Homework 1 - Machine Learning: Compiler Provenance

Edoardo Piroli - 1711234

November 10, 2019

Contents

1 Introduction

1.1 Assignment

The assignment of the homework was to solve two classification problems:

1. **Compiler Provenance**: Given a function in assembly code determine whether it has been compiled using *gcc*, *icc* or *clang*;
2. **Optimization Classification**: Given a function in assembly code determine whether it has been compiled using optimization *'H'*, i.e. high, or *'L'*, i.e. low.

Furthermore the homework required to solve these problems using any supervised model but neural networks.

1.2 Dataset

The provided dataset was formed by 30'000 different functions labelled with both the compiler and the optimization setting used to compile them. In particular, the dataset resulted being:

- Formed by 10'000 instances per compiler;
- Formed by 12'076 instances highly optimized, hence labelled with *'H'*, functions and 17'924 lowly optimized, hence labelled with *'L'*, functions.

In summary, the dataset resulted being perfectly balanced for the first problem(Compiler Provenance) but not balanced for the second one(Optimization Classification).

Every function in the dataset was expressed as a sequence of assembly instructions with the corresponding parameters.

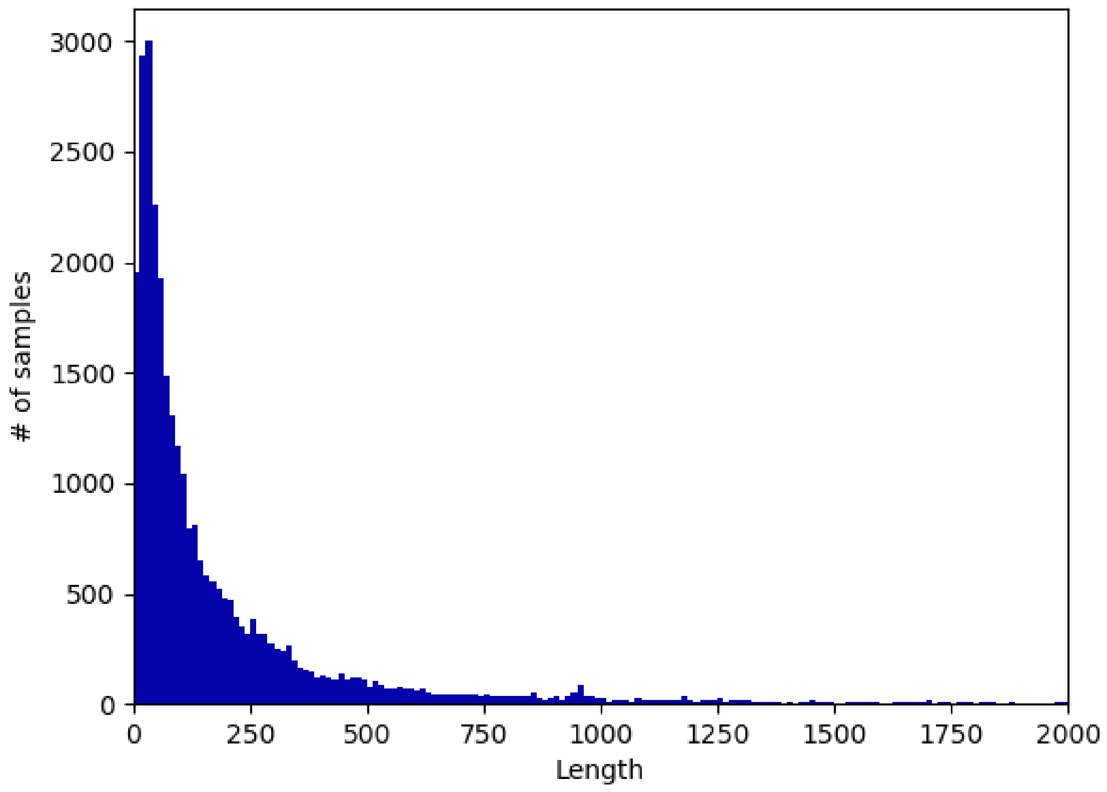


Figure 1: Distribution of functions' lengths in the training dataset

As represented in Figure 1, the functions contained in the dataset are, for the vast majority(76%), sequences of less than 250 instructions. The plot is limited to functions long 2'000 or less instructions, however there are 289 longer ones; in particular the longest one is composed by 21'719 instructions.

