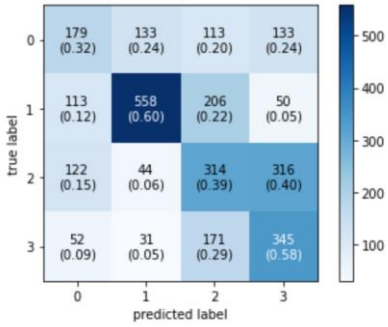
6.1.2 Multinomial



e) Confusion matrix using the first method

	precision	recall	f1-score	support
encryption	0.99	0.69	0.81	558
math	1.00	0.93	0.96	927
sort	0.69	1.00	0.81	796
string	0.97	0.77	0.86	599
accuracy			0.87	2880
macro avg	0.91	0.85	0.86	2880
weighted avg	0.90	0.87	0.87	2880

f) Classification Report using the first method



g) Confusion matrix using the second method

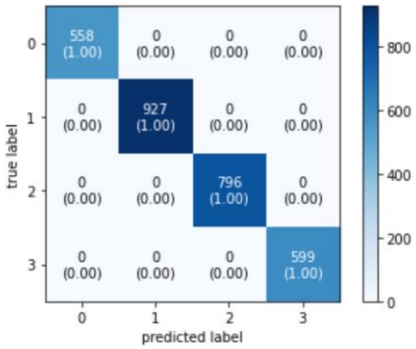
	precision	recall	f1-score	support
encryption	0.38	0.32	0.35	558
math	0.73	0.60	0.66	927
sort	0.39	0.39	0.39	796
string	0.41	0.58	0.48	599
accuracy			0.48	2880
macro avg	0.48	0.47	0.47	2880
weighted avg	0.50	0.48	0.49	2880

h) Classification Report using the second method

Using the first method, the accuracy remained about the same as I had using Bernoulli and I can see in figure f) that all values are good, and the accuracy also is good.

Using the second method, the accuracy is higher than I had using Bernoulli, but it is still less than the accuracy that I have obtained using TfidfVectorizer. This model does not work well with the second method. I can see in figure h) the value that are the best F1-score is math class, but for the other class I have value that are low.

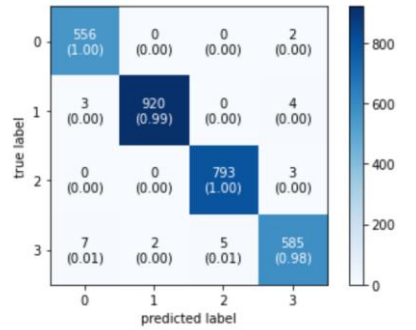
6.1.3 Random Forest



i) Confusion matrix using the first method

	precision	recall	f1-score	support
encryption	1.00	1.00	1.00	558
math	1.00	1.00	1.00	927
sort	1.00	1.00	1.00	796
string	1.00	1.00	1.00	599
accuracy			1.00	2880
macro avg	1.00	1.00	1.00	2880
weighted avg	1.00	1.00	1.00	2880

j) Classification Report using the first method



k) Confusion matrix using the second method

	precision	recall	f1-score	support
encryption	0.98	1.00	0.99	558
math	1.00	0.99	1.00	927
sort	0.99	1.00	0.99	796
string	0.98	0.98	0.98	599
accuracy			0.99	2880
macro avg	0.99	0.99	0.99	2880
weighted avg	0.99	0.99	0.99	2880

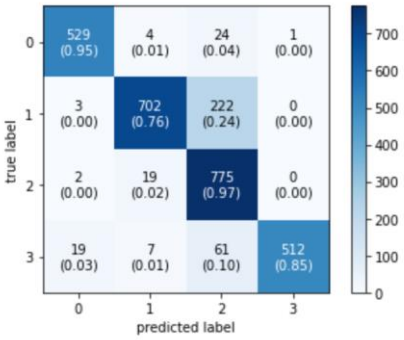
l) Classification Report using the second method

With Random Forest, we get high performance with both the first and the second method.
I can also see this result visually looking at figure k) only by looking at the confusion matrix which all the samples are on the main diagonal.

