Report Homework 1 - Machine Learning

Sveva Pepe

Chapter 1

Project Overview

The goal is to solve two classification problems:

1. Optimization prediction, more precisely, build a binary classifier that can predict if a function has been compiled with optimization HIGH or LOW
2. Compiler prediction, more precisely, build a multiclass classifier that can predict the compiler used to compile a specific function.

1.1 Dataset

The dataset contains 30000 functions compiled with 3 different compilers: gcc, icc, clang.
Compiler distribution is balanced and the optimization distribution is not balanced.
The structure of the dataset is a JSON file composed in the following way:


```
{
    "instructions": ["xor edx edx", "cmp rdi rsi", "mov eax 0xffffffff", "seta
                    dl", "cmovae eax edx", "ret"],
    "opt": "H",
    "compiler": "gcc "
}
```


Figure 1.1: Dataset Structure

The json file, which is actually of the jsonl type, I decided to read it using a particular python library, pandas, which reads through a function called *read_json* and returns a variable consisting of a DataFrame type that has the three labels: instructions , opt and compiler.
Each row consists of a "program" (instance) which has 3 columns (features): the first is the program instructions (list of instructions), the second are the optimizations (H,L) and each line has a single optimization associated with it, finally the last column are the compilers (gcc, clang, icc)

After that, I have to go and extract the instructions. In particular I decided to try to do two different types of extraction:

1. I extract all the instructions of the various programs
2. I extract only the first word of each instruction of the various programs

Having chosen the method by which i want to extract the various instructions, i get, in both cases, a list of strings that i will then pass to a Vectorizer.

1.2 Vectorizer

The first step is to understand why i use a Vectorizer. This is because in machine learning the algorithms operate on a numerical space of functionality. To perform machine learning on our

data, we need to transform our data into vector representations so we can apply numerical machine learning. This process is just the vectorization, so i need the Vectorizer.
Now i have to decide which Vectorizer i want to use for this kind of problems.
There are 3 types of Vectorizer:

- CountVectorizer: provides a simple way to both tokenize a collection of text documents (our case list of strings) and build a vocabulary of known words, but also to encode new documents using that vocabulary.
- TfidfVectorizer: similar to CountVectorizer but instead of count words it calculate word frequencies. TF-IDF are word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across documents. The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents.
- HashingVectorizer: mplements this approach that can be used to consistently hash words, then tokenize and encode documents as needed.

I choose TfdifVectorizer because it turns out to be better as a performance for this problem since if i used the CountVectorizer i could go to the problem of having some words that will appear many times and their important counts will not be very significant in the coded vectors.
I also do not use HashingVectorizer because they require large vectors for encoding data and they set large requirements on memory and slow down algorithms and also that the hash is a one-way function so there is no way to convert the encoding back to a word.

A very important thing to consider is that the Vectorizer must be the same for both classification problems, this because in the end with the blind set i will only have to apply on its data the vocabulary created by Vectorizar.
That is, for the two exercises I have to do two functions of the Vectorizer: the "fit" and the "transform"; instead, as far as the Blind Set is concerned, I will only do the transfom.
Where the fit is the function in order to learn a vocabulary from one or more documents, instead the transform is function on one or more documents as needed to encode each as a vector.

With function *fit_trasform* TfidfVectorizer learn vocabulary and idf, return term-document matrix. This is equivalent to "fit" followed by "transform", but more efficiently implemented.

Another important thing was to consider the parameter **ngram_range** of the vectorizer that goes to consider the lower and upper limit of the range of values for different n-grams to be extracted. All values of n will be used such that $min\_n \leq n \leq max\_n$. Useful for vocabulary. In this case i consider a ngram_range = (1,3) because in this way i have different combinations of words in vocabulary and i can improve the results of evaluations of model.

One common mistake when evaluating a model is to train and test it on the same dataset: this is problematic because you this will not evaluate how well the model works in realistic conditions, on unseen data, and models that overfit to the data will seem to perform better. So we need to divide the data into training and testing sets.