2 Experimentation

2.1 Models

For both the problems I have run several experiments training the following supervised models:

- **Naive-Bayes Classifier**: This method is usable in learning tasks where each instance is represented by a conjunction of attribute values. It is based on the very strong assumption that the probability of each sample belonging to one class is conditionally independent on all the other attributes given one of them. I have used the *sklearn.naive_bayes.MultinomialNB()* implementation;
- **Support Vector Machine**: SVMs are based on the idea of finding an hyperplane that best divides a dataset into 2 classes; it relies on the assumption that the samples are linearly separable using the proposed features. This concept is generalized to multiclass classification using the one-against-one approach[1]. I have used the *sklearn.svm.SVC()* implementation.

2.2 Feature extraction

As for the features, I have taken into account only the mnemonics of the instructions, discarding the arguments. In particular I have tried 3 different sets of features:

1. The number of occurrences of every mnemonic in the function;
2. An ordered list of the mnemonics composing the function. In order to use this I had to impose a fixed-length; since the shape of the features must be fixed. In practice, I have splitted longer functions and added padding, as explained below, to shorter ones obtaining all fixed-length functions. I have chosen a fixed-length of 250, because as explained above most of the functions were shorter than that. Once all the functions had been splitted into fixed-length ones I have labelled them with the labels of the functions they were obtained from and trained on all of them. As for prediction on longer sequences I have simply used an average over the predictions of the fixed-length functions.
3. A mixed one containing the length (in mnemonics) of the function along with its first and last 5 mnemonics;

In order to map every mnemonic to one unique integer I have tried 2 different mappings:

- Full mapping: Mapping all the mnemonics in the training dataset;
- Partial mapping: Mapping all the mnemonics which occurred at least 1000 times in the dataset.

Both these mappings included 2 special tokens: '<UNK>' used to handle mnemonics not occurring(or occurring less than 1000 times for the partial mapping) in the dataset and '<PAD>' used to represent padding in short functions.

2.3 Data

In order to be able to evaluate the presented models I have splitted the labelled dataset into 2 smaller ones:

1. The training dataset: containing 85% of the functions;
2. The testing dataset: containing the remaining 15%.

[1]Further details: https://pdfs.semanticscholar.org/a8f7/bb52ebfc291eb23b41ae6994b4865e166ae4.pdf.

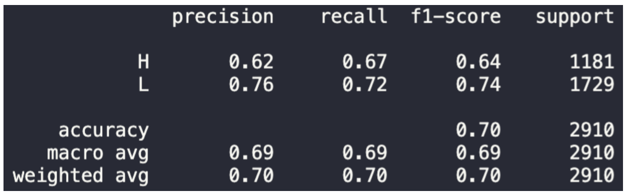
3 Results

3.1 Optimization Classification

3.1.1 Naive-Bayes Classifier

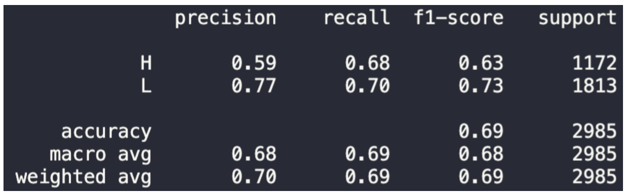
Number of occurrences

- Full Mapping:



```
              precision    recall  f1-score   support

           H       0.62      0.67      0.64      1181
           L       0.76      0.72      0.74      1729

    accuracy                           0.70      2910
   macro avg       0.69      0.69      0.69      2910
weighted avg       0.70      0.70      0.70      2910
```

