2020

Report Homework1

MACHINE LEARNING

PAPA EDOARDO MATRICOLA: 1962169

Sommario

1. Project Overview

The goal of this project is solve one classification problem.
In particular the aim is a semantic prediction, more precisely, build a classifier that can predict which type of function we are facing, starting to specific list of assembly functions.

1.1 Dataset

If we choose a "*dataset.json*", it contains 14397 functions, each function belongs to one of aformetioned classes (*Encryption*, *String Manipulation*, *Math*, *Sorting*):

- 2724 are Encryption Functions;
- 3104 are String Manipulation;
- 4064 are Sorting Functions;
- 4504 are Math Functions;

The structure of the dataset is a JSONL, so for each lines we have a JSON composed in the following way:

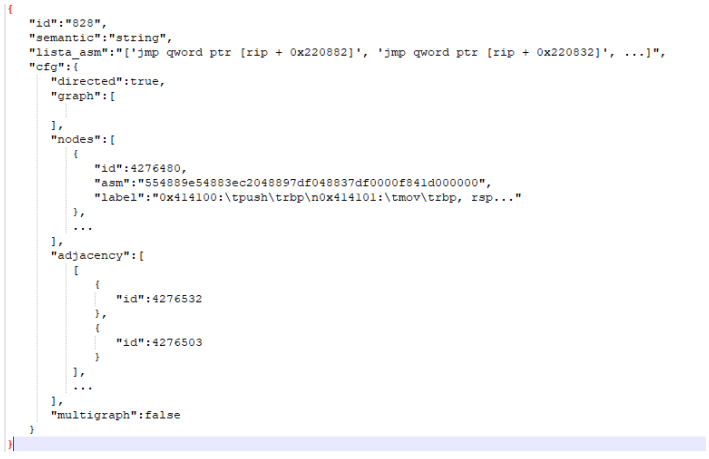

Figura 1 - Dataset Structure

The JSON file, that is composed of a multiple JSON lines, i decided to read it and go over using a pandas library, that allow read each lines through a function called "*read_json*". This function returns a DataFrame type that has a multi labels. In particular: *id*, *semantic*, *lista_*asm, *cfg*.

So, each lines have:

- ***ID*** parameter that provide to give it a unique value inside the JSON;
- ***SEMANTIC*** specified the which classes belong;
- ***LISTA_ASM*** specified the list of all assembly instructions;
- ***CFG*** rapresent the graph.

2. Vectorizer

The first question that someone can do are: "*why we should use a Vectorizer?*"

The answear of this question is very simply. In machine learning the algorithms operate on a numerical space of functionality. So, to improve machine learning on our data, we need to transform the data into a vector rapresentation. Now we can apply a numerical machine learning.

Inside this project i decided to use 3 different type of vectorizer, where each one have different property:

- CountVectorizer: creates a matrix with each unique word is rappresented by a column of the matrix and each text sample from the documenti s a row in the matrix. The value of each cell is the count of the word in that particular text semple.
- HashingVectorizer: HashingVectorizer is very similar to CountVectorizer. It consinst on hashing the words, then tokenize and encode document as neeeded. With this vectorizer, each token is mapped to a column position in a matrix.
- TfidfVectorizer: also this vectorizer is similar to CountVectorizer. In this case the vectorizer doesn't count the word but it calculate the word frequencies. This vectorizer have another important feature; infact it uses TF-IDF that provide to highlight words that are more interesting then the others. The TfidfVectorizer tokenize documents, learn the vocabulary and inverse document frequency weightings, ans allow you to encode new documents.

Before choose the Vectorizer i evaluated all the ones mentioned above. After that i decide to choose the TfdifVectorizer because it come out the better performant solution for this problem. If insead i used the CountVectorizer i could have some words that will appear many times and their aren't very significant in the coded vectors. I also decided to don't use HashingVectorized because this vectorizer require large vectors for encoding data. Moreover the hash is one-way function so there isn't way to convert the dencoding data to word.

Is important to consider in this vectorizer the parameter **ngram_range** that define the lower and upper limit range of values for different n-gram. This is useful for the vocabulary. In my case i consider a **ngram_range**

= (1,4) because in this way i have different combinations of the word in vocabulary. It improve the result of evaluation.