1.3 Metrics

1.3.1 Accuracy

Accuracy can tell us immediately whether a model is being trained correctly and how it may perform generally. However, it does not give detailed information regarding its application to the problem.
The problem with using accuracy as your main performance metric is that it does not do well when you have a severe class imbalance, such as exercise 1 that we cannot take care only of accuracy

but also of f1_score

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

In particular in confusion matrix we have

$$\text{accuracy} = \frac{\text{True positive} + \text{True negative}}{\text{Total example}}$$

1.3.2 Precision

Precision helps when the costs of false positives are high. So in confusion matrix

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

1.3.3 Recall

Recall helps when the cost of false negatives is high.

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

1.3.4 f1_score

f1 is an overall measure of a model's accuracy that combines precision and recall. That is, a good f1_score means that you have low false positives and low false negatives, so you're correctly identifying real threats and you are not disturbed by false alarms. An f1 score is considered perfect when it's 1, while the model is a total failure when it's 0

$$f1\_score = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

Chapter 2

Binary Classification

2.0.1 Split Data

We want to consider the output as optimization, so we take a column of dataset labelled "opt". The *train_test_split* split in an easy way the dataset into the training and the testing set in various proportions. The main parameters that are passed to this function are, one is a sparse matrix (result of vectorization by TfidfVectorizer) and another is a list of optimization value of the dataset ("opt"). In particular there are others important parameters: **test size**, that decides the size of the data that has to be split as the test set, for example, if you pass 0.2 as the value, the dataset will be split 20% as the test set and the remain 80% is the training set; and **random state**, that will act as the seed for the random number generator during the split.

2.1 Extraction: Choose all the instructions of the various programs

2.1.1 Model

Different models are tested such as Bernoulli, LogisticRegression, SVM. Each model will be evaluated using the classification report and confusion matrix. Furthermore, given that the distribution of the optimization is not balanced, I will not only observe the accuracy of the model but also f1_score.

2.1.2 First Evaluation

Bernoulli

Bernoulli is a Naive Bayes classifier for multivariate Bernoulli models. Like MultinomialNB, this classifier is suitable for discrete data. The difference is that while MultinomialNB works with occurrence counts, BernoulliNB is designed for binary/boolean features. I did not consider Random Forest because it is a model that for rather high datasets almost certainly generates overfitting, which compromises the evaluation of the model itself. In this evaluation Bernoulli model is created without passing any type of parameter.

Classification Report and Confusion Matrix

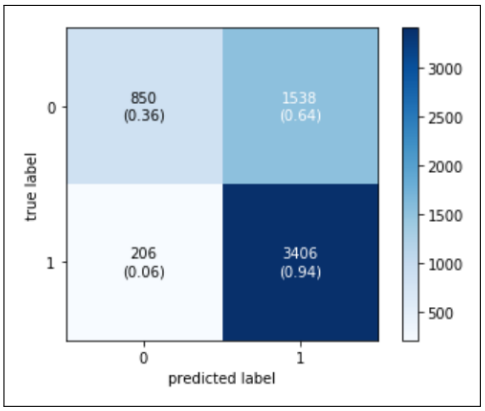
The performance is not the best as we observe how there is a big gap between f1_score of H and L. Accuracy is not bad but we cannot rely solely on it because it can be misleading. We can see if by changing different parameters of the model, it improves performance.

Logistic Regression

Logistic regression without specifying any parameters in the creation of the model.

	precision	recall	f1-score	support
H	0.80	0.36	0.49	2388
L	0.69	0.94	0.80	3612
accuracy			0.71	6000
macro avg	0.75	0.65	0.64	6000
weighted avg	0.74	0.71	0.68	6000

(a) Classification Report Bernoulli Test Set

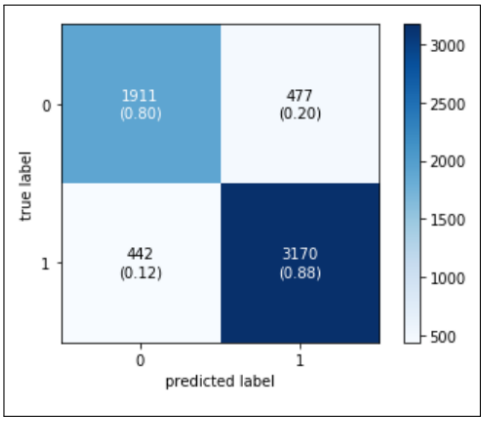


(b) Confusion Matrix Bernoulli

Classification Report and Confusion Matrix

	precision	recall	f1-score	support
H	0.81	0.80	0.81	2388
L	0.87	0.88	0.87	3612
accuracy			0.85	6000
macro avg	0.84	0.84	0.84	6000
weighted avg	0.85	0.85	0.85	6000