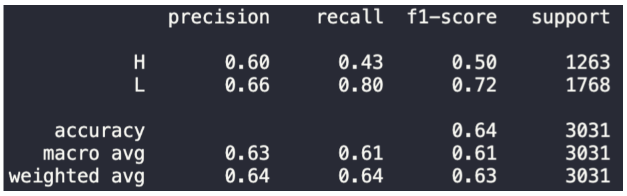
- Partial Mapping



```
              precision    recall  f1-score   support

           H       0.59      0.68      0.63      1172
           L       0.77      0.70      0.73      1813

    accuracy                           0.69      2985
   macro avg       0.68      0.69      0.68      2985
weighted avg       0.70      0.69      0.69      2985
```

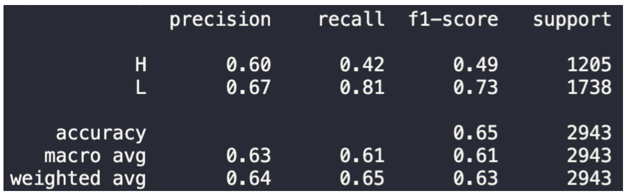
Mixed Features

- Full Mapping:



```
              precision    recall  f1-score   support

           H       0.60      0.43      0.50      1263
           L       0.66      0.80      0.72      1768

    accuracy                           0.64      3031
   macro avg       0.63      0.61      0.61      3031
weighted avg       0.64      0.64      0.63      3031
```

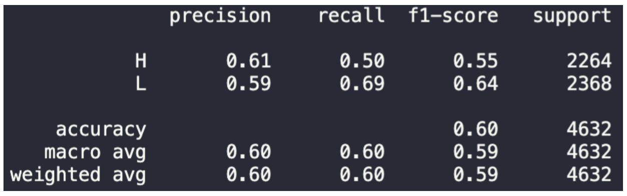
- Partial Mapping



```
              precision    recall  f1-score   support

           H       0.60      0.42      0.49      1205
           L       0.67      0.81      0.73      1738

    accuracy                           0.65      2943
   macro avg       0.63      0.61      0.61      2943
weighted avg       0.64      0.65      0.63      2943
```

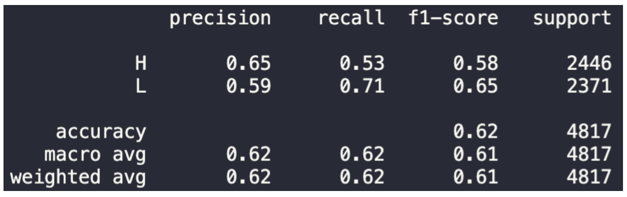
Ordered list of mnemonics

- Full Mapping:



```
              precision    recall  f1-score   support

           H       0.61      0.50      0.55      2264
           L       0.59      0.69      0.64      2368

    accuracy                           0.60      4632
   macro avg       0.60      0.60      0.59      4632
weighted avg       0.60      0.60      0.59      4632
```

- Partial Mapping

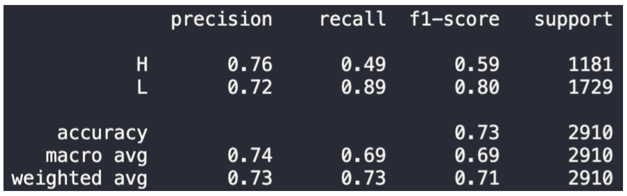


```
              precision    recall  f1-score   support

           H       0.65      0.53      0.58      2446
           L       0.59      0.71      0.65      2371

    accuracy                           0.62      4817
   macro avg       0.62      0.62      0.61      4817
weighted avg       0.62      0.62      0.61      4817
```

