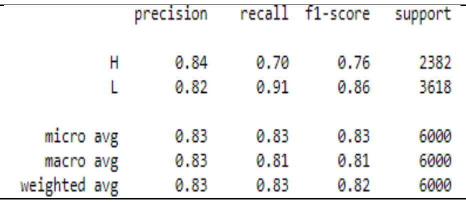
ML - Homework1 - Compiler Provenance

Theodoros Sofianos 1867968

Preprocessing: We will consider only the mnemonics of each instruction, which is essentially the first word for each set of instructions, for running time purposes. The mnemonics, optimizer and compiler will be written in a dataframe, which afterwards will be stored as a csv file, so there is no need to run it every time. Then, we will vectorize the mnemonics of each file, so we will be able to process it. The feature extraction will be done by using only the method of hashingvectorizer, since it scales for big datasets and creates a sparse matrix that doesn't consume much memory. Based on intuition, I chose to create tokens consisting of 5,6 and 7 respectively words-mnemonics, so we can 'capture' the order of the mnemonics. I will later evaluate this choice based on the results. So, eventually the mnemonics of each file will be represented with a sparse vector, where in every cell there is the hashed value of the number of occurrences of the specific token. By using the default number of fatures when creating the hashingvectorizer, we can see that after the transformation we have 1048576 columns, something that will make some machine learning methods computationally expensive. Now, the only thing missing is to split the dataframe in train set and test set. I chose to split the 30000 original files into 24000 for the training set and 6000 for the test set, using also random state for reproducibility.

Evaluation of machine learning models for optimizer:

Name of method	Confusion Matrix	Classification report	Accuracy (test set)	Weighted f1 score (cross-valid.)
Logistic Regression	H L H [1669 713 L 324 3294]	 precision recall f1-score support H 0.84 0.70 0.76 2382 L 0.82 0.91 0.86 3618 micro avg 0.83 0.83 0.83 6000 macro avg 0.83 0.81 0.81 6000 weighted avg 0.83 0.83 0.82 6000	0.8271666666666667	0.822 (+/- 0.01)

Logistic Regression (After grid search on penalty and regularization parameter)	H L H [1893 489 L 369 3249]	precision recall f1-score support H 0.84 0.80 0.82 2382 L 0.87 0.90 0.88 3618 micro avg 0.86 0.86 0.86 6000 macro avg 0.85 0.85 0.85 6000 weighted avg 0.86 0.86 0.86 6000	0.857	0.856 (+/- 0.004)
Linear Support Vector Machines	H L H [1796 586 L 322 3296]	precision recall f1-score support H 0.85 0.75 0.80 2382 L 0.85 0.91 0.88 3618 micro avg 0.85 0.85 0.85 6000 macro avg 0.85 0.83 0.84 6000 weighted avg 0.85 0.85 0.85 6000	0.8486666666666667	Not computed, running time purposes
Decision Tree with gini impurity loss	H L H [1674 708 L 517 3101]	[[1674 708] [517 3101]] precision recall f1-score support H 0.76 0.70 0.73 2382 L 0.81 0.86 0.84 3618 micro avg 0.80 0.80 0.80 6000 macro avg 0.79 0.78 0.78 6000 weighted avg 0.79 0.80 0.79 6000	0.7958333333333333	0.789 (+/- 0.01)
Decision tree with entropy loss	H L H [1662 720 L 536 3082]	precision recall f1-score support H 0.76 0.70 0.73 2382 L 0.81 0.85 0.83 3618 micro avg 0.79 0.79 0.79 6000 macro avg 0.78 0.77 0.78 6000 weighted avg 0.79 0.79 0.79 6000	0.7906666666666666	0.787 (+/- 0.01)
Random Forest	H L H [1706 676 L 341 3277]	precision recall f1-score support H 0.83 0.72 0.77 2382 L 0.83 0.91 0.87 3618 micro avg 0.83 0.83 0.83 6000 macro avg 0.83 0.81 0.82 6000 weighted avg 0.83 0.83 0.83 6000	0.8305	0.828 (+/- 0.01)
K-NN	H L H [1338 994 L 1032 2586]	precision recall f1-score support H 0.57 0.58 0.58 2382 L 0.72 0.71 0.72 3618 micro avg 0.66 0.66 0.66 6000 macro avg 0.65 0.65 0.65 6000 weighted avg 0.66 0.66 0.66 6000	0.6623333333333333	0.662 (+/- 0.01)
Multinomial Naïve Bayes	H L H [1402 980 L 305 3313]	precision recall f1-score support H 0.82 0.59 0.69 2382 L 0.77 0.92 0.84 3618 micro avg 0.79 0.79 0.79 6000 macro avg 0.80 0.75 0.76 6000 eighted avg 0.79 0.79 0.78 6000	0.7858333333333334	0.775 (+/- 0.01)

