Does Feature Selection Improve Classification? A Better, but smaller, Large Scale Experiment in OpenML

Koen Ponse s1861581
Ivan Horokhovskyi s3069176
Ernest Vanmosuinck s3210359

Abstract

This report aims to answer the question of whether feature selection methods will improve the predictive performance and computation time of a range of well-known machine learning models. We conduct a large scale experiment through OpenML on a subset of the OpenML-CC18 dataset, a well-curated classification dataset. We find that feature selection can greatly decrease training and prediction time, but does little in terms of predictive performance gain. In fact, it may instead hurt performance slightly.

1 Introduction

An important part of the machine learning pipeline is preprocessing the data. Many papers claim an important preprocessing step is feature selection [1, 2, 3]. Feature selection, or dimensionality reduction, is selecting a subset of the available features according to some selection criterion. Feature selection can in theory have many benefits. Examples of this are improving the interpretability, reducing training time due to the lower amount of features, and even improving test accuracy due to a lower risk of overfitting.

The goal of this paper is to answer the question of whether feature selection makes a difference in both the training time and predictive performance. Similar work has been produced by Martijn J. Post et al. [4], but it lacked investigation in training time and also did not optimize hyperparameters. Our experiments are conducted on the OpenML-CC18 [5], a set of 72 curated classification datasets published through OpenML [6, 7]. We will use a variety of 5 different well-known and popular classification algorithms. Hyperparameter optimization is done by means of Bayesian optimization, as it is currently considered the best approach for finding good hyperparameters in a relatively short amount of time. The final code can be found on our GitHub[1].

Our report is structured as follows. In Section 3 we describe our choice of algorithms and methods. We also explain our experimental setup. In Section 4 we show our experimental results. And finally in Section 5 we draw conclusions.

In addition it is worth mentioning that we had limited time and computation resources this is why we needed to make some sacrifices to get as much results as posible for the limits we had.

[1] `https://github.com/ernestvmo/feature_selection`

2 Related Works

As explained in the introduction of this paper, feature selection has many benefits; increasing readability, reducing training time, and even improved accuracy and lowering the risk of overfitting. The problem of feature selection can be looked at in the form of a search problem, with a search space and an objective. Performing an exhaustive search on all of the features combination is naturally impractical, especially when we consider large datasets, hence different methods have to be considered.
Chandrashekar and Sahin [8] proposed different feature selection methods tested on two supervised learning algorithms Support Vector Machine (SVM) and Radial Basis Function Network (RBF)) using seven different datasets.
Martijn J. Post et al. [4] looked at the benefit of using feature-selection on a variety of algorithms. Their research is made available on OpenML. Their research stands as baseline for this paper, and we decided to extend their research by investigating the training time performance, and applying Bayesian optimization on the models' hyperparameters before applying feature selection.

3 Methods

For our experiments, we have chosen 5 different commonly used classification algorithms. We picked these algorithms firstly because they represent different alogorithm families like boosting, tree-based, linear, metrics-based, probabilistic and even simple neural network. Secondly, some of them are state-of-the-art methods and some are more simple but have their own benefits (like speed, interpretability and also require much less training data), nevertheless, all of them are widely used in production systems and in addition they are implemented in Scikit-learn [9], which is a widely used machine learning toolkit which in turn allows for good reproducibility of our results. Scikit-learn is open source, gives a wide variety of models, has C++ backend and is well optimized which gives good performance and is well involved in the python data science ecosystem working out of the box with numpy, pandas, scipy and data visualisation instruments. The algorithms used and their corresponding hyperparameter search spaces are listed in Table 1.
We have chosen these algorithms due to them being well known and used. Furthermore, we wanted to explore a wide range of different methods, namely linear models, boosting methods, a neighbour method, a decision tree and an ensemble method.

We use Bayesian optimization for optimizing the hyperparameters of each algorithm. The Bayesian optimization internally uses 3-fold cross-validation and 50 iterations. 3-fold cross-validation is not the standard here, but allowed for good enough results and saved us quite a bit of time, which in turn allowed us to get more results. To compare the final results after optimization, we have used 10-fold cross-validation as it is a robust way to measure the performance of a model.

