Project Report

Elise Mol, Wendy Nieuwkamer, and Isobel Smith

21 December 2016

1 Problem

After three months of learning about several machine learning techniques and classifying algorithms it was time to put our newly gained skills into practice. We decided to try our hand at one of the competitions hosted by Kaggle. The challenge we chose is Leaf Classification; the objective of the competition is to use binary leaf images and their extracted features to predict what species each leaf belongs to. The dataset which was provided includes around 1584 images of leaf specimens; these samples include 99 species. From every image three sets of features are extracted: a shape contiguous descriptor, an interior texture histogram and a fine-scale margin histogram. Every feature is represented as a 64-attribute vector.

As the competition started at 30 August 2016, there were already quite some submissions made and kernels active. One of them stood out, namely "10 Classifier Showdown in Scikit-Learn". In this kernel Jeff Delaney compared ten classifiers which are part of the python library Scikit-Learn. The results of his experiment are shown in figure 1. In his experiment Delaney used the default setup for the classifiers , combined with semi-random parameters. He notes that the performance of the classifiers could be improved by tuning the hyper-parameters. Thus, our research question is whether we can improve the accuracy found by Delaney by tuning the hyper-parameters of the classifiers.

As we did not have time to run experiments for all ten classifiers, we decided to look into five of them. We chose the K-neigbors classifier, decision tree classifier and Gaussian Naive Bayes as they had relatively decent accuracy at respectively 88%, 65%, and 57%. Also, we had some background in these algorithms as they had been taught in our Machine Learning course. Additionally, we decided to choose either Ada boost or quadratic discriminant analysis as they could improve the most with found accuracies of 4.5% and 2.5%.

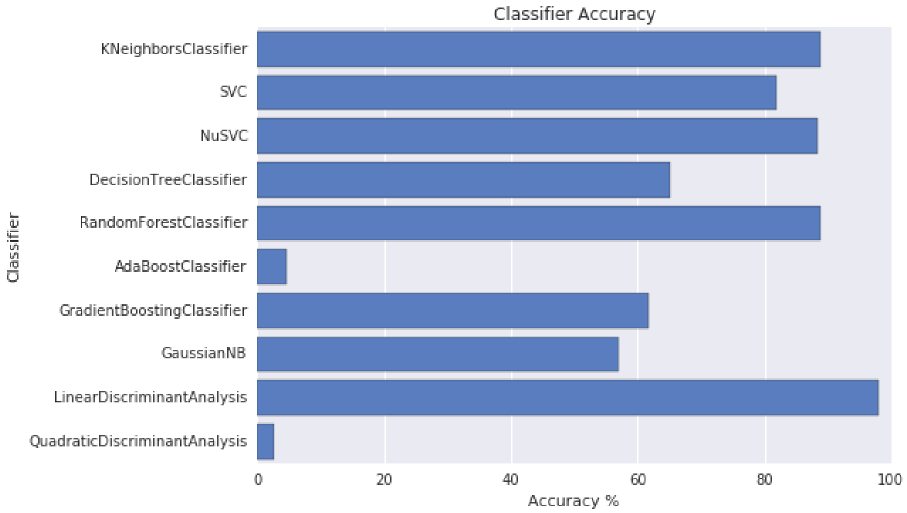


Figure 1: The 10 Classifier Showdown

2 The algorithms

Before we could start optimizing the hyper-parameters we had to research the algorithms we chose. In order to discover the most relevant information for our experiment we aimed to answer the following questions:

- How does the classifier work?
- Which hyper-parameters exist for each classifier??
- Which hyper-parameters have the most effect on the classifier?

The answers to these questions are specific to the Scikit-Learn package.

2.1 K-neighbors

The k-neighbors algorithm classifies an object according to the k nearest objects from the training set. For a higher k noise might be filtered out, but class boundaries might become more vague and less accurate. Thus, it would be interesting to find out which value of k gives the highest accuracy for our dataset. There are two ways of computing the class from the neigbors. The first one is by giving all the neighbors equal weight, no matter how far or close they are. The second is to give each neighbor a weight relative to its distance. The weighted version of the

2.2 Decision Tree Classifier

2.3 Gaussian Naive Bayes

2.4 Ada Boost Classifier

2.5 Quadratic Discriminant Analysis

3 Choices made and justifications

Which classifiers you chose ; why? Describe the relevant features of the classifiers.

4 Hyper parameter optimization

Hyperparameter optimisation is one of the most important steps in machine learning [1]. The goal is to find the best hyperparameters for the given classifier, in order to optimise the loss function and to avoid overfitting [2]. Some hyper parameter optimisation algorithms not only find the best hyperameters, but identify those that carry the most weight, [2]. There are two hyperparameter optimisation algorithims built in to scikit-learn, grid search and randomised parameter optimisation, [3], as well as alternative methods such as model specific cross validation. [4].