Explaining evaluation procedures for optimizer:

First of all, we know that the optimization distribution is not balanced. Indeed, we have almost 60% of the optimizers in the original dataset being of type Low and in our test set, we have 2382 instances of high optimization and 3618 of low optimization. Thus, the baseline estimator that guesses always the optimizer to be low, will have accuracy of 3618/6000=0.603. For unbalanced distributions, we know that accuracy is not the best estimator. That's why we will be considering the f1 metric, weighted by the support of each class, as our result in order to compare the models. The evaluation of the models in the above table includes the confusion matrix, classification report and accuracy score on the test set, consisting of 6000 instances and the average weighted f1 score over 5 folds Stratified validation, with shuffling. Since the dataset is imbalanced, we use the stratified cross validation so we get 5 folds while maintain the percentage of each class in every fold(around 60% of the instances will have low optimization, just like the original dataset.) Also, it should be mentioned that for creating the naïve bayes model, we added the command non_negative=True to the hashingvectorizer function, something that changed some hashed values of our matrix. By looking at the table above, we see that most models perform well both on the test set and on the cross-validation, which shows that the decision to create n_grams of 5,6 and 7 mnemonics was sound, although probably not the optimal one. However, since there is not a threshold value that must be surpassed, we will accept the current settings of the feature selection and we will not try different combinations of n_grams , number of features or other feature extraction models, like countvectorizer.

Explaining results and selection of the best model:

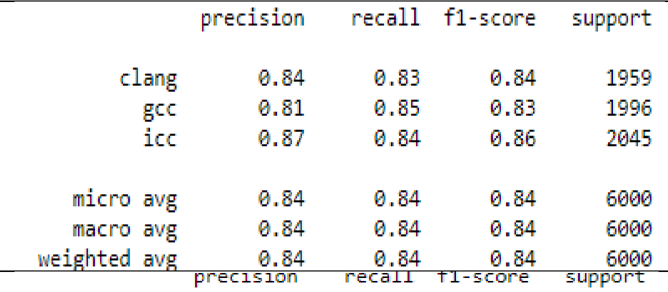
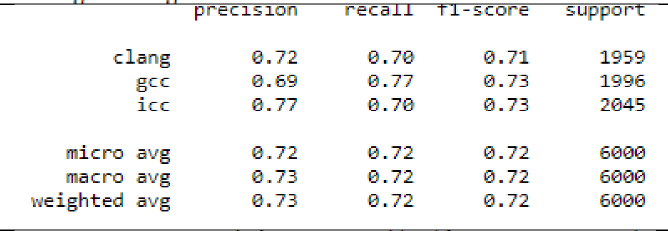
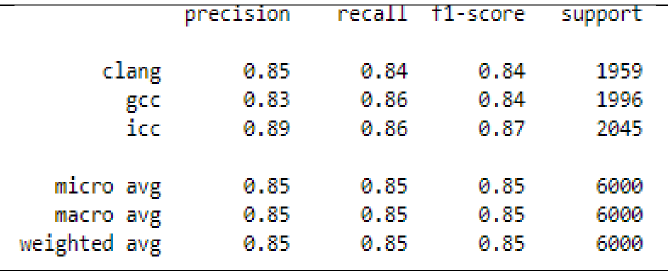
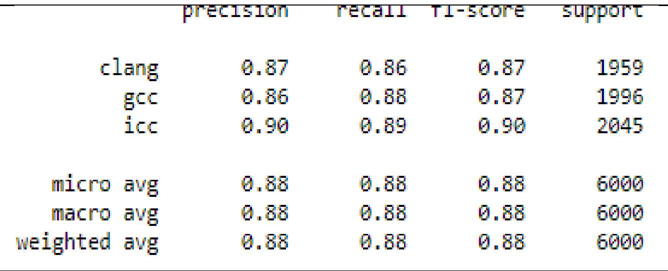
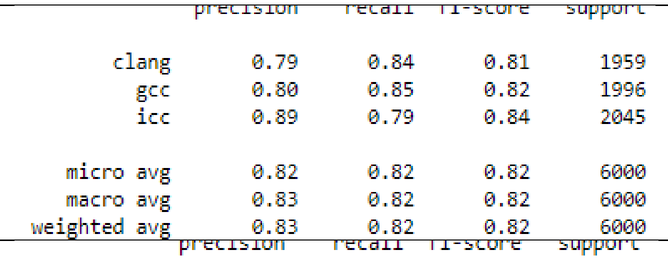
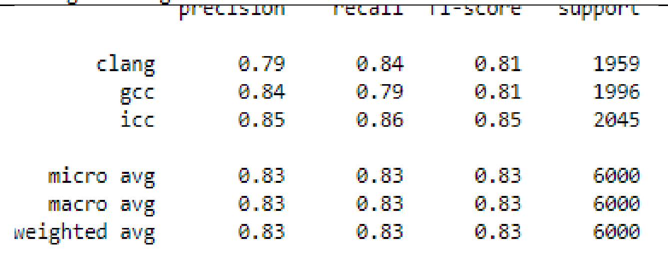
We can observe from the table above that the majority of the models give us good estimators, with small variance in the stratified k folds validation. We also notice that the accuracy of the test set is very close to the f1 score. However, we also notice another 'pattern'. The optimizer H, which is the minority class, has relatively small values of recall(around 0.7 for most of the models) and that is biased. In other words, the estimators with probability 30% will miss the actual class of H and predict L instead(majority class). So the estimators seem to have a bias towards predicting the majority class more times than they should. In order to deal with this issue, we performed a grid search on the logistic regression model, based on the weighted f1 score, to tune in the penalty and regularization parameters. After the grid search, we see that the recall for High optimization increased from 70% to 80%, with a small decrease of the recall of the Low optimization by 1%. Also, the precision of L increased by 5% and all these changes are depicted in the f1 score, which was increased from 0.822 to 0.856 on average on the 5 folds validation. Thus, the estimator of logistic regression after the grid search has much smaller biased compared to the other estimators, and since all estimators, with logistic regression included, have small variance, we can claim that this is the best model to predict the binary classification problem of optimization method. We also see that the support vector machines model performs well on the test set and probably with a grid search we can reduce its bias, like how we did with logistic regression. However, since our sparse matrix has around 1 million columns, the performance of SVM is not optimal and that's why the weighted f1 score on stratified k-fold validation wasn't calculated. We also notice that the recall of the L optimization on the test set with SVM is the best compared to the other models, so we can be almost sure that we can decrease further the bias of the estimator, but for performance reasons we choose the logistic regression model, that after the grid search performs well. Also, since the linear models performed well on this classification problem, I will not try SVM with different kernels, since data seem to be linearly seperable. As far as the other models concerned, we see that the 2 decision trees with the different loss criteria perform pretty much the same, with some very small differences, perhaps due to randomness. Random forest seems to perform well, similar to logistic regression before the grid search, but its performance is worse, so its ruled out. I also tried a knn approach, with bad results, since the accuracy of the model marginally surpass the performance of the baseline estimator. We could try to improve its hyperparameters, like number of neighbors, but it seems that non-parametric models are not adequate for this classification. Finally, the Naïve Bayes has very small recall for the High optimization and thus is more biased than the other models. It also has smaller precision on predicting L, but it has the highest recall for this class.