Compared to Bernoulli and multinomial classifier, when I do the pre-processing of the data with the second method, is much better to use the random forest as a model.

In the end for Random Forest is indifferent to choose one or the other method.

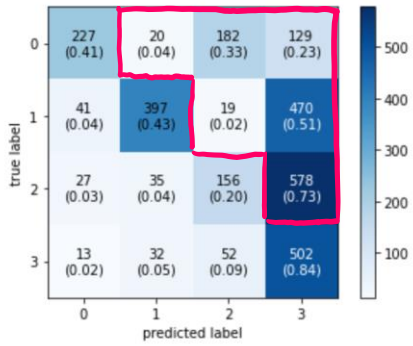
6.1.4 Gaussian



m) Confusion matrix using the first method

	precision	recall	f1-score	support
encryption	0.96	0.95	0.95	558
math	0.96	0.76	0.85	927
sort	0.72	0.97	0.83	796
string	1.00	0.85	0.92	599
accuracy			0.87	2880
macro avg	0.91	0.88	0.89	2880
weighted avg	0.90	0.87	0.88	2880

n) Classification Report using the first method



o) Confusion matrix using the second method

	precision	recall	f1-score	support
encryption	0.74	0.41	0.52	558
math	0.82	0.43	0.56	927
sort	0.38	0.20	0.26	796
string	0.30	0.84	0.44	599
accuracy			0.45	2880
macro avg	0.56	0.47	0.45	2880
weighted avg	0.57	0.45	0.45	2880

p) Classification matrix using the second method

Using the first method, I can see that the value are high, in particular I obtain an high accuracy.

I can see in figure o) that by using the second method, I have a problem in the upper triangular part of the matrix, but in the low traingular part of the matrix the value are similar to those in the figure m).

Looking at figure p) and n) I can notice that the accuracy obtainted using the first method is higher than the accuray obtained using the second method.

But, for the second method I can also say that I have an accucuray better than the accuray that I have by using Bernoulli model, but worse than the accuray that I have by using Random Forest model and Multinomial model (that is too similir).

6.1.1.0 Summury of the performances given by the first and the secondo model

	First method		Second method	
Model	Avarage F1-score	Accuracy	Avarage F1-score	Accuracy
Bernoulli	0.90	0.90	0.16	0.34
Multinomial	0.86	0.87	0.47	0.48
Random Forest	1.00	1.00	0.99	0.99
Gaussian	0.89	0.87	0.45	0.45
Avarage	0.91	0.91	0.52	0.57

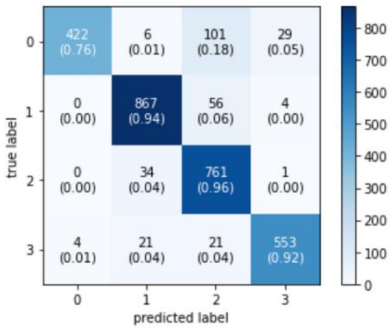
In this table I have also chosen to compute the average of the F1-score value obtianed from the models considered and I can conclude that with TfidfVectorizer method I get better performances than the second method. The same goes for the accuracy. I think that I obtained this results because using the second method of pre-processing, I lost some useful infromation.

6.2 Comparison of the performances of the first method using dataset with duplicates and dataset without duplicates

I analyze both first and second method with 4 models and in the end the first method has better performances in each model than the second.

So, I choose to analyze the performance of the first method using the first dataset that conatins the duplicates and then the dataset that does not contain the duplicates.

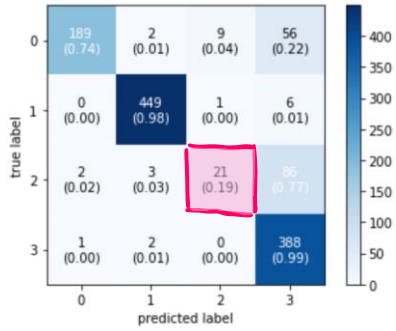
6.2.1 Bernoulli



a) Confusion matrix using dataset with duplicates

	precision	recall	f1-score	support
encryption	0.99	0.76	0.86	558
math	0.93	0.94	0.93	927
sort	0.81	0.96	0.88	796
string	0.94	0.92	0.93	599
accuracy			0.90	2880
macro avg	0.92	0.89	0.90	2880
weighted avg	0.91	0.90	0.90	2880