After choose a vectorizer i execute the *fit_transform* function. This function allow to learn the vocabulary and do the idf. This function is equivalent to call *fit* and after that *transform*.

One thing that is very important to think it's when evaluating a model we must use two different dataset, one for train and another one for test. This is very important because if use the same dataset this won't evaluete well and the model not works in realistic conditions. So we use two dataset: *train dataset* and *test dataset.*

3. Classification

Insiede this project we'll facing to a classification report that it contains different type of informations and is about key metrics in a classification problem.

In particular, the information that we'll see are mentioned below.

3.1 Accuracy

This parameter allow immediately if a model is being trained correctly or not. However it doesn't give more detail to the application problem.

I cannot use only this parameter for the metric, because it not works good when we have a imbalance class dataset. In this case, we also condider the f1-score.

$$\text{Accuracy} = \frac{Number\ of\ correct\ predictions}{Number\ of\ total\ predictions}$$

In confusion matrix we have

$$\text{Accuracy} = \frac{True\ positive + True\ negative}{Total\ predictions}$$

3.2 Pecision

The precisioni s the accuracy of the system predict a positive classes. In other words, helps when the costs of false positive are high.

In confusion matrix we have

$$\text{Precision} = \frac{True\ positive}{True\ positive + False\ positive}$$

3.3 Recall

The recall it's also called "sensitive" or "true positive rate" and indicates the ratio of positive instances correctly identified. In other words, helps when the cost of false negative is high.

$$\text{Recall} = \frac{True\ positive}{True\ positive + False\ negative}$$

3.4 F1-score

F1-score is an overall measure of a model's accuracy that combines precision and recall. For have a good f1-score, we could have a low false positives and low false negatives. An f1-score is considered perfect when it's 1, while the model is a failure when it's 0.

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

4. Multiclass Classification inside project

The first think that we need to do are: split the dataset into two different dataset called by "*X_train*" and "*X_test*" with relative labels "*y_train*" and "*y_test*". After that we need to create a model for prediction and at the end we need to predict the value and plot the result with *classification_report* and *confusion_matrix*.

4.1 Split Data

We want to considera the output as a semantic, so we take a column of the dataset labelled with "*semantic*". The *train_test_split* is a function defined in Sklearn model section for split in a esy way the dataset into two different ones. The first is a training set and the second is a testing set.
We pass to this function 4 parmameters. The first parameter is a sparse matrix of the dataset passed before inside vectorizer (for this project i choose TfidfVectorizer). The second parameter is a list of all label of the dataset. The third parameter is a test dataset size, that decides the size of the data tha has to be split. For example if we use 0.2 as the value, automatically we'll have a train set with 80% of the original dataset and the remain 20% is test set of the original dataset. The last parameter is a random state that will act as a seed for random number generator during the split.

4.2 Model

In this project different models are tested, such as:

- Logistic Regression;
- Multinomial Naive Bayes;
- Bernoulli
- SVM Linear Kernel

Each models will be evaluated using the classification report and confusion matrix.

4.3 Evaluations

4.3.1 Bernoulli

Bernulli is a Naive Bayes classifier for multivariate Bernoulli models. This classifier is suitable for discrete data. I don't consider Random Forest because we have high data inside dataset that alsmost certanly generates overfitting, compromising the evaluation of the model.

Classification Report and Confusion Matrix

Bernoulli with alpha value = 1.0

	precision	recall	f1-score	support
encryption	1.00	0.65	0.79	396
math	0.97	0.99	0.98	778
sort	0.45	0.03	0.05	173
string	0.70	0.99	0.82	678
accuracy			0.84	2025
macro avg	0.78	0.66	0.66	2025
weighted avg	0.84	0.84	0.81	2025