(a) Classification Report Logistic Regression Test Set



(b) Confusion Matrix Logistic Regression

Here the values are not bad, because as we can see both from the confusion matrix and from the classification report we have that the f1_score of the two optimizations are almost similar and also the accuracy is a fairly high value.
But we need to check if Logistic regression does overfitting.

Check if a model does overfitting by comparing the accuracy of the train set with that of the test set. If the difference between the two is very high it means that the model does overfitting, otherwise it does not.

	precision	recall	f1-score	support
H	0.81	0.80	0.81	2388
L	0.87	0.88	0.87	3612
accuracy			0.85	6000
macro avg	0.84	0.84	0.84	6000
weighted avg	0.85	0.85	0.85	6000

(a) Classification Report Logistic Regression Test Set

	precision	recall	f1-score	support
H	0.91	0.89	0.90	9688
L	0.93	0.94	0.93	14312
accuracy			0.92	24000
macro avg	0.92	0.92	0.92	24000
weighted avg	0.92	0.92	0.92	24000

(b) Classification Report Logistic Regression Train Set

As we can see we have the difference between the accuracy is quite high, we can see that it is also the difference between f1_score of the two sets. This means that we cannot use logistic regression in this way but that by changing certain parameters of the model we could avoid this overfitting problem. we can take this model into consideration but trying to avoid overfitting of some kind,

but this avoiding overfitting does not worsen the performance of the model on datasets.

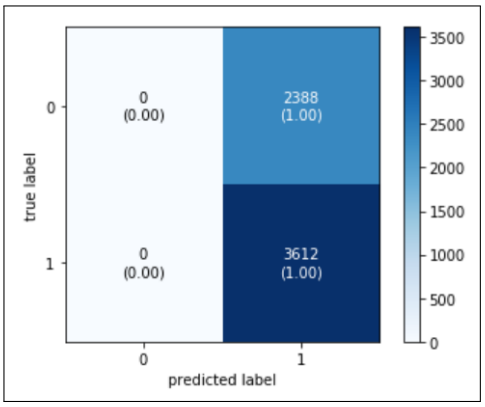
SVM

SVM without specifying any parameters in the creation of the model.

Classification Report and Confusion Matrix

	precision	recall	f1-score	support
H	0.00	0.00	0.00	2388
L	0.60	1.00	0.75	3612
accuracy			0.60	6000
macro avg	0.30	0.50	0.38	6000
weighted avg	0.36	0.60	0.45	6000

(a) Classification Report SVM Test Set



(b) Confusion Matrix SVM

The evaluation leads to rather negative results as can be seen from the f1_score of H. This model cannot be taken into consideration to be used to solve this problem. Here we need to understand if it is possible to improve performance without overffing.

2.1.3 Second Evaluation

Bernoulli

I considered changing the alpha parameter of the Bernoulli model to try to improve the situation, but I increased the accuracy and f1_score but this was going to generate an overfitting, which makes me completely reject the Bernoulli model to solve this problem.

Logistic Regression

Classification Report

	precision	recall	f1-score	support
H	0.81	0.81	0.81	2388
L	0.87	0.87	0.87	3612
accuracy			0.85	6000
macro avg	0.84	0.84	0.84	6000
weighted avg	0.85	0.85	0.85	6000

(a) Classification Report Logistic Regression Test Set

	precision	recall	f1-score	support
H	0.83	0.83	0.83	9688
L	0.88	0.88	0.88	14312
accuracy			0.86	24000
macro avg	0.86	0.86	0.86	24000
weighted avg	0.86	0.86	0.86	24000

(b) Classification Report Logistic Regression Train Set

Having changed several parameters in the logistic regression such as the solver and the type of penalty I have come to the conclusion of having good performances, both as regards the two f1_score and as regards the accuracy, without overfitting.