3.1.2 Support Vector Machine

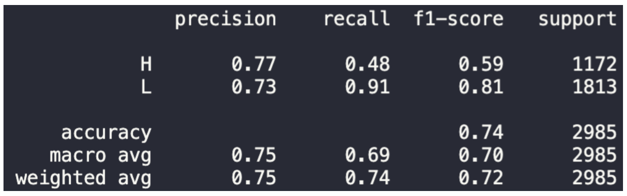
Number of occurrences

- Full Mapping:



```
              precision    recall  f1-score   support

           H       0.76      0.49      0.59      1181
           L       0.72      0.89      0.80      1729

    accuracy                           0.73      2910
   macro avg       0.74      0.69      0.69      2910
weighted avg       0.73      0.73      0.71      2910
```

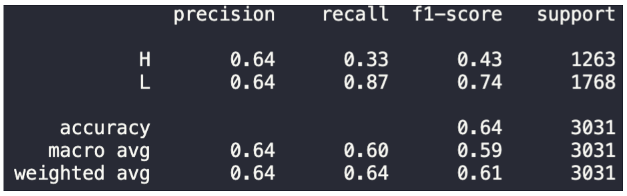
- Partial Mapping



```
              precision    recall  f1-score   support

           H       0.77      0.48      0.59      1172
           L       0.73      0.91      0.81      1813

    accuracy                           0.74      2985
   macro avg       0.75      0.69      0.70      2985
weighted avg       0.75      0.74      0.72      2985
```

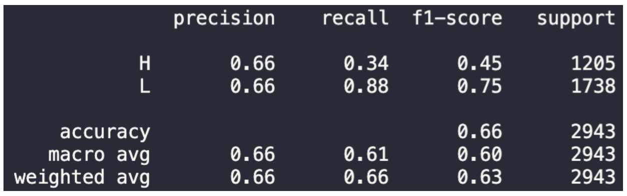
Mixed Features

- Full Mapping:



```
              precision    recall  f1-score   support

           H       0.64      0.33      0.43      1263
           L       0.64      0.87      0.74      1768

    accuracy                           0.64      3031
   macro avg       0.64      0.60      0.59      3031
weighted avg       0.64      0.64      0.61      3031
```

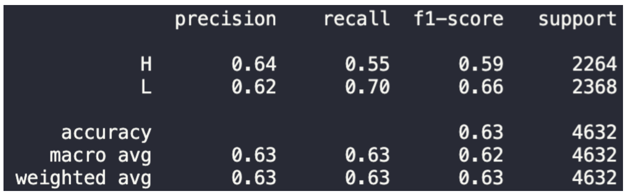
- Partial Mapping



```
              precision    recall  f1-score   support

           H       0.66      0.34      0.45      1205
           L       0.66      0.88      0.75      1738

    accuracy                           0.66      2943
   macro avg       0.66      0.61      0.60      2943
weighted avg       0.66      0.66      0.63      2943
```

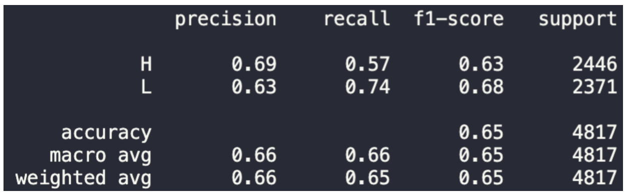
Ordered list of mnemonics

- Full Mapping:



```
              precision    recall  f1-score   support

           H       0.64      0.55      0.59      2264
           L       0.62      0.70      0.66      2368

    accuracy                           0.63      4632
   macro avg       0.63      0.63      0.62      4632
weighted avg       0.63      0.63      0.63      4632
```

- Partial Mapping



```
              precision    recall  f1-score   support

           H       0.69      0.57      0.63      2446
           L       0.63      0.74      0.68      2371

    accuracy                           0.65      4817
   macro avg       0.66      0.66      0.65      4817
weighted avg       0.66      0.65      0.65      4817
```