We considered different types of feature selection algorithms. The first considered type is statistical filter-based feature selection, it is simple but needs to tune some hyperparameters like actual test and threshold, different recommendations can be observed on Figure 1, the main disadvantage of this method is that it can not see any relations between variables which is often important. Next type of feature selection is recursive feature elimination the estimator is trained on the initial set of

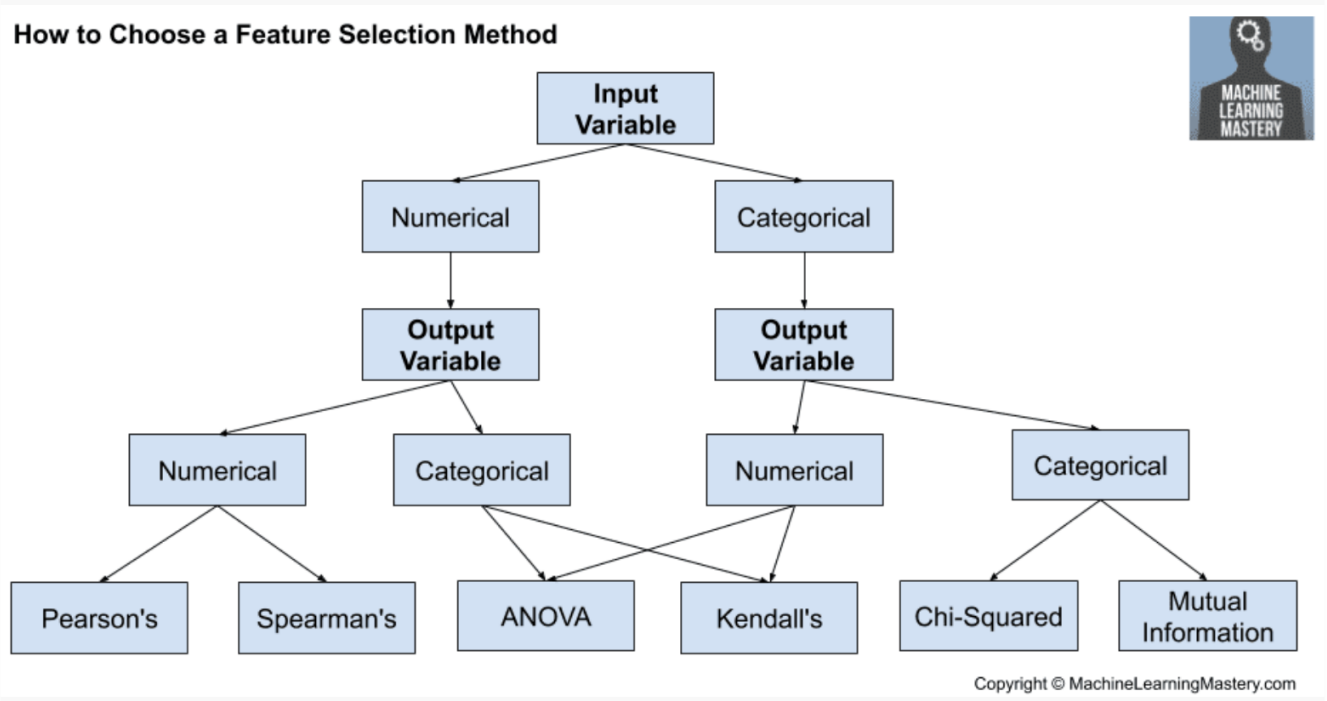


Figure 1: Statistical feature selection

features and the importance of each feature is obtained through the models feature importance attribute from Scikit-learn. After that, the least important features are pruned from the current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached. Such a method obviously requires a lot of time and moreover is a greedy method which means it can stuck in local optimums or overfit. The last considered feature selection type is embedded methods and are based of feature importance after passing through a machine learning model usually either linear or random forest on gradient boosting model, this is relatively fast, allows to take into the account dependencies between features and can work on any dataset.