I have not considered to put the parameter "class_weight = balanced" on purpose because I would have forced to make balanced a distribution that doesn't appear to be the one, going to modify then my distribution

SVM

In the SVM the parameters "kernel = linear" and "gamma = scale" have been considered to increase the performance of the model.

Classification Report

	precision	recall	f1-score	support
H	0.00	0.00	0.00	2388
L	0.60	1.00	0.75	3612
accuracy			0.60	6000
macro avg	0.30	0.50	0.38	6000
weighted avg	0.36	0.60	0.45	6000

(a) Classification Report SVM Test Set

	precision	recall	f1-score	support
H	0.97	0.97	0.97	9688
L	0.98	0.98	0.98	14312
accuracy			0.98	24000
macro avg	0.98	0.98	0.98	24000
weighted avg	0.98	0.98	0.98	24000

(b) Classification Report SVM Train Set

SVM seems to be better in terms of results but we immediately notice how it does overfitting and therefore also he is not to be considered to solve the binary classification.

2.2 Extraction: Choose first word of each instructions of programs

Consider the same models seen in the previous extraction to evaluate which of the two extraction results to be better for the problem.

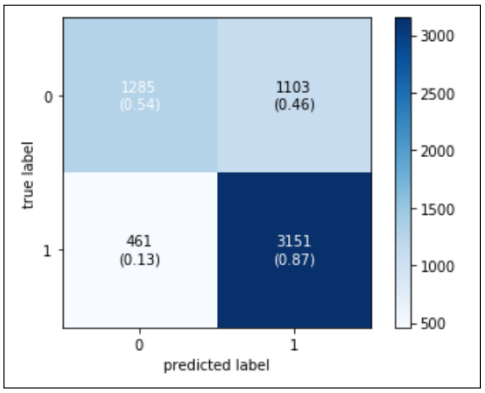
2.2.1 First Evaluation

Bernoulli

Classification Report and Confusion Matrix

	precision	recall	f1-score	support
H	0.74	0.56	0.64	2388
L	0.75	0.87	0.81	3612
accuracy			0.75	6000
macro avg	0.75	0.71	0.72	6000
weighted avg	0.75	0.75	0.74	6000

(a) Classification Report Bernoulli Test Set



(b) Confusion Matrix Bernoulli

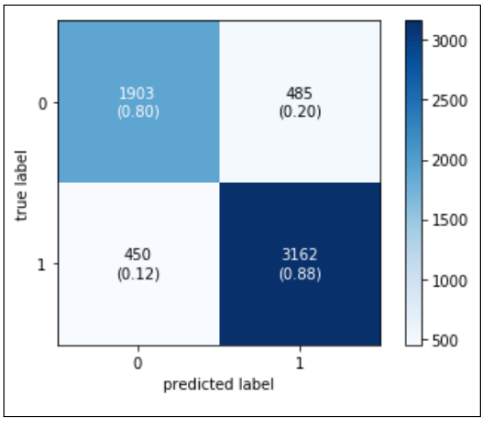
Bernoulli appears to be better in this extraction than the previous one, given also by the fact of having fewer instructions, but still having a fairly large gap between the two H and L optimization, so it is not so good, althought it doesn't do overfitting.

Logistic Regression

Classification Report and Confusion Matrix

	precision	recall	f1-score	support
H	0.81	0.80	0.80	2388
L	0.87	0.88	0.87	3612
accuracy			0.84	6000
macro avg	0.84	0.84	0.84	6000
weighted avg	0.84	0.84	0.84	6000

(a) Classification Report Logistic Regression Test Set



(b) Confusion Matrix Logistic Regression

	precision	recall	f1-score	support
H	0.85	0.83	0.84	9688
L	0.89	0.90	0.89	14312
accuracy			0.87	24000
macro avg	0.87	0.87	0.87	24000
weighted avg	0.87	0.87	0.87	24000