3.1.3 Comments

As expected SVM performs better, accuracy-wise, than Naive-bayes using the first(number of occurrences) and the third(ordered list of mnemonics) feature selection. In fact, in both cases, the assumption of conditional independence among different attributes is strongly wrong: one instruction by its own will not give much information about whether the whole function was compiled using a highly-optimized option or not, no matter if the ordering of such instructions is taken into account or not. On the other hand, the mixed feature approach seems to perform slightly better when used to train the naive-bayes model; this is probably due to the fact that this feature selection is smaller, hence granting less information to work with to the SVM model, while less-strongly negating the conditional-independence assumption of naive-bayes. In general, the best results, again accuracy-wise, are obtained by both models using the number of occurrences. However, also the SVM model doesn't perform optimally and this is due to the fact that the data is not easily linearly-separable given the proposed feature selections.

One important aspect to take in consideration is the fact that the dataset is not balanced, e.g. one model that simply labels all the samples as 'L', i.e. lowly-optimized, would reach an accuracy of 59.75%. This suggests that the accuracy is not a satisfactory indicator of the performance of the models, in fact, based on the application, we may want a higher precision or recall; although since I'm not really sure which one is to be preferred in this case, I have selected the best model taking into account the f1-score; which is the harmonic mean of the two. In this case, the naive-bayes classifier seems to be more robust to the unbalancedness of the dataset.

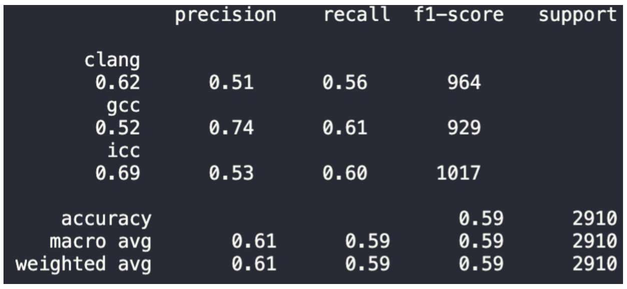
Finally, the usage of full and partial mapping doesn't provide huge shifts in performance of the models in most of the cases, but when it does it almost always works better when using the latter; moreover the models are lighter and faster to train when using it.

3.2 Compiler Provenance

3.2.1 Naive-Bayes Classifier

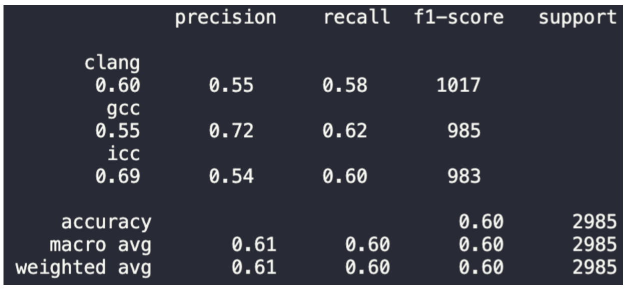
Number of occurrences

- Full Mapping:



```
              precision    recall  f1-score   support

       clang
        0.62      0.51      0.56       964
         gcc
        0.52      0.74      0.61       929
         icc
        0.69      0.53      0.60      1017

    accuracy                           0.59      2910
   macro avg       0.61      0.59      0.59      2910
weighted avg       0.61      0.59      0.59      2910
```