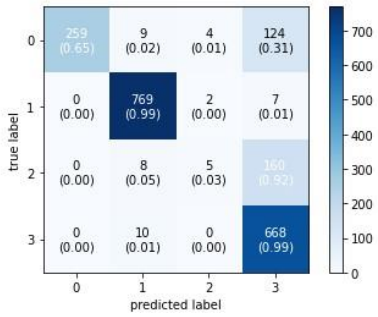

Figura 2 - Bernoulli test set (alpha = 1.0)

	precision	recall	f1-score	support
encryption	1.00	0.72	0.84	714
math	0.96	1.00	0.98	1634
sort	1.00	0.03	0.06	364
string	0.72	0.99	0.83	1336
accuracy			0.86	4048
macro avg	0.92	0.68	0.68	4048
weighted avg	0.89	0.86	0.82	4048

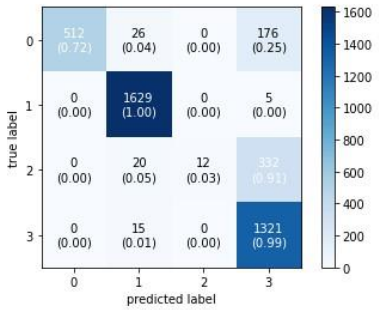

Figura 3 - Bernoulli train set (alpha = 1.0)

Bernoulli with alpha value = 0.01

	precision	recall	f1-score	support
encryption	1.00	0.89	0.94	396
math	0.99	0.99	0.99	778
sort	0.74	1.00	0.85	173
string	1.00	0.99	0.99	678
accuracy			0.97	2025
macro avg	0.93	0.97	0.94	2025
weighted avg	0.98	0.97	0.97	2025

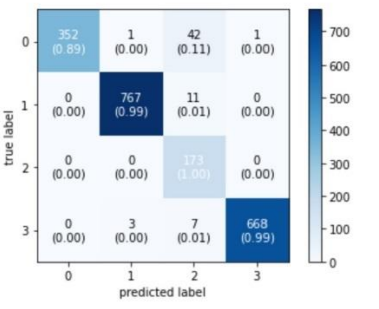

Figura 4 - Bernoulli test set (alpha = 0.01)

	precision	recall	f1-score	support
encryption	1.00	0.95	0.97	714
math	1.00	1.00	1.00	1634
sort	0.91	0.99	0.95	364
string	1.00	1.00	1.00	1336
accuracy			0.99	4048
macro avg	0.98	0.98	0.98	4048
weighted avg	0.99	0.99	0.99	4048

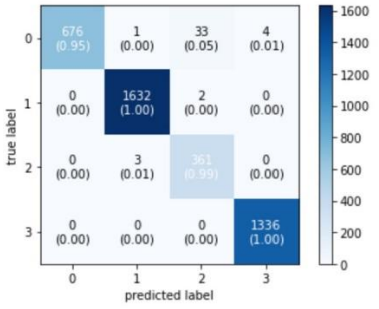

Figura 5 - Bernoulli train set (alpha = 0.01)

Conclusion

The performance of this model isn't the best that i observed. I started trying the Bernoulli model with alpha *value = 1.0* .There are a big gap between f1_score of encryption, math, sort and string. I made a second step where i tried to change an alpha *value = 0.01*. In this case i diminished the gap between f1_score of encryption, math, sort and string. In conclusion the Bernoulli model generate an overfitting, so i completely reject this model for solve this problem.

4.3.2 Logistic Regression

The idea of Logistic Regression is find a releptionship between specific feature and probability of particular comeout.

For this evaluation the Logistic Regression is created specifying some parameters. We need to find multiclass classifier, so we consider *multi_class = multinomial* but for do this operation we need to choose different solver from the default one. In my case i select "*solver = saga*" and "*penality = l1*". The penality is a regularization that allow to avoid the overfitting by discriminate the high-valued regression coefficients. In other words, it reduce the parameters for semplify the model.