Based on the considerations above as our feature selection method, we have opted to use an ensemble method to calculate impurity-based feature importance, which can then be used to select relevant features. Moreover, this model is implemented in Scikit-learn. Which has the benefits we described earlier. Furthermore, we have only chosen a single method as it was found that different feature selection methods only lead to marginally different results [4].

Our experiments are conducted by testing each algorithm on each dataset twice, once with feature selection and once without feature selection. Before both runs, the hyperparameters are optimized with Bayesian optimization which is only trained on a random subset of an arbitrarily chosen 72% of the total data, such split allows us to assure that our model has not overfitted and generalized dataset features correctly. The final score is then evaluated by means of 10-fold cross-validation for both of the runs. During the cross-validation, we measure the accuracy, precision, recall, f1 and Area under the ROC curve. Because we have 10 measurements, due to the 10 folds, we can do

statistical analysis to test whether our results are statistically significant. Furthermore, we measure the time it took both runs to complete in seconds.

Table 1: Used Algorithms and the hyperparameter spaces used for the hyperparameter optimization process.

Algorithm	**Hyperparameter space**
Decision Tree	criterion: [gini, entropy], max_depth: [1, 1000, uniform], min_samples_split: [1, 100, uniform], min_samples_leaf: [1, 100, uniform], max_features: [auto, log2]
GradientBoosting	learning_rate: [1e-2, 1e2], n_estimators: [1, 500], subsample: [0.1, 1], criterion: [friedman_mse, squared_error], min_samples_split: [0.01, 1], min_samples_leaf: [0.001, 0.5, uniform], max_depth: [1, 100], tol: [1e-4, 1e-2, log-uniform]
K-nearest neighbor (kNN)	n_neighbors: [1, 20, uniform], weights: [uniform, distance], algorithm: [auto, ball_tree, kd_tree, brute], p: [1, 3, uniform]
Logistic Regression	penalty: [elasticnet, l1, l2], C: [1e-4, 10000, log-uniform], solver: [newton-cg, liblinear], tol: [1e-5, 1e-3, uniform]
Random Forest	n_estimators: [50, 500, uniform], criterion: [gini, entropy], max_features: [auto, log2], max_depth: [1, 1000, uniform], min_samples_split: [1, 100, uniform], min_samples_leaf: [1, 100, uniform]