4.1 Grid Search

Grid Search optimises hyperparamters by iterating over all of the variables in a given range, and selecting the best combination to use in the chosen classifier. Grid search is guided by a perfomance metric, which is usually measured by cross-validation [4]. A disadvantage of grid search is that it can be time consuming and computationally expensive to run, [4]. As grid search was recommended to us, and is the most widely used way to optimize hyper parameters, we decided to use it in our project.

4.2 Grid Search in Scikit-Learn

Scikit-Learn has a built in function for Grid Search [3], which is defined below:

```
GridSearchCV(estimator, param_grid, scoring=None, fit_params=None,
    n_jobs=1, iid=True, refit=True, cv=None, verbose=0,
    pre_dispatch='2*n_jobs', error_score='raise',
    return_train_score=True)
```

The param _grid is a dictionary, with the parameters you want to optimise as the keys and the range of parameters to try as the values. The grid search algorithm will then run the given classifier over the entire parameter grid in order to find the best possible combination.

cv stands for cross validation, which, when an integer is specified, is the amount of folds in KFold cross validation. Otherwise the default is 3 folds.

Scoring allows you to choose to train for precision or recall. Precision is the fraction of retrieved instances that are relevant, whilst recall is the fraction of relevant instances that are retrieved.

$$Precision = truepositive/truepositive + falsepositive$$
$$Recall = truepositive/truepositive + falsenegative$$

We decided to focus on precision, as we wanted to tune the classifiers for accuracy, and therefore be able to compare them to the 10 classifier showdown, [5].

4.3 Our use of grid search

We used the built-in grid search function from scikit-learn, and ran it on each of our classifiers. The range of values selected for the hyperparameters were chosen through trial and error. We set $cv = 5$, as we decided to use 5 folds in the KFold cross validation scheme.

The output of GridSearch is the a dict, with the names of the hyperparameter as keys, and the optimal result as values.

```
{hyperparameter: optimal value, hyperparameter: optimal value}
```

We than ran the classifiers on the training data with the optimal parameters, and computed the new accuracy scores.

5 Evaluation of Approach and Solution

All the curves, the results

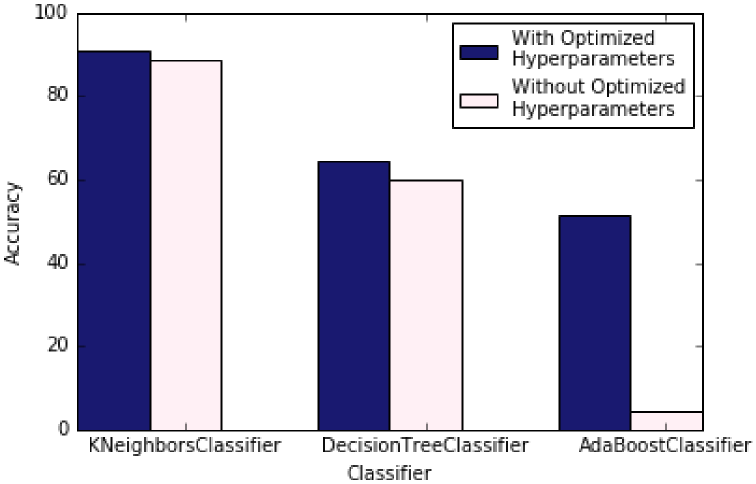
The other hyperparamter optimisation function built in to scikit-learn is randomized parameter optimisation [3]. Instead of iterating over each potential combination of parameters, like grid search, it randomly selects these combinations. The advantage if this is that it can be less computationally expensive than grid search, and that adding parameters that do not influence the performance does not decrease the algorithms efficiency. The number of random samples that are tried can be specified. As AdaBoost had such a high amount of hyperparameter combinations, using randomised parameter optimisation in this case may reduced the computation time.

An improvement would be a better way to select the range of hyperparameters to test, rather than to use trial and error.

Another improvement would be to look at choosing another value for the k-folds, instead of default 5.

6 Results

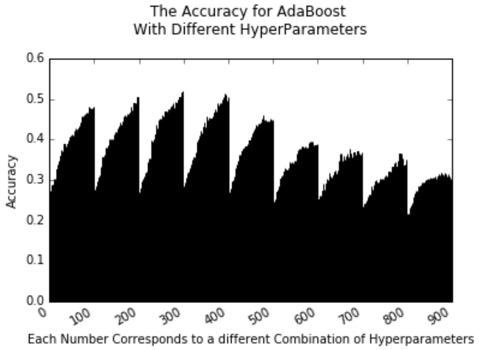
The accuracy of classifiers with optimised hyperparameters were graphed against those without optimised hyperparameters:



In each of the cases, grid search clearly improved the accuracy of the classifier, reaffirming the importance of hyperparameter optimisation in machine learning.