Classification Report and Confusion Matrix

Logistic Regression with penality = l1

	precision	recall	f1-score	support
encryption	0.99	0.94	0.96	396
math	0.99	0.98	0.99	778
sort	0.84	0.90	0.87	173
string	0.97	0.99	0.98	678
accuracy			0.97	2025
macro avg	0.95	0.95	0.95	2025
weighted avg	0.97	0.97	0.97	2025

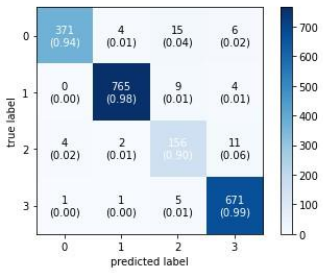

Figura 6 - Logistic Regression test set penality l1

	precision	recall	f1-score	support
encryption	0.98	0.95	0.97	714
math	0.99	0.99	0.99	1634
sort	0.91	0.93	0.92	364
string	0.98	1.00	0.99	1336
accuracy			0.98	4048
macro avg	0.97	0.97	0.97	4048
weighted avg	0.98	0.98	0.98	4048

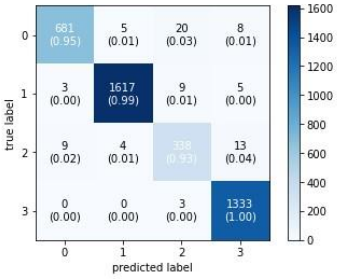

Figura 7 - Logistic Regression train set penality l1

Logistic Regression with penality = l2

	precision	recall	f1-score	support
encryption	0.98	0.96	0.97	396
math	1.00	0.99	0.99	778
sort	0.93	0.89	0.91	173
string	0.96	1.00	0.98	678
accuracy			0.98	2025
macro avg	0.97	0.96	0.96	2025
weighted avg	0.98	0.98	0.98	2025

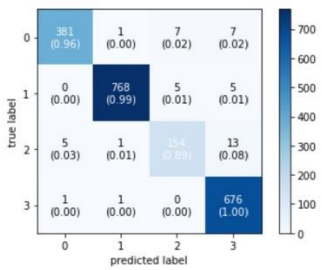

Figura 8 - Logistic Regression test set penality l2

	precision	recall	f1-score	support
encryption	1.00	0.98	0.99	714
math	1.00	1.00	1.00	1634
sort	0.98	0.98	0.98	364
string	0.99	1.00	0.99	1336
accuracy			0.99	4048
macro avg	0.99	0.99	0.99	4048
weighted avg	0.99	0.99	0.99	4048

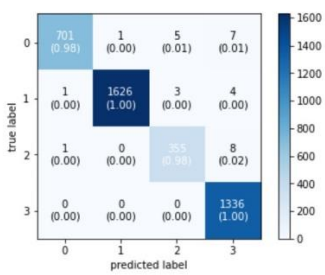

Figura 9 - Logistic Regression train set penality l2

Consider the figure 6 we have 4x4 matrix, on the columns the prediction and on the rows the true values. The column 0 corresponding to "*encryption*", 1 corresponding to "*math*", 2 corresponding to "*sort*", 3 corresponding to "*string*". In this particular case, if we consider the *encryption* row, we can see that 371 of these are correctly and incorrecly predicted 4 of the *encryption* to be *math*, 15 of *encryption* to be *sort* and 6 of *encryption* to be *string*. Now, if we consider the *encryption* column, we can see thare are 376 element that are predicted as *encryption* by the model but only 371 element are really *encryption*, 4 elements are *sort* and the last one is *string*. The same is for other three classes.

Conclusion

This is the best result that i've obtained. I started trying the Linear Regression with *penality = l1 (figure 6)*.We can see from this evaluations that Logistic Regression has high values with regard to *accuracy* and also the other metrics have high values, in particular the *f1_score*. The most important thing is check if the model causes overfitting. Fortunally, in this case the model doesn't cause a overfitting problem. I made a second step where i changed the value of penality to l2. We can see the *f1_score* and *accuracy* was improve, but we entroduced a overfitting problem. So, the best result is using this model with *penality=l1*.