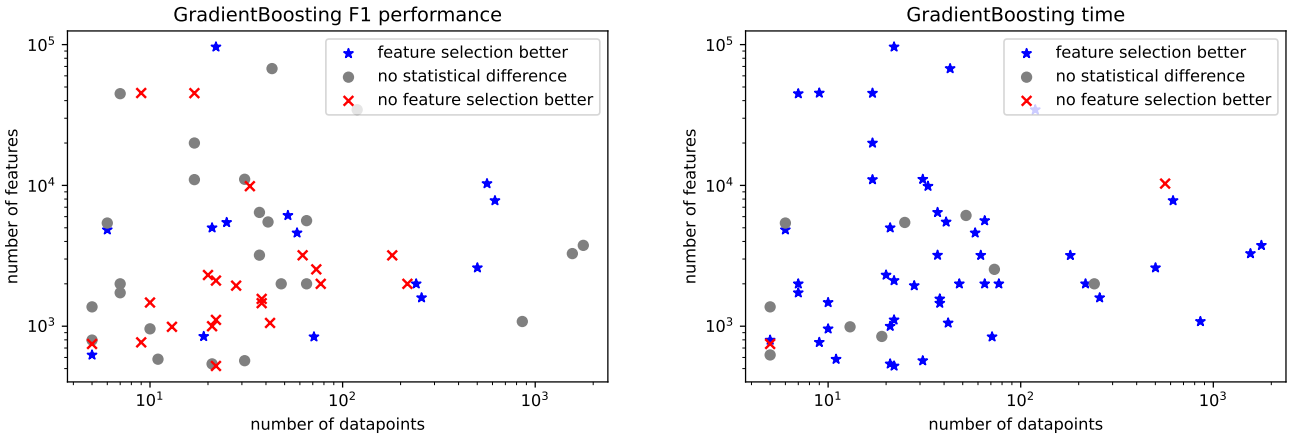
4 Experiments

For experiments below, we ran as many datasets as possible on each of the algorithms. We measured the training time, testing time and f1 scores during 10-fold cross-validation both before and after feature selection. The time taken for the feature selection itself, is taken into account in the training time of the feature selection method. For the statistical tests, we use the Wilcoxon statistical test as it was successfully used in similar work [10]. The statistical test is applied to each of the 10 folds of the cross-validation from before and after the feature selection.
The remaining parts of this Section is outlined as follows. First, we will briefly describe the performance difference between no feature selection and feature selection applied for each of the 5 algorithms. Then we will summarise these results for the predictive performance and time difference in terms.

4.0.1 Gradient Boosting

Looking at figure 2, we can view the performance of the GradientBoosting based on the F1 score and total time. When it comes to F1 performance (figure 2a and table 2), it is unclear if using feature selection really improves the results. Using feature selection had a noticeable impact on the time performance for GradientBoosting. Excluding two datasets, scores with feature selection are all better (figure 2b and table 2). Overall, using feature selection along with GradientBoosting lead

to a F1 performance 3.30% poorer than **without** feature selection, but it lead to an improvement of 55% in total time.



(a) F1 performance

(b) Training & Evaluation time

Figure 2: Whether GradientBoosting performs better on various datasets with or without feature selection or whether there is no statistical difference found.

Table 2: GradientBoosting performance on all datasets. The cells in the first two rows indicate how many datasets performed better or worse with feature selection, and in how many datasets a statistical difference was found. The last columns indicate the average improvement by feature selection in both performance and time.

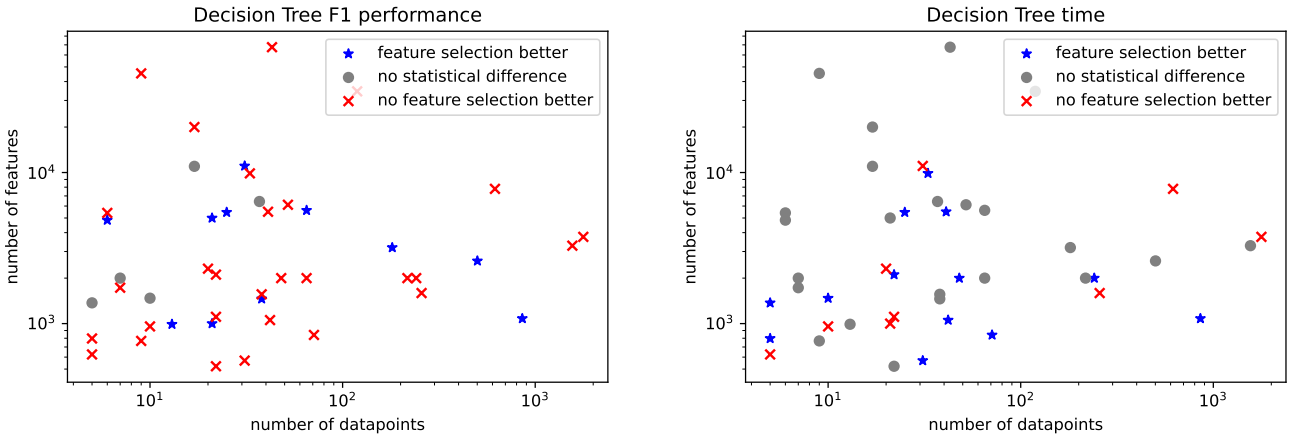
	Absolute		Statistical difference	
	Better	Worse	Yes	No
Performance	21	38	38	21
Time	48	11	57	2
Avg f1 improvement	-3.308%			
Avg Time improvement	55.000%			

4.0.2 Decision Tree

The results for decision tree a bit stands out from other models results. It was assumed that such behaviour was related to feature selection algorithm since it is also tree-based and selects features that can be interpreted as good by other tree-based models. All results for the decision tree experiments can be found in Figure 3 and Table 3.