AdaBoost was significantly improved by hyperparameter optimisation, from 4.56% to 51.52%. This is due to the fact that the default parameters that were used in the 10 Classifier Showdown [5] were inadequate for the leaf classification problem. K-neighbors was improved from 88.89% to 90.91%, and Decision Trees were improved from 60.10% to 66.16%.

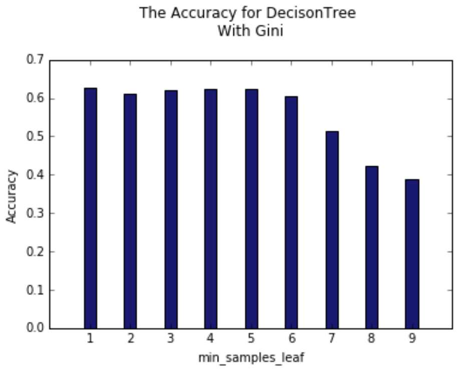
6.1 AdaBoost

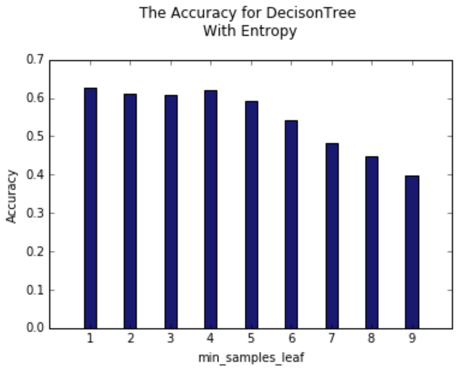


The graph above shows how grid search across each of the different hyperparameter combinations improved the accuracy of AdaBoost. The numbers on the x-axis corresponds to a different combination of hyperparameters. As can be clearly seen from the graph the amount of possible combinations was very high. This was computationally quite expensive, and running grid search on AdaBoost took a long time compared to the other classifiers.

The optimal hyperparameters for AdaBoost were found to be $n\_estimators =$ 72 with $learning\ \_rate = 0.03$

6.2 Decision Trees



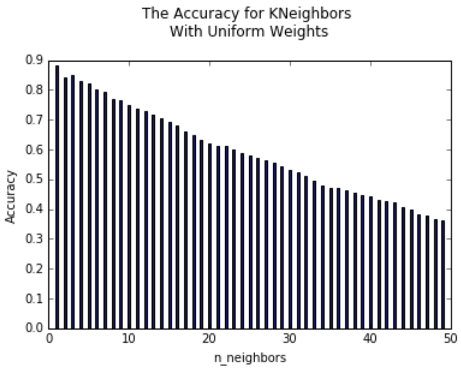


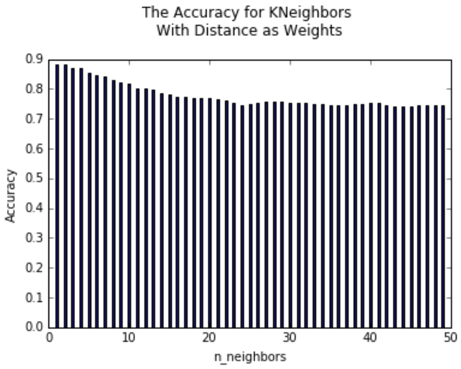
The accuracy of decision trees were comparatively not as improved as much as the accuracy of the other two were. This was because the default parameters used in the 10 classifier showdown [5] happened to be similar to the optimal parameters as found by grid search.

Interestingly, the difference between grid search using entropy, or using Gini was minimal, as can be seen from the graph. This suggests that for this problem the splitting criteria hyperparameter does not hold much weight as other hyperparameters might.

The optimal hyperparameters found for decision trees were to be $splitter =' best', criterion =' entropy'$ and $min\ \_samples\ \_leaf =' 1'$

6.3 K-Neighbors





One of the hyperparameters we needed to tune for k-neighbors were the wieights. We needed to choose between using uniform weights or distance as weights. The accuracy graphs clearly show that the accuracy rate for k-neighbors with uniform weights quickly tails of when the number of neighbors increases, whereas the accuracy when using distance as weights remains relatively high as the number of neighbours increases.

The optimal hyperparameters chosen for K-Neighbors was that $n\_neighbors = 1$, and $weights =' uniform'$

References

[1] R. Bardenet, M. Brendel, B. Kegl, and M. Sebag. Collaborative hyperparameter tuning. *30th Annual Conference on Machine Learning.*, 28:199–207, 2013.

[2] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.

[3] SciKitLearn. Gridsearchcv. [Online; accessed 15-December-2016].

[4] Wikipedia. Adaboost — Wikipedia, the free encyclopedia, 2016. [Online; accessed 15-December-2016].

[5] J. Delaney. 10 classifier showdown in scikit-learn. [Online; accessed 18-December-2016].