Statistical Machine Learning Final Project

solved with Random Forest and SVM

1st Tilman Marquart
Computer Science Bachelor
SDU Exchange Student
Nuremberg, Germany
timar20@student.sdu.dk
written text will appear in green

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
written text will appear in purple

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
written text will appear in blue

***Abstract*—final project abstract, use random forest and ... maybe write some intro sentences about the problem we try to solve**

I. Introduction

Our overall goal was to choose one fast but not so precise algorithm and one slower but more precise algorithm. We interpret fast as low training and prediction time. To choose the right algorithm we took a look at the asymptotic time complexity of the statistical machine learning algorithms (Table I).

For the fast but not so precise algorithm we choosed Random Forest. In addition to classical Decision Trees, will Random Forest generalize better on unseen data (important for the distinct people problem) by controlling the amount of various trees (n_{trees}). At the same time n_{trees} will have a direct influence on our training and prediction time as seen in Table I, so we have to find a good balance for this hyper parameter. Other critical hyper parameters for Random Forest are: The amount of input features (p); Amount of features tried (m_{try}); Nodesize ($nodesize$) controlling the tree depth; And lastly the sample size ($sampsize$). We will discuss them in detail in Section II-B.

For the slower but more precise method we decided to use a Support Vector Machine (SVM).

TABLE I: Time complexity in Big-O-Notation

Algorithm	Training	Prediction
Decision Tree	$\mathcal{O}(n^2p)$	$\mathcal{O}(p)$
Random Forest	$\mathcal{O}(n^2pn_{trees})$	$\mathcal{O}(pn_{trees})$
Support Vector Machine	$\mathcal{O}(n^2p + n^3)$	$\mathcal{O}(n_{sv}p)$
k-Nearest Neighbors	-	$\mathcal{O}(np)$

n size of training dataset, p number of features used
n_{trees} number of various trees, n_{sv} number of support vectors

II. Dataset and Preprocessing

During a visual inspection of the corner and mid dataset, the corner dataset seemed to better center the scanned digits and also cover the corners a bit better (See first two rows of Fig. 1). We therefore decided to use the corner dataset for all out further analysis and the hyperparameter tuning. We later tried again the mid dataset on our tuned and optimized models and

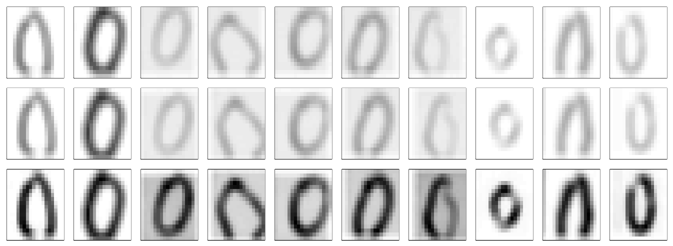
Fig. 1: Zero digits from first 10 students (left to right) in 100dpi mid dataset in first row; The same with the corner dataset in second row; Image wise min-max-normalization of corner dataset in third row

had worse results as with the corner dataset. This could proof our initial observations or more likely, the worse results are owed to the optimization of the model on the corner dataset. While visually inspecting the dataset, we also discovered some images with low saturation, most likely these digits were written with a bright or colored pencil. In order to reduce the difference between images with high and low saturation, we apply a min-max-normalization on each image. Important to say is, that we are not using a traditional min-max-normalization with the minimum and maximum taken from the whole dataset but instead take the minimum and maximum from all 324 pixels of a single image and normalize the other pixel values with these minimum and maximum values. The result of this normalization is a dataset with equally saturated images as we can see in the third row of Fig. 1. Especially for the disjunct problem this normalization improved the accuracy by 1,88% for Random Forest.

1) Random Forest Dataset and Preprocessing: As seen in Table I the amount of features will have a direct influence on the runtime. Since we want to keep the runtime low for our Random Forest we try to reduce the amount of features with PCA beforehand. In behalf to minimize the runtime we used the dataset with 100dpi, since calculating the Principal Components will take way longer as higher the resolution of the input dataset is.