4.0.3 Logistic Regression

Looking into the overall performance of the Logistic Regressor (figures 4, 4a, 4b, and table 4), we can clearly see the impact of using feature selection. A majority of datasets performed worse after feature selection, but a similar ratio can be seen in the total time performance. Overall, using



(a) F1 performance

(b) Training & Evaluation time

Figure 3: Whether Decision Tree performs better on various datasets with or without feature selection or whether there is no statistical difference found.

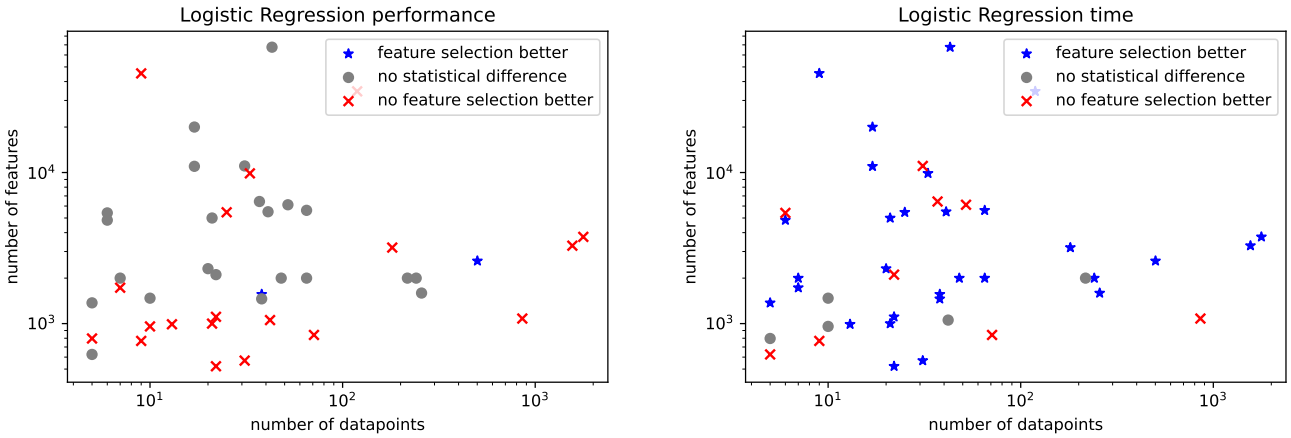
Table 3: Decision tree performance on all datasets. The cells in the first two rows indicate how many datasets performed better or worse with feature selection, and in how many datasets a statistical difference was found. The last columns indicate the average improvement by feature selection in both performance and time.

	Absolute		Statistical difference	
	Better	Worse	Yes	No
Performance	30	15	16	29
Time	18	27	36	9
Avg f1 improvement	1.425%			
Avg Time improvement	-12.934%			

feature selection along with Logistic Regression lead to an F1 performance 2.946% poorer than **without** feature selection, but it leads to an improvement of 56.772% in total time.

Table 4: Logistic regression performance on all datasets. The cells in the first two rows indicate how many datasets performed better or worse with feature selection, and in how many datasets a statistical difference was found. The last columns indicate the average improvement by feature selection in both performance and time.

	Absolute		Statistical difference	
	Better	Worse	Yes	No
Performance	8	36	25	19
Time	34	10	35	9
Avg f1 improvement	-2.946%			
Avg Time improvement	56.772%			