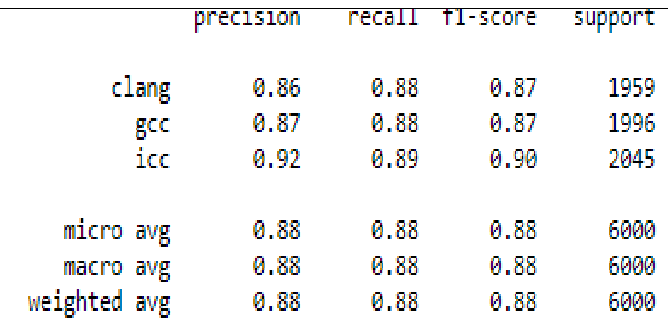
Explaining evaluation procedures for compiler:

In this multiclass classification problem, we have a balanced dataset, with 10000 instances per compiler. Thus, the analysis will be done by using the overall accuracy for all classes, since no anomalies were detected. Also, we will be performing a Shuffle split validation, with 5 splits and the validation size being 0.333 of the total dataset every time. Since the classes are balanced, we can be sure that almost all classes will be represented in the validation splits and their support size will not be dramatically different. Again, for being able to use the Naïve Bayes classifier, we will change only for this model the hashingvectorizer function, by adding the parameter non_negative to be True.

Evaluation of machine learning models for compiler:

Name of method	Confusion Matrix	Classification report	Overall Accuracy(cross-validation)