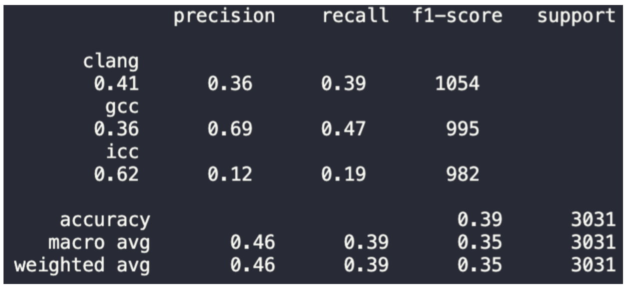
- Partial Mapping



```
              precision    recall  f1-score   support

       clang
        0.60      0.55      0.58      1017
         gcc
        0.55      0.72      0.62       985
         icc
        0.69      0.54      0.60       983

    accuracy                           0.60      2985
   macro avg       0.61      0.60      0.60      2985
weighted avg       0.61      0.60      0.60      2985
```

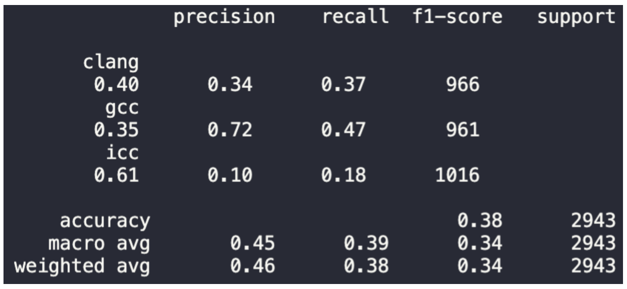
Mixed Features

- Full Mapping:



```
              precision    recall  f1-score   support

       clang
        0.41      0.36      0.39      1054
         gcc
        0.36      0.69      0.47       995
         icc
        0.62      0.12      0.19       982

    accuracy                           0.39      3031
   macro avg       0.46      0.39      0.35      3031
weighted avg       0.46      0.39      0.35      3031
```

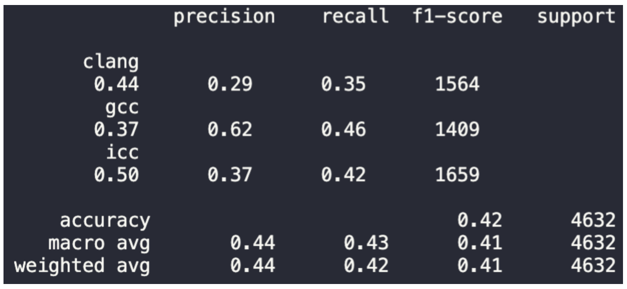
- Partial Mapping



```
              precision    recall  f1-score   support

       clang
        0.40      0.34      0.37       966
         gcc
        0.35      0.72      0.47       961
         icc
        0.61      0.10      0.18      1016

    accuracy                           0.38      2943
   macro avg       0.45      0.39      0.34      2943
weighted avg       0.46      0.38      0.34      2943
```

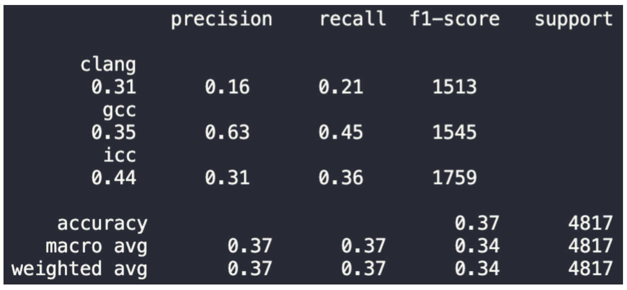
Ordered list of mnemonics

- Full Mapping:



```
              precision    recall  f1-score   support

       clang
        0.44      0.29      0.35      1564
         gcc
        0.37      0.62      0.46      1409
         icc
        0.50      0.37      0.42      1659

    accuracy                          0.42      4632
   macro avg       0.44      0.43      0.41      4632
weighted avg       0.44      0.42      0.41      4632
```