(a) Classification Report Logistic Regression Train Set

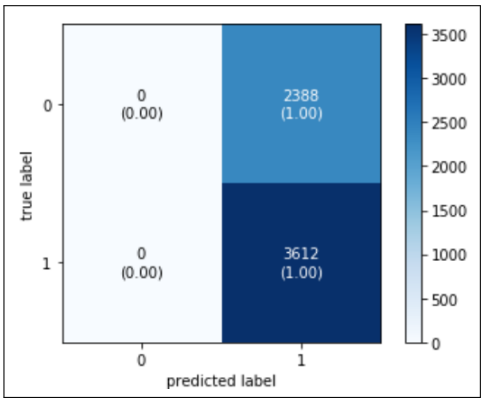
Logistic regression is better for now but always has the problem of overfitting, even if less than the first evaluation of the other extraction.

SVM

Classification Report and Confusion Matrix

	precision	recall	f1-score	support
H	0.00	0.00	0.00	2388
L	0.60	1.00	0.75	3612
accuracy			0.60	6000
macro avg	0.30	0.50	0.38	6000
weighted avg	0.36	0.60	0.45	6000

(a) Classification Report SVM Test Set



(b) Confusion Matrix SVM

Not going to specify any parameter we see that between the two extractions the SVM are the same, this implies that the default kernel, that is "rbf", is not valid for this problem that we are solving.

2.2.2 Second Evaluation

Bernoulli

Changing the "alpha" values of Bernoulli model does not change the outcome, it always remains on those values, the model is not perfect but it is not to be discarded because it does not overfitting.

Logistic Regression

Specify the same parameters as the previous extraction for logistic regression.

	precision	recall	f1-score	support
H	0.80	0.81	0.80	2388
L	0.87	0.87	0.87	3612
accuracy			0.84	6000
macro avg	0.84	0.84	0.84	6000
weighted avg	0.84	0.84	0.84	6000

(a) Classification Report Logistic Regression Test Set

	precision	recall	f1-score	support
H	0.83	0.82	0.83	9688
L	0.88	0.89	0.88	14312
accuracy			0.86	24000
macro avg	0.86	0.86	0.86	24000
weighted avg	0.86	0.86	0.86	24000

(b) Classification Report Logistic Regression Train Set

As we can see also here as in the previous extraction we get an improvement in the results of the model. No overfitting is done and the values are quite high and good to be taken into consideration for the resolution of the problem.

SVM

Specific as the previous extraction "kernel = linear" and "gamma = scale"

	precision	recall	f1-score	support
H	0.82	0.83	0.83	2388
L	0.89	0.88	0.88	3612
accuracy			0.86	6000
macro avg	0.85	0.85	0.85	6000
weighted avg	0.86	0.86	0.86	6000

(a) Classification Report SVM Test Set

	precision	recall	f1-score	support
H	0.88	0.87	0.87	9688
L	0.91	0.92	0.91	14312
accuracy			0.90	24000
macro avg	0.89	0.89	0.89	24000
weighted avg	0.90	0.90	0.90	24000

(b) Classification Report SVM Train Set

SVM, from how it can be seen it makes a slight overfitting even if not too much and obtains excellent values regarding f1_score and accuracy.

In general, both the Regression and SVM logistic would be good, but since we want to avoid as much as possible that there is any overfitting, I go to choose the Logistic Regression, even if the performance is slightly lower than that of the SVM.

In conclusion, as far as the optimization problem is concerned, we have the same model for both extractions (Logistic Regression), and as far as extraction is concerned, the former is preferable but only because it is better than little in the evaluation of metrics

Chapter 3

Multiclass Classification

3.0.1 Split Data

We want to consider the output as optimization, so we take a column of dataset labelled "compiler". I use the same function used in the previous problem *train_test_split* thath split the dataset into the train and the test set. Test set contains 20% of dataset and train set contains 80% of dataset.

3.1 Extraction: Choose all the instructions of the various programs

3.1.1 Model

We use only two models are considered: Logistic Regression and Multinomial Naive Bayes.