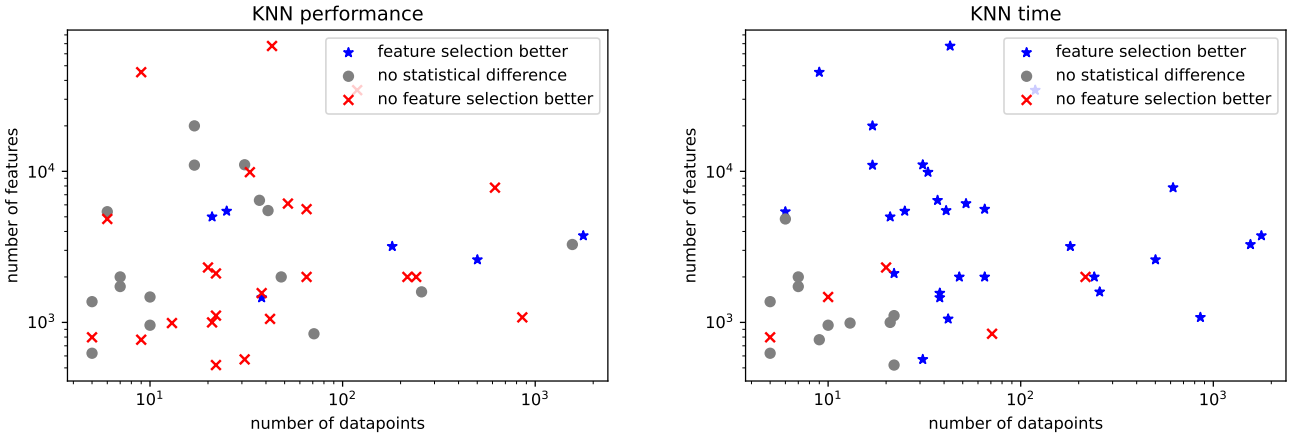


(a) F1 performance (b) Training & Evaluation time

Figure 4: Whether Logistic regression performs better on various datasets with or without feature selection or whether there is no statistical difference found.

4.0.4 K-Nearest Neighbour

Results for the KNN experiments are found in Figure 5 and Table 5. kNN algorithm also benefits a lot from feature selection since reducing the amount of features helps to fight with "curse of dimensionality" which in theory should have improved the performance, but in fact it only improved the performance by 63% because fewer calculation are required to calculate metrics value in less dimensional space.



(a) F1 performance (b) Training & Evaluation time

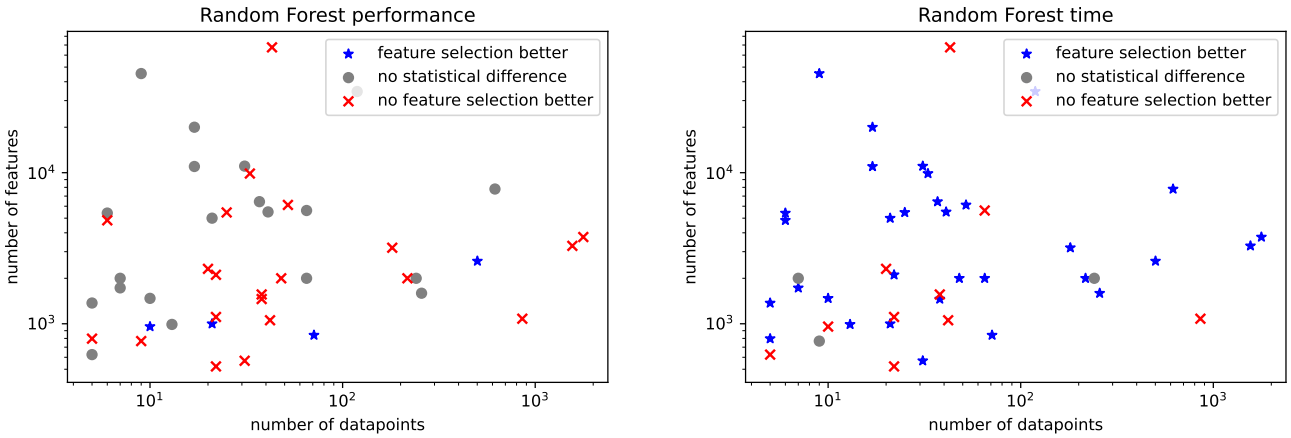
Figure 5: Whether k-nearest neighbour performs better on various datasets with or without feature selection or whether there is no statistical difference found.

Table 5: K-Nearest Neighbour performance on all datasets. The cells in the first two rows indicate how many datasets performed better or worse with feature selection, and in how many datasets a statistical difference was found. The last columns indicate the average improvement by feature selection in both performance and time.