b) Classification Report using dataset with duplicates



a1) Confusion matrix using dataset without duplicates

	precision	recall	f1-score	support
encryption	0.98	0.74	0.84	256
math	0.98	0.98	0.98	456
sort	0.68	0.19	0.29	112
string	0.72	0.99	0.84	391
accuracy			0.86	1215
macro avg	0.84	0.73	0.74	1215
weighted avg	0.87	0.86	0.84	1215

b1) Classification Report using dataset without duplicates

The main difference between figura a) and figure a1) is the cell 22 in which:

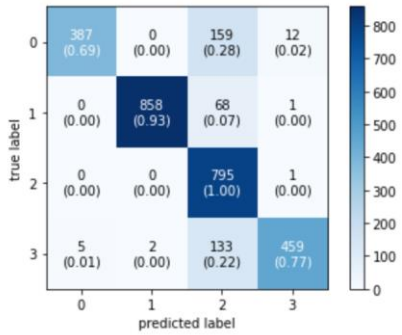
- Using dataset with duplicates I can see that the sort class has an hight F1-score;
- Using dataset without duplicates I can see that the sort class has a very low F1-score.

Considering only the F1-score I can say that:

- For encryption class is better to use the dataset with duplicates even If there isn't a big difference in the values;
- For math class is better to use the dataset without duplicates because I have F1-score that is better that the F1-score obtained usying the dataset with duplicates. But the difference is low;
- For sorting class I will use absolutely the dataset with duplicates;
- For string manipulation is better to use the dataset with duplicates.

So, in general I have better performance using the dataset with duplicates and in fact I have a better accuracy how I can see by compared figure b) with figure b1).

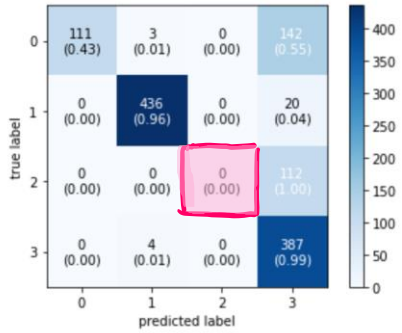
6.2.2 Multinomial



e) Confusion matrix using dataset with duplicates

	precision	recall	f1-score	support
encryption	0.99	0.69	0.81	558
math	1.00	0.93	0.96	927
sort	0.69	1.00	0.81	796
string	0.97	0.77	0.86	599
accuracy			0.87	2880
macro avg	0.91	0.85	0.86	2880
weighted avg	0.90	0.87	0.87	2880

f) Classification Report using dataset with duplicates



e1) Confusion matrix using dataset without duplicates

	precision	recall	f1-score	support
encryption	1.00	0.43	0.60	256
math	0.98	0.96	0.97	456
sort	0.00	0.00	0.00	112
string	0.59	0.99	0.74	391
accuracy			0.77	1215
macro avg	0.64	0.59	0.58	1215
weighted avg	0.77	0.77	0.73	1215

f1) Classification Report using dataset with duplicates

The main difference between figure e) and figure e1) is the cell 22:

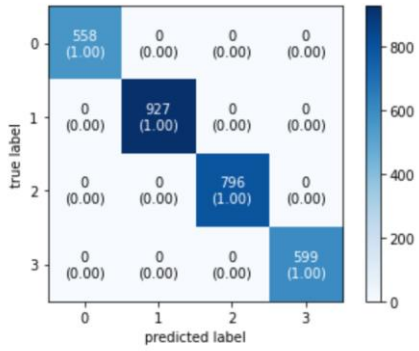
- Using dataset with duplicates I can see that the sort class has an hight F1-score;
- Using dataset without duplicates I can see that the sort class has F1-score equal to zero, which derives from the fact that both precision and recall have value equal to zero.