3.1.2 Evaluation

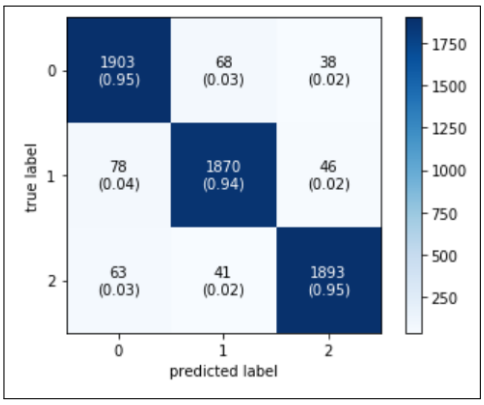
Logistic Regression

Using the Logistic Regression specifying some parameters, since we need to find a multiclass classifier, we consider the "multi_classes = multinomial" parameter, but to do this we need to set a solver different from the default one ('liblinear'), for example "solver = saga" and "penalty = l1". Where penalty is used to specify the norm used in the penalization and solver is algorithm to use in the optimization problem, in this case for multiclass problem.

Classification Report and Confusion Matrix

	precision	recall	f1-score	support
clang	0.94	0.94	0.94	2001
gcc	0.93	0.94	0.94	1978
icc	0.96	0.95	0.95	2021
accuracy			0.94	6000
macro avg	0.94	0.94	0.94	6000
weighted avg	0.94	0.94	0.94	6000

(a) Classification Report Logistic Regression Test Set



(b) Confusion Matrix Logistic Regression

For 3x3 matrix we have that the columns are the predictions and the rows must therefore be the

true values. 0 correspond to 'gcc', '1' correspond to 'clang' and finally 2 correspond to 'icc'. The first row are the true 'gcc'. The model of Logistic Regression predicted 1903 of these correctly and incorrectly predicted 68 of the 'gcc' to be 'clang' and 38 of the 'gcc' to be 'icc'. Looking at the 'gcc' column, of the 2044 'gcc' predicted by the model (sum of column 0), 1903 were actually 'gcc', while 78 were 'clang' incorrectly predicted to be 'gcc' and 63 were 'icc' incorrectly predicted to be 'gcc'. The same is for the other two classes.

We can see from both evaluations that Logistic Regression has rather high values with regard to accuracy and in this case unlike the previous problem we have a balanced distribution of the compiler so we can consider mainly the accuracy, but also taking into account the other metrics because accuracy is not foolproof. But the most important thing is to check if the model causes overfitting. In this case Logistic regression doesn't causes overfitting, as we can see:

	precision	recall	f1-score	support
clang	0.94	0.94	0.94	2001
gcc	0.93	0.94	0.94	1978
icc	0.96	0.95	0.95	2021
accuracy			0.94	6000
macro avg	0.94	0.94	0.94	6000
weighted avg	0.94	0.94	0.94	6000

(a) Classification Report Logistic Regression Test Set

	precision	recall	f1-score	support
clang	0.94	0.95	0.95	7999
gcc	0.94	0.95	0.94	8022
icc	0.96	0.95	0.96	7979
accuracy			0.95	24000
macro avg	0.95	0.95	0.95	24000
weighted avg	0.95	0.95	0.95	24000

(b) Classification Report Logistic Regression Train Set

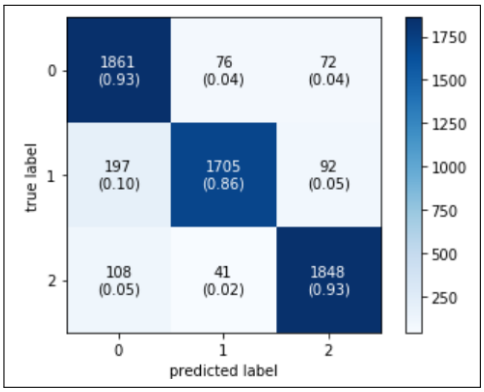
MultinomialNB

In the MultinomialNB model i have not specified any parameter.