	Absolute		Statistical difference	
	Better	Worse	Yes	No
Performance	13	32	23	22
Time	32	13	40	5
Avg f1 improvement	-1.363%			
Avg Time improvement	63.094%			

4.0.5 Random Forest

The results for the Random Forest experiments can be found in Figure 6 and Table 6. Feature selection will more often than not hurt performance for Random Forest in terms of predictions. While the average f1 difference is rather small, it is about half of the time statistically significant. Feature selection will however, often benefit a Random Forest in terms of time to train and predict. In 40 out of 45 cases, there was an improvement in time and often there was found to be a statistically significant increase in training time.



(a) F1 performance

(b) Training & Evaluation time

Figure 6: Whether Random Forest performs better on various datasets with or without feature selection or whether there is no statistical difference found.

4.1 Execution time & Performance

Execution time for most algorithms went down, except for the decision tree. This is somewhat expected as the data is simplified and algorithms have to spend less time on useless features. What is surprising is the amount of time feature selection can save, as we found percentages of over 50% for most algorithms.

Table 6: Random Forest performance on all datasets. The cells in the first two rows indicate how many datasets performed better or worse with feature selection, and in how many datasets a statistical difference was found. The last columns indicate the average improvement by feature selection in both performance and time.

	Absolute		Statistical difference	
	Better	Worse	Yes	No
Performance	12	33	24	21
Time	40	5	35	10
Avg f1 improvement	-1.833%			
Avg Time improvement	60.795%			

When investigating the performance of feature selection on different models visible in figure 7, we can see that using feature selection does not greatly impact the performance of any algorithm. For most algorithms, the F1 score of the algorithm before feature selection is slightly greater than that with feature selection applied. This can be explained by models being sophisticated enough that they will properly use features themselves or less so when they are of less importance. When feature selection does increase performance, the difference is so small that there is no real benefit of using feature selection purely for performance. On the other hand, when one wants to get the most predictive performance out of their model, it is more often found that it is better not to use feature selection. However, given the sometimes huge performance increase in terms of time required, this can be a tricky trade-off.

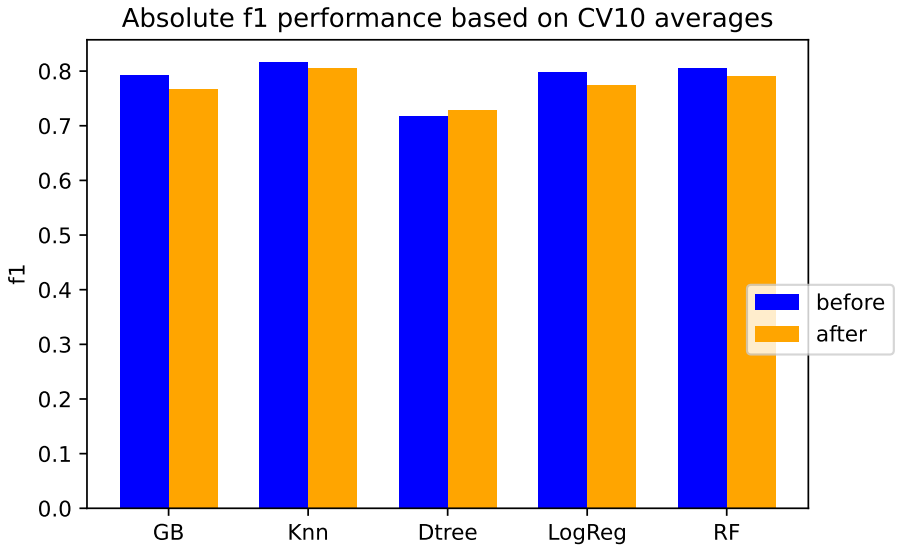


Figure 7: Absolute F1 of all models based on CV10 averages

5 Conclusions

Even though feature selection tends to give a significant boost in execution and training time it often performs worth from a predictive metrics perspective. Unfortunately, due to lack of time (some datasets took days to be processed!) we can not statistically prove it, but large datasets tend to gain more advantages from feature selection, this is somewhat expected.
Following our experimentation, we concluded that the key deciding factor into whether one should use feature selection or not comes down to the size of ones data. If the data in question holds a lot of features that do not give much value, or if resource efficiency is a crucial factor to solving ones' task, then we suggest using feature selection to improve the results, since huge benefits in training time can be achieved. However, when one wants to maximise predictive performance, it is likely that feature selection will not be of any benefit to you. Instead, it may even hurt predictive performance.