Logistic Regression	clang gcc icc clang [1623 233 103 gcc 151 1701 144 icc 153 176 1716]	 precision recall f1-score support clang 0.84 0.83 0.84 1959 gcc 0.81 0.85 0.83 1996 icc 0.87 0.84 0.86 2045 micro avg 0.84 0.84 0.84 6000 macro avg 0.84 0.84 0.84 6000 weighted avg 0.84 0.84 0.84 6000	0.832 (+/- 0.01) s
Decision Tree	clang gcc icc clang [1379 358 222 gcc 247 1536 213 icc 285 336 1324]	 precision recall f1-score support clang 0.72 0.70 0.71 1959 gcc 0.69 0.77 0.73 1996 icc 0.77 0.70 0.73 2045 micro avg 0.72 0.72 0.72 6000 macro avg 0.73 0.72 0.72 6000 weighted avg 0.73 0.72 0.72 6000	0.712 (+/- 0.01
Linear SVM(one vs all)	clang gcc icc clang [1652 213 94 gcc 155 1710 131 icc 147 147 1751]	 precision recall f1-score support clang 0.85 0.84 0.84 1959 gcc 0.83 0.86 0.84 1996 icc 0.89 0.86 0.87 2045 micro avg 0.85 0.85 0.85 6000 macro avg 0.85 0.85 0.85 6000 weighted avg 0.85 0.85 0.85 6000	Not computed for running time purposes
Linear SVM(one vs one)	clang gcc icc clang [1691 179 89 gcc 130 1755 111 icc 112 114 1819]	 precision recall f1-score support clang 0.87 0.86 0.87 1959 gcc 0.86 0.88 0.87 1996 icc 0.90 0.89 0.90 2045 micro avg 0.88 0.88 0.88 6000 macro avg 0.88 0.88 0.88 6000 weighted avg 0.88 0.88 0.88 6000	Not computed for running time purposes
Random Forest	clang gcc icc clang [1644 244 71 gcc 189 1688 119 icc 250 185 1610]	 precision recall f1-score support clang 0.79 0.84 0.81 1959 gcc 0.80 0.85 0.82 1996 icc 0.89 0.79 0.84 2045 micro avg 0.82 0.82 0.82 6000 macro avg 0.83 0.82 0.82 6000 weighted avg 0.83 0.82 0.82 6000	0.817 (+/- 0.01)
Multinomial Naïve Bayes	clang gcc icc clang [1641 188 130 gcc 239 1572 185 icc 188 102 1755]	 precision recall f1-score support clang 0.79 0.84 0.81 1959 gcc 0.84 0.79 0.81 1996 icc 0.85 0.86 0.85 2045 micro avg 0.83 0.83 0.83 6000 macro avg 0.83 0.83 0.83 6000 weighted avg 0.83 0.83 0.83 6000	0.824 (+/- 0.01)

Multinomial Naïve Bayes(after grid search on parameters prior distribution and alpha)	clang gcc icc clang [1716 167 76 gcc 147 1758 91 icc 124 107 1814]		0.884 (+/- 0.004)

```
             precision    recall  f1-score   support

       clang       0.86      0.88      0.87      1959
         gcc       0.87      0.88      0.87      1996
         icc       0.92      0.89      0.90      2045

   micro avg       0.88      0.88      0.88      6000
   macro avg       0.88      0.88      0.88      6000
weighted avg       0.88      0.88      0.88      6000
```

Explaining results and selection of the best model:

Starting with the logistic regression model, we get an estimator with low bias, since the precision and recall is bigger than 80% for all the classes and we can be sure that the model has good predicting ability. The decision tree model significantly underperforms compared to logistic regression, since its accuracy in the 5 splits is more than 10% lower. I tried 2 methods of support vector machines, one vs all and one vs one. The one vs all approach produces somewhat similar results with the logistic regression, slightly better in terms on f1 score and accuracy. The one vs one approach produces the best results on the test set, compared to all the other models. Its not biased and the accuracy reaches 90%. However, since the decision was to create the sparse matrix with the default number of features(more than 1 million columns), support vector machines have performance disadvantages, especially for computing the overall average accuracy and the variation of the estimator on 5 splits validation. Thus, we will rule out both version of SVM and we will try to approach their results with other, more scalable methods. Random forest again seems like a decent option, but its outperformed by the logistic regression in terms of f1 score, overall average accuracy and performance. Finally, I tried a Naïve Bayes model. It is slightly outperformed by the logistic regression in terms of accuracy on the 5 splits, but since it has by far the best scalability and running time performance, I performed a grid search on the normalizing alpha value. Since the data is balanced, the distribution of the classes is practically uniform, that's why the argument fit_prior makes absolutely no difference. We see that after tuning in the alpha parameter, the overall average accuracy for all classes in the 5 splits improves by roughly 6%

and the f1 score and accuracy on the test set is only comparable to the one vs one SVM. If we take into account its results but also the scalability of this method for big data, its clear that it’s the best model compared with the others. It also has the estimator with the smallest variance, 0.4%, so its clearly the best model in terms of bias, variance and performance.

Comments:

There are many things that can be done for further optimization of the 2 classification problems. We can try different combinations of n_grams in the hashingvectorizer or try other methods of feature extraction and see how much they help the models get better results. Also, we can try different number of features. If by increasing the number of features improves significantly the results, then it should be done, but if a smaller number of features produces pretty much the same results we should consider it. With smaller number of features, we could also use the support vector machines models that are very powerful in terms of predicting ability, but do not scale so well for huge datasets. We can also perform grid search on more models and see if their hyperparameters’ tuning give us better results.