- Partial Mapping

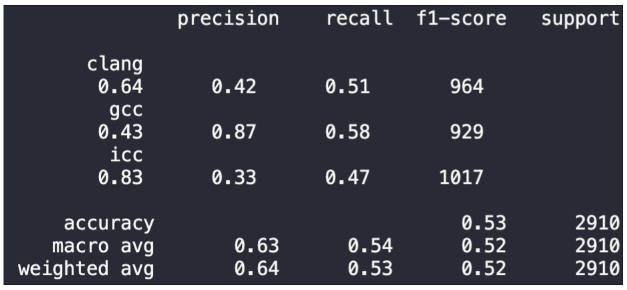


```
              precision    recall  f1-score   support

       clang
        0.31      0.16      0.21      1513
         gcc
        0.35      0.63      0.45      1545
         icc
        0.44      0.31      0.36      1759

    accuracy                          0.37      4817
   macro avg       0.37      0.37      0.34      4817
weighted avg       0.37      0.37      0.34      4817
```

3.2.2 Support Vector Machine

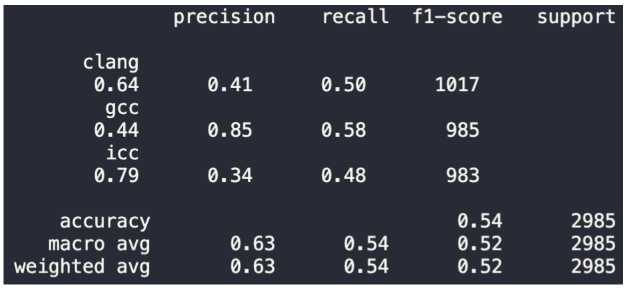
Number of occurrences

- Full Mapping:



```
              precision    recall  f1-score   support

       clang
        0.64      0.42      0.51       964
         gcc
        0.43      0.87      0.58       929
         icc
        0.83      0.33      0.47      1017

    accuracy                          0.53      2910
   macro avg       0.63      0.54      0.52      2910
weighted avg       0.64      0.53      0.52      2910
```

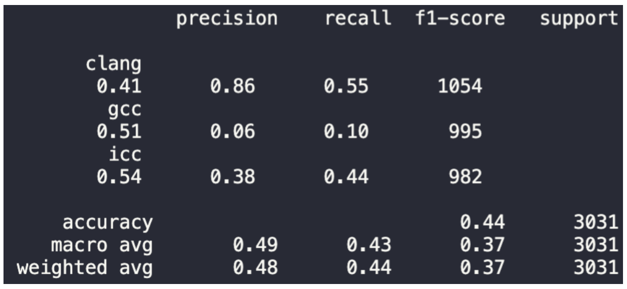
- Partial Mapping



```
              precision    recall  f1-score   support

       clang
        0.64      0.41      0.50      1017
         gcc
        0.44      0.85      0.58       985
         icc
        0.79      0.34      0.48       983

    accuracy                          0.54      2985
   macro avg       0.63      0.54      0.52      2985
weighted avg       0.63      0.54      0.52      2985
```

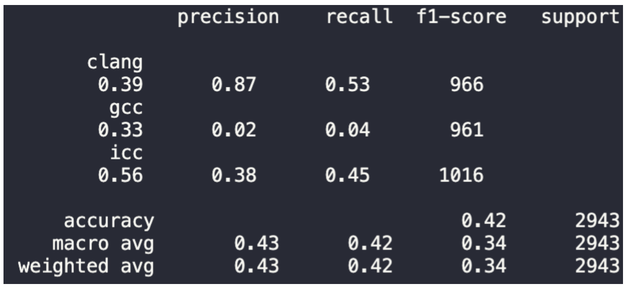
Mixed Features

- Full Mapping:



```
              precision    recall  f1-score   support

       clang
        0.41      0.86      0.55      1054
         gcc
        0.51      0.06      0.10       995
         icc
        0.54      0.38      0.44       982

    accuracy                           0.44      3031
   macro avg       0.49      0.43      0.37      3031
weighted avg       0.48      0.44      0.37      3031
```