6 Discussion & future work

For completeness and reproducibility, we have added all datasets used by the various algorithms to Table 7 in the Appendix. However, we would recommend using all the OpenML-CC18 datasets in any future work.
It is worth noticing however, that OpenML-CC18 has already picked and cleaned dataset, as opposed to the real world where there is more noise and there might be more non-informative features. Investigation into more noisy datasets could be worthwhile.
Furthermore, we wanted to experiment with more algorithms, as there are more widely-known types such as Support Vector Machines and neural networks. We were unfortunately unable to do so in this study, but would recommend these additional algorithms in future work.

References

[1] V. Kumar and S. Minz, "Feature selection: a literature review," SmartCR, vol. 4, no. 3, pp. 211–229, 2014.

[2] S. Khalid, T. Khalil, and S. Nasreen, "A survey of feature selection and feature extraction techniques in machine learning," in 2014 science and information conference, pp. 372–378, IEEE, 2014.

[3] J. Cai, J. Luo, S. Wang, and S. Yang, "Feature selection in machine learning: A new perspective," Neurocomputing, vol. 300, pp. 70–79, 2018.

[4] M. J. Post, P. Van Der Putten, and J. N. Van Rijn, "Does feature selection improve classification? a large scale experiment in OpenML," in International Symposium on Intelligent Data Analysis, pp. 158–170, Springer, 2016.

[5] B. Bischl, G. Casalicchio, M. Feurer, F. Hutter, M. Lang, R. G. Mantovani, J. N. van Rijn, and J. Vanschoren, "Openml benchmarking suites," arXiv:1708.03731v2 [stat.ML], 2019.

[6] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo, "Openml: Networked science in machine learning," SIGKDD Explorations, vol. 15, no. 2, pp. 49–60, 2013.

[7] M. F. et al., "Openml-python: an extensible python api for openml," arXiv, vol. 1911.02490.

[8] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," Computers Electrical Engineering, vol. 40, no. 1, pp. 16–28, 2014. 40th-year commemorative issue.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[10] B. Strang, P. van der Putten, J. N. van Rijn, and F. Hutter, "Don't rule out simple models prematurely: a large scale benchmark comparing linear and non-linear classifiers in openml," in International Symposium on Intelligent Data Analysis, pp. 303–315, Springer, 2018.

A Appendix

Table 7: OpenML Dataset Id's used by each of the algorithms. GradientBoosting has processed more datasets as others while being slower on average because it was instantiated earlier and we did not want to exclude these results.

model	**amount**	**OpenML dataset ID's**
GradientBoosting	59	3, 6, 11, 12, 14, 16, 18, 22, 23, 28, 31, 32, 37, 44, 46, 50, 54, 151, 182, 300, 307, 458, 469, 1049, 1050, 1063, 1067, 1068, 1461, 1462, 1464, 1468, 1475, 1478, 1480, 1485, 1486, 1487, 1489, 1494, 1497, 1501, 1510, 4134, 4534, 4538, 23517, 40499, 40668, 40670, 40701, 40975, 40978, 40979, 40982, 40983, 40984, 40994, 41027
KNN, Decision Tree, Random Forest	45	6, 11, 12, 16, 18, 22, 23, 28, 31, 32, 37, 50, 151, 182, 300, 307, 458, 469, 1049, 1050, 1063, 1067, 1068, 1462, 1468, 1475, 1485, 1486, 1489, 1494, 1497, 1501, 1510, 4134, 4534, 4538, 40499, 40668, 40670, 40701, 40975, 40978, 40979, 40983, 40984
Logistic Regression	44	6, 11, 12, 16, 18, 22, 23, 28, 31, 32, 37, 50, 151, 182, 307, 458, 469, 1049, 1050, 1063, 1067, 1068, 1462, 1468, 1475, 1485, 1486, 1489, 1494, 1497, 1501, 1510, 4134, 4534, 4538, 40499, 40668, 40670, 40701, 40975, 40978, 40979, 40983, 40984