Considering only the F1-score I can say that:

- For encryption class is better to use the dataset with duplicates because I have a better F1-score than the other;
- For math class is indifferent to choose one or the other, because I have in both cases a very high value of F1-score;
- For sorting class I will use absolutely the dataset with duplicates, that has an optimal recall which allows to have a good F1-score. While using the dataset without duplicates I can see that sort has all the values (precision, recall, F1-score) equal to zero;
- For string manipulation is better to use the dataset with duplicates because I have a better F1-score than the other.

So, in general I have better performance using the dataset with duplicates and in fact I have a better accuracy how I can see by compared figure f) with figure f1).

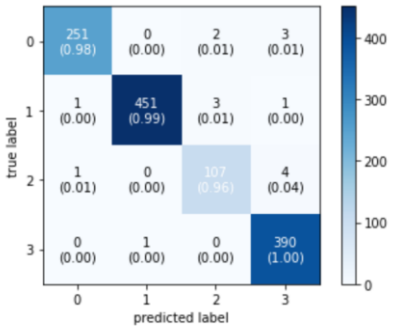
6.2.3 Random Forest



i) Confusion matrix using dataset with duplicates

	precision	recall	f1-score	support
encryption	1.00	1.00	1.00	558
math	1.00	1.00	1.00	927
sort	1.00	1.00	1.00	796
string	1.00	1.00	1.00	599
accuracy			1.00	2880
macro avg	1.00	1.00	1.00	2880
weighted avg	1.00	1.00	1.00	2880

j) Classification Report using dataset with duplicates



i1) Confusion matrix using dataset without duplicates

	precision	recall	f1-score	support
encryption	0.99	0.98	0.99	256
math	1.00	0.99	0.99	456
sort	0.96	0.96	0.96	112
string	0.98	1.00	0.99	391
accuracy			0.99	1215
macro avg	0.98	0.98	0.98	1215
weighted avg	0.99	0.99	0.99	1215

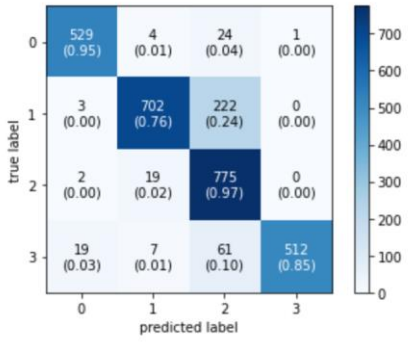
j1) Classification Report using dataset without duplicates

In this case all the classes that there are in the dataset have the same F1-score values.

So, it is indifferent to use the dataset with duplicates or without duplicates.

But to all intents and purposes if I look at the accuracy in figure j) and figure j1) I see that it is better to use the dataset with duplicates.

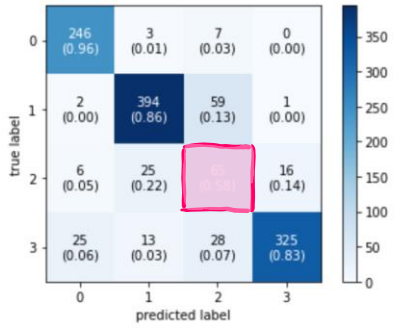
6.2.4 Gaussian



m) Confusion matrix using dataset with duplicates

	precision	recall	f1-score	support
encryption	0.96	0.95	0.95	558
math	0.96	0.76	0.85	927
sort	0.72	0.97	0.83	796
string	1.00	0.85	0.92	599
accuracy			0.87	2880
macro avg	0.91	0.88	0.89	2880
weighted avg	0.90	0.87	0.88	2880

n) Classification Report using dataset with duplicates



m1) Confusion matrix using dataset without duplicates

	precision	recall	f1-score	support
encryption	0.88	0.96	0.92	256
math	0.91	0.86	0.88	456
sort	0.41	0.58	0.48	112
string	0.95	0.83	0.89	391
accuracy			0.85	1215
macro avg	0.79	0.81	0.79	1215
weighted avg	0.87	0.85	0.86	1215

n1) Classification Report using dataset without duplicates

The main difference between figure m) and figure m1) is the cell 22:

- Using dataset with duplicates I can see that the sort class has an hight F1-score;
- Using dataset without duplicates I can see that the sort class has a low F1-score.