Classification Report and Confusion Matrix

	precision	recall	f1-score	support
clang	0.86	0.93	0.89	2009
gcc	0.94	0.86	0.89	1994
icc	0.92	0.93	0.92	1997
accuracy			0.90	6000
macro avg	0.90	0.90	0.90	6000
weighted avg	0.90	0.90	0.90	6000

(a) Classification Report MultinomialNB Test Set



(b) Confusion Matrix Multinomial Naive Bayes Test Set

	precision	recall	f1-score	support
clang	0.93	0.97	0.95	7991
gcc	0.97	0.92	0.95	8006
icc	0.96	0.97	0.97	8003
accuracy			0.96	24000
macro avg	0.96	0.96	0.96	24000
weighted avg	0.96	0.96	0.96	24000

(c) Classification Report MultinomialNB Train Set

Even the multinomial model is good enough to take into consideration but has slightly worse

performance than logistics.
Furthermore making the classification report of the Multinomial it makes a slight overffing and this makes it excluded from being a candidate as a model to solve the multiclass problem.

3.2 Extraction: Choose first word of each instructions of programs

3.2.1 Model

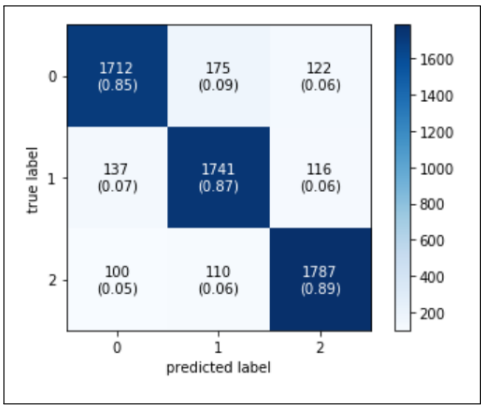
Consider the same models seen in the previous extraction to evaluate which of the two extraction results to be better for the problem.

Logistic Regression

Classification Report and Confusion Matrix

	precision	recall	f1-score	support
clang	0.87	0.85	0.86	2009
gcc	0.86	0.86	0.86	1994
icc	0.89	0.90	0.89	1997
accuracy			0.87	6000
macro avg	0.87	0.87	0.87	6000
weighted avg	0.87	0.87	0.87	6000

(a) Classification Report Logistic Regression Test Set



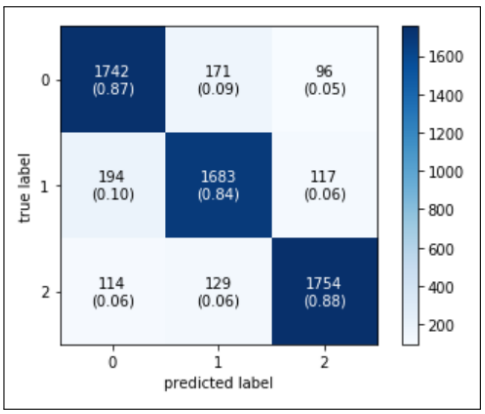
(b) Confusion Matrix Logistic Regression

Also in this case the Logistic Regression is better than the Multinomial Naive Bayes.

Confusion Matrix

	precision	recall	f1-score	support
clang	0.85	0.87	0.86	2009
gcc	0.85	0.84	0.85	1994
icc	0.89	0.88	0.88	1997
accuracy			0.86	6000
macro avg	0.86	0.86	0.86	6000
weighted avg	0.86	0.86	0.86	6000

(a) Classification Report MultinomialNB Test Set



(b) Confusion Matrix Multinomial Naive Bayes

As in the previous case the Logistic Regression turns out to be better than the Multinomial looking at both the accuracy and f1_score, and as the previous case doesn't do overfitting.

We can conclude that to solve the problem I observe that as a model in both cases the Logistic Regression is preferable since it does not overfitting and better results are obtained, and then between the two extractions the first is preferable since having considered the same model with the same parameters get metrics closer to the value 1.

Chapter 4

Blind Test

The blind test contains only input data to classify with 2 models that i have choose for opt and for compiler, respectively binary and multiclass classification.
Extract all the instructions from the dataset, converting them to a list of strings that I will then go to the Vectorizer.
The Vectorizer uses the transform that transform documents to document-term matrix function. Uses the vocabulary and document frequencies learned by fit_transform function.
And after make predictions using first Model, choosen for Binary Classification (Logistic Regression), and second Model, choosen for Multiclass CLassification (Logistic Regression).

Finally we can create the CSV file containing, the various predictions of the first model and the second model, in the following form: <compiler, opt>