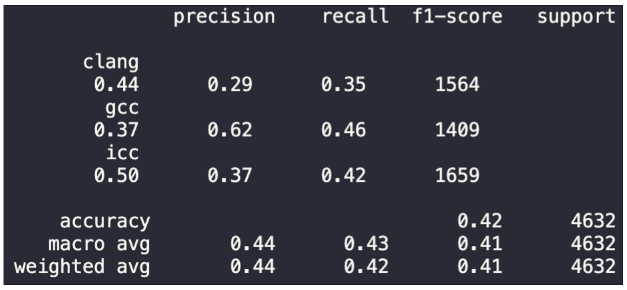
- Partial Mapping



```
              precision    recall  f1-score   support

       clang
        0.39      0.87      0.53       966
         gcc
        0.33      0.02      0.04       961
         icc
        0.56      0.38      0.45      1016

    accuracy                           0.42      2943
   macro avg       0.43      0.42      0.34      2943
weighted avg       0.43      0.42      0.34      2943
```

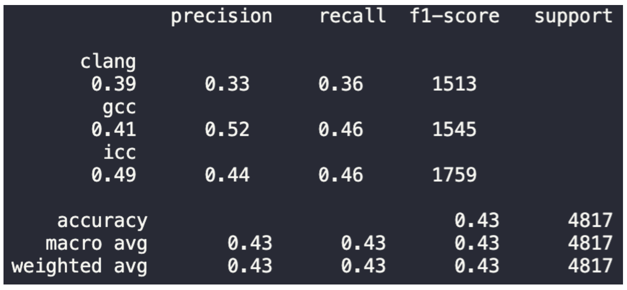
Ordered list of mnemonics

- Full Mapping:



```
              precision    recall  f1-score   support

       clang
        0.44      0.29      0.35      1564
         gcc
        0.37      0.62      0.46      1409
         icc
        0.50      0.37      0.42      1659

    accuracy                           0.42      4632
   macro avg       0.44      0.43      0.41      4632
weighted avg       0.44      0.42      0.41      4632
```

- Partial Mapping



```
              precision    recall  f1-score   support

       clang
        0.39      0.33      0.36      1513
         gcc
        0.41      0.52      0.46      1545
         icc
        0.49      0.44      0.46      1759

    accuracy                           0.43      4817
   macro avg       0.43      0.43      0.43      4817
weighted avg       0.43      0.43      0.43      4817
```

3.2.3 Comments

For this problem, the Naive-bayes classifier performed better than SVM using the first(number of occurrences) and the second(mixed features) feature selections; this is probably due to the fact that the Multiniomal naive-bayes classifier generalizes better to multiclass than the SVM with the one-against-one approach does.

As in the optimization classification problem, the overall best feature selection resulted being the number of occurrences by a large margin. This is probably correlated to the fact that one compiler tends to use some particular instructions to a larger extent than the others.

Also the different results of partial and full mapping reflected this aspect, in fact most models performed worse when using the partial model; probably due to the fact that some rare instructions are used almost exclusively by one of the compilers.

Finally, it is interesting to note how the selection of the mixed features performed very poorly with respect to two different classes for the two models; in particular naive-bayes had a very bad f1-score w.r.t. the *icc* compiler and SVM w.r.t. the *gcc* compiler.

3.3 Best Model

In summary, the best model resulted being for both the problems the naive-bayes trained using the number of occurrences as features with the partial mapping.

4 Conclusions

The obtained results are far from ideal and further work is required to improve them, some possible approaches might be:

1. Explore different learning methods. In particular, I think RNNs would probably outperform both SVM and Naive-Bayes by a huge margin;
2. Run GridSearch or RandomSearch to optimize the hyperparameters:
   - Number of minimum occurrences of one mnemonic to be inserted in the mapping;
   - Fixed-length of the sequences;
   - Number of instructions to take at the start and at the end of the functions for the mixed approach.
3. Perform further feature engineering. This will require more knowledge on the domains of the problems.