4.3.3 Multinomial Naive Bayes

Classification Report and Confusion Matrix

Multinomial NB with alpha = 1.0

	precision	recall	f1-score	support
encryption	1.00	0.80	0.89	396
math	0.99	0.97	0.98	778
sort	1.00	0.02	0.03	173
string	0.72	1.00	0.83	678
accuracy			0.87	2025
macro avg	0.93	0.70	0.69	2025
weighted avg	0.90	0.87	0.83	2025

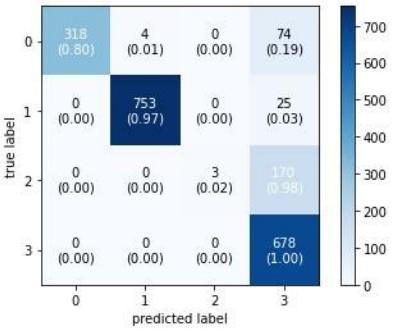

Figura 10 - Multinomial Naive Bayes test set alpha=1.0

	precision	recall	f1-score	support
encryption	1.00	0.87	0.93	714
math	1.00	0.98	0.99	1634
sort	1.00	0.03	0.06	364
string	0.74	1.00	0.85	1336
accuracy			0.88	4048
macro avg	0.93	0.72	0.71	4048
weighted avg	0.91	0.88	0.85	4048

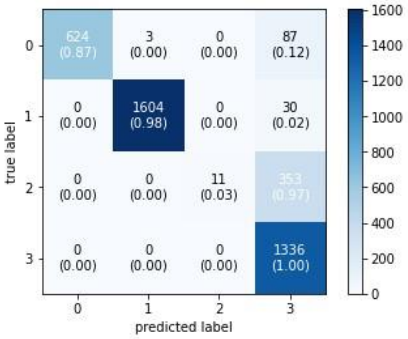

Figura 11 - Multinomial Naive Bayes train set alpha=1.0

Multinomial NB with alpha = 0.01

	precision	recall	f1-score	support
encryption	1.00	0.97	0.98	396
math	1.00	0.98	0.99	778
sort	0.89	0.98	0.93	173
string	0.99	1.00	0.99	678
accuracy			0.99	2025
macro avg	0.97	0.98	0.98	2025
weighted avg	0.99	0.99	0.99	2025

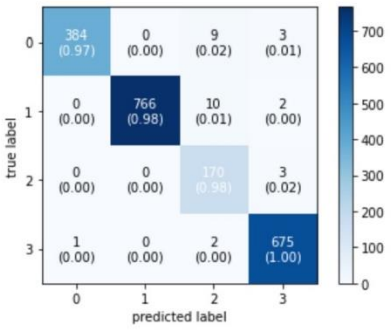

Figura 12 - Multinomial Naive Bayes test set alpha=0.01

	precision	recall	f1-score	support
encryption	1.00	0.99	1.00	714
math	1.00	1.00	1.00	1634
sort	0.99	0.99	0.99	364
string	0.99	1.00	1.00	1336
accuracy			1.00	4048
macro avg	1.00	1.00	1.00	4048
weighted avg	1.00	1.00	1.00	4048

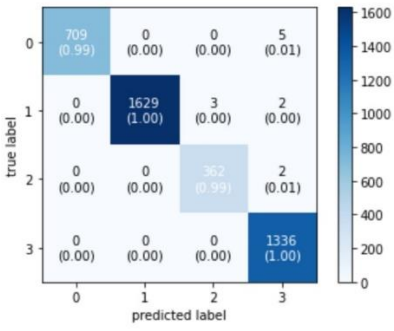

Figura 13 - Multinomial Naive Bayes train set alpha=0.01

Conclusion

The performance of this model isn't the best that i observed. I started trying the Multinomial NB model with alpha *value = 1.0* .There are a big gap between f1_score of sort and others values. I made a second step where i tried to change an alpha *value = 0.01*. In this case i diminished the gap between f1_score of encryption, math, sort and string. In conclusion the Multinomial NB model generate an overfitting, so i completely reject this model for solve this problem.

4.3.4 SVM Linear Kernel

Linear Kernel SVM is mainly used when the data is linear saparable. With this model we can use a C parameter that it's useful for the SVM optimization for avoid misclassifying. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane. Instead a small value of C will cause the optimizer to look for a larger-margin separating hyperplane.