Considering only the F1-score I can say that:

- For encryption class is better to use the dataset with duplicates even If there isn't a big difference in the values;
- For math class is indifferent to choose one or the other, because I have in both cases a very high value of F1-score;
- For sorting class is better to use the dataset with duplicates;
- For string manipulation is indifferent to choose one or the other, because I have in both cases a very high value of F1-score.

So, in general I have better performance using the dataset with duplicates and in fact I have a better accuracy how I can see by compared figure n) with figure n1). But they are toosimilar.

Finally, I can say that is better to use a dataset with duplicates but if I only considered the accuracy I could say that even using the dataset without deuplicates I have quite good performace.

6.2.0.0 Summury of the performances given by the first and the second model

I represent the most interesting data in this summury table, in which I have the F1-score values in the cell corresponding to the model and class and I have the accuracy for each model.
We have two columns in each class because I insert in the first column the value that I have considering the dataset with duplicates and I insert in the second column the value that I have considering the dataset without duplicates.

In this table I decide to compute the avarage of F1-score value for each class considering all the model that I have analayzed. From this average I can still confirm that for all classes it is better to use the dataset with duplicates to have better performance. I only have an exception for the Math class, but in any case the average obtained with the dataset with duplicates is also very high.
For what concerns the accuracy, I see that is similar in both cases, but is better when I use the dataset with duplicates.

	F1-score								Accuracy	
Model	Encryption		Math		Sort		String			
Bernoulli	0.86	0.84	0.93	0.98	0.88	0.29	0.93	0.84	0.90	0.86
Multinominal	0.81	0.60	0.96	0.97	0.81	0.00	0.86	0.74	0.87	0.77
Random Forest	1.00	0.99	1.00	0.99	1.00	0.99	1.00	0.99	1.00	0.99
Gaussian	0.95	0.92	0.85	0.88	0.83	0.48	0.92	0.89	0.87	0.85
Avarage	**0.91**	**0.84**	**0.94**	**0.96**	**0.88**	**0.44**	**0.93**	**0.87**	**0.91**	**0.87**

6.3 Computational training time

I think that is interesting to report the computational training time.

To compute the time I have to import time among the libraries.

How I can see in the figure 1, using the pre-processing 1 or using the pre-processing 2:

- the time for Bernoulli and Multinomial doesn't change to much;
- the time for Random forest is six times better using pre-processing 2 instead of pre-processing 1.
- the time for Gaussian is sixty times better using pre-processing 2 instead of pre-processing 1.

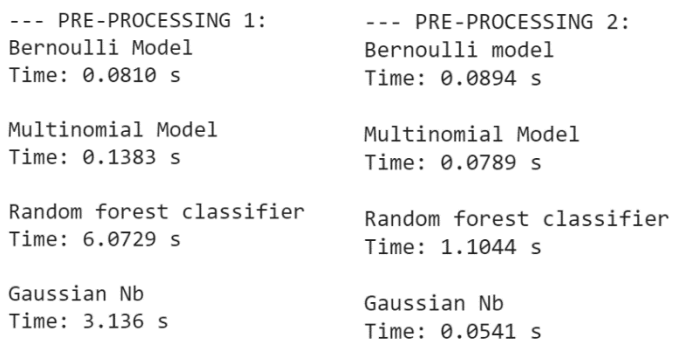


Figure 1: Computational time

In conclusion, I can consider that of the point of view of the time complexity is better to use the second type of pre-processing because I have only benefit, while for the perfromance is better to choose the first pre-processing.

Chapter 7: Blind Test

For blind test I have to decide one of the two methods that I explian above for the pre-processing of the data and I have to choose one of the four models that I use to see the performance.

I have five phases:

1. I loaded the blind test dataset with the functions to be classified;
2. I pre-process the dataset, I splitted the data and I fitted the model;
3. I have pre-processed the blind test dataset;
4. The model has classified the data of the blindset;
5. I wrote the result on the file.

Chapter 8: Conclusion

In conclusion, for blind test I decide to use *TfidfVectorizer()* for the pre-processing of the data because it has better performance than the pre-processing in which I subdivide the instructions in classes.
As model I choose *BernoulliNB()* because on average it has better values than gaussian and multinomial models.