Classification Report and Confusion Matrix

SVM Linear Kernel with C = 1.0

	precision	recall	f1-score	support
encryption	0.99	0.97	0.98	396
math	1.00	0.99	0.99	778
sort	0.92	0.98	0.95	173
string	0.99	1.00	1.00	678
accuracy			0.99	2025
macro avg	0.98	0.98	0.98	2025
weighted avg	0.99	0.99	0.99	2025

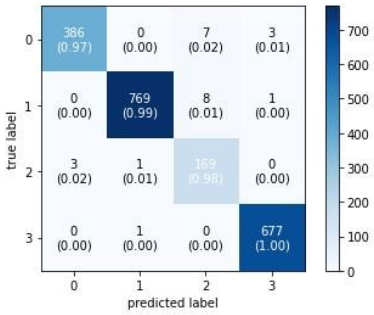

Figura 14 - SVM Linear Kernel test set C=1

	precision	recall	f1-score	support
encryption	1.00	1.00	1.00	714
math	1.00	1.00	1.00	1634
sort	1.00	1.00	1.00	364
string	1.00	1.00	1.00	1336
accuracy			1.00	4048
macro avg	1.00	1.00	1.00	4048
weighted avg	1.00	1.00	1.00	4048

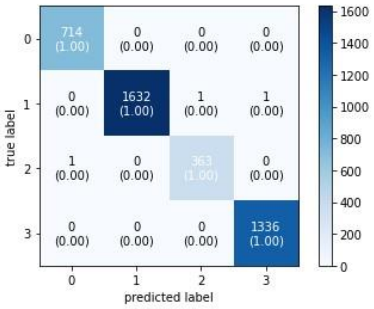

Figura 15 - SVM Linear Kernel train set C=1

SVM Linear Kernel with C = 10.0

	precision	recall	f1-score	support
encryption	1.00	0.98	0.99	396
math	1.00	0.99	0.99	778
sort	0.92	0.99	0.96	173
string	1.00	1.00	1.00	678
accuracy			0.99	2025
macro avg	0.98	0.99	0.98	2025
weighted avg	0.99	0.99	0.99	2025

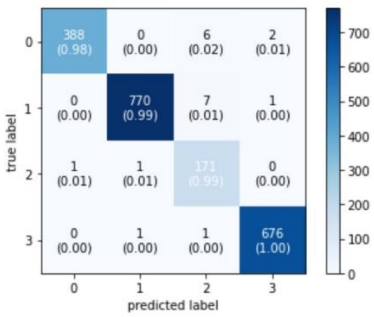

Figura 16- SVM Linear Kernel test set C=10

	precision	recall	f1-score	support
encryption	1.00	1.00	1.00	714
math	1.00	1.00	1.00	1634
sort	1.00	1.00	1.00	364
string	1.00	1.00	1.00	1336
accuracy			1.00	4048
macro avg	1.00	1.00	1.00	4048
weighted avg	1.00	1.00	1.00	4048

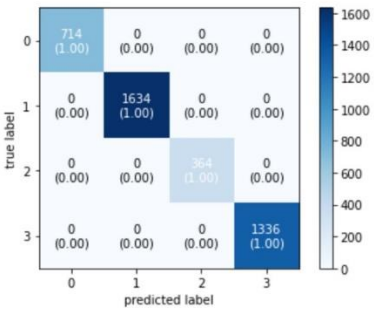

Figura 17 - SVM Linear Kernel train set C=10

Conclusion

I stared trying this model with parameter of *C=1* and how we can see, this model generates overfitting, so we cannot considerate it. Even if it has a good values of metrics, in particular *f1_score*. I made a second step where i change the value of *C* to 10 and also in this case it generates overfitting.

4.4 Conclusion of evaluation

In conclusion, the best model for solve this problem is Logistic Regression because we want to avoid as much as possible the overfitting problem, and this model is the best for prediction of *semantic*. Another advantage point of this model it's that the *f1_score* metric is very good.

5. Blind Test

In this project we have a file avaiable called "*blindtest*" that contains a set of data without semantic information. So the aim is predict all the semantics for the input data and make another file with the result. In this phase i read the file and i pass it inside the vectorizer (TfidfVectorizer). After that i do the *fit* and *transformation*. I make prediction using as model Logistic Regression. After do this operations, i write the predictions inside a .txt file. For each row we can find a prediction that corrisponding line of *blindtest* file.