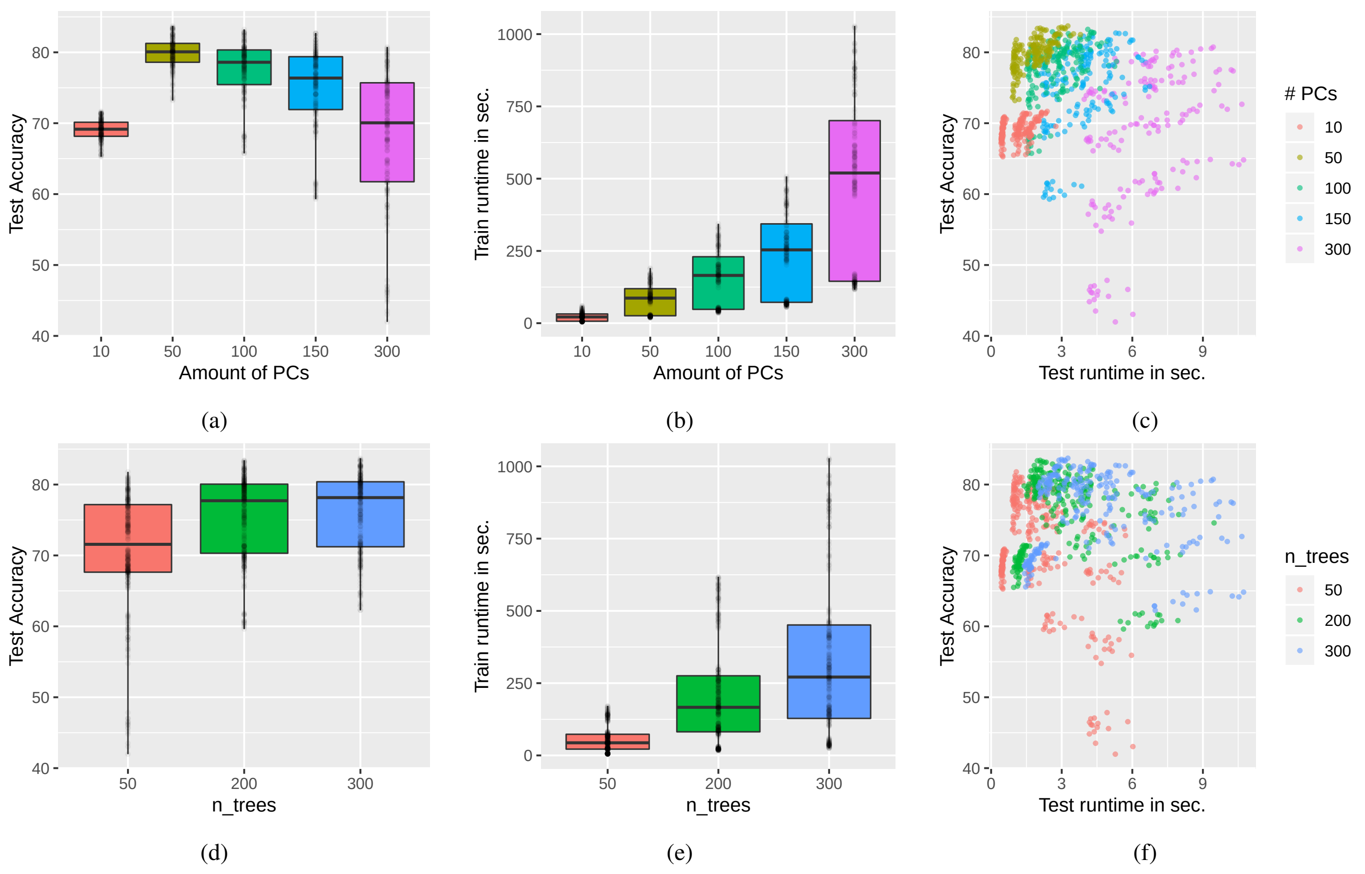

Fig. 2: Accuracy and runtime results of the 960 hyperparameter combinations; (a) Boxplot showing the accuracy for the hypergrid PCA sequence; (b) equivalent but for training runtime instead of accuracy; (c) accuracy over test runtime, ideal would be a point in the top left corner; (d,e,f) respectively to (a,b,c) but for amount of various trees n_tree instead of p.

A. *Hyper parameter Optimization*

Optimize critical parameters (discussed under 1) on a smaller set and document this optimization process.

TABLE II: Sequences for the hyperparameter grid

Hyperparameter	Sequence
Amount of Principal Components p	10, 50, 100, 150, 300
Random Forest n_{trees}	50, 200, 300
Random Forest m_{try}	1, 2, 4, 8
Random Forest $nodesize$	5, 20, 40, 60, 80
Random Forest $sampsize$	42000, 48000, 54000

B. *Hyper parameter Optimization for Random Forest*

Since optimizing random forest on the smaller dataset, was not bringing the right generalization, we decided to use a medium size dataset of 30 distinct people for training and the remaining 17 for testing and automate the finding of the right hyperparameters with a grid search. Therefore we create a hyperparameter grid with 960 combinations of the 5 hyperparameters (See Table II for details of the grid) and splitted them into 10 segments. We then used 4 Google Colab and 6 Kaggle notebooks in parallel, to let each segment of combinations run on each notebook. For each run we saved the Random Forest training and prediction runtime and the prediction accuracy on the test set. Google Colab and Kaggle are using nearly the same Intel Xeon CPU which was important to have a comparable runtime. In total the gridsearch took 47 hours and 53 minutes. The best result had a test accuracy of 83.72% (See Table III for further information). In the following sections we will discuss the results of the gridsearch and how we used the results for a further manual fine tuning of the hyper parameters.

1) Amount of Principal Components p: The amount of features p will have a direct influence on the training and testing runtime as we can see in the computational complexity $\mathcal{O}(n^2pn_{trees})$ and $\mathcal{O}(pn_{trees})$. In order to keep Random Forest fast, we use Principal Components Analysis (PCA) to reduce the amount of features and use the first p Principal Components (PCs) as our p input features for Random Forest. The runtime results of our hypergrid search also proof the theoretical computational complexity as we can see in Figure 2b). The highest test accuracy of our results (Figure 2a) could be achieved by taking the first 50 PCs as input features. Adding more components even decreased the accuracy, which indicates an overfitting. With 50 PCs as Input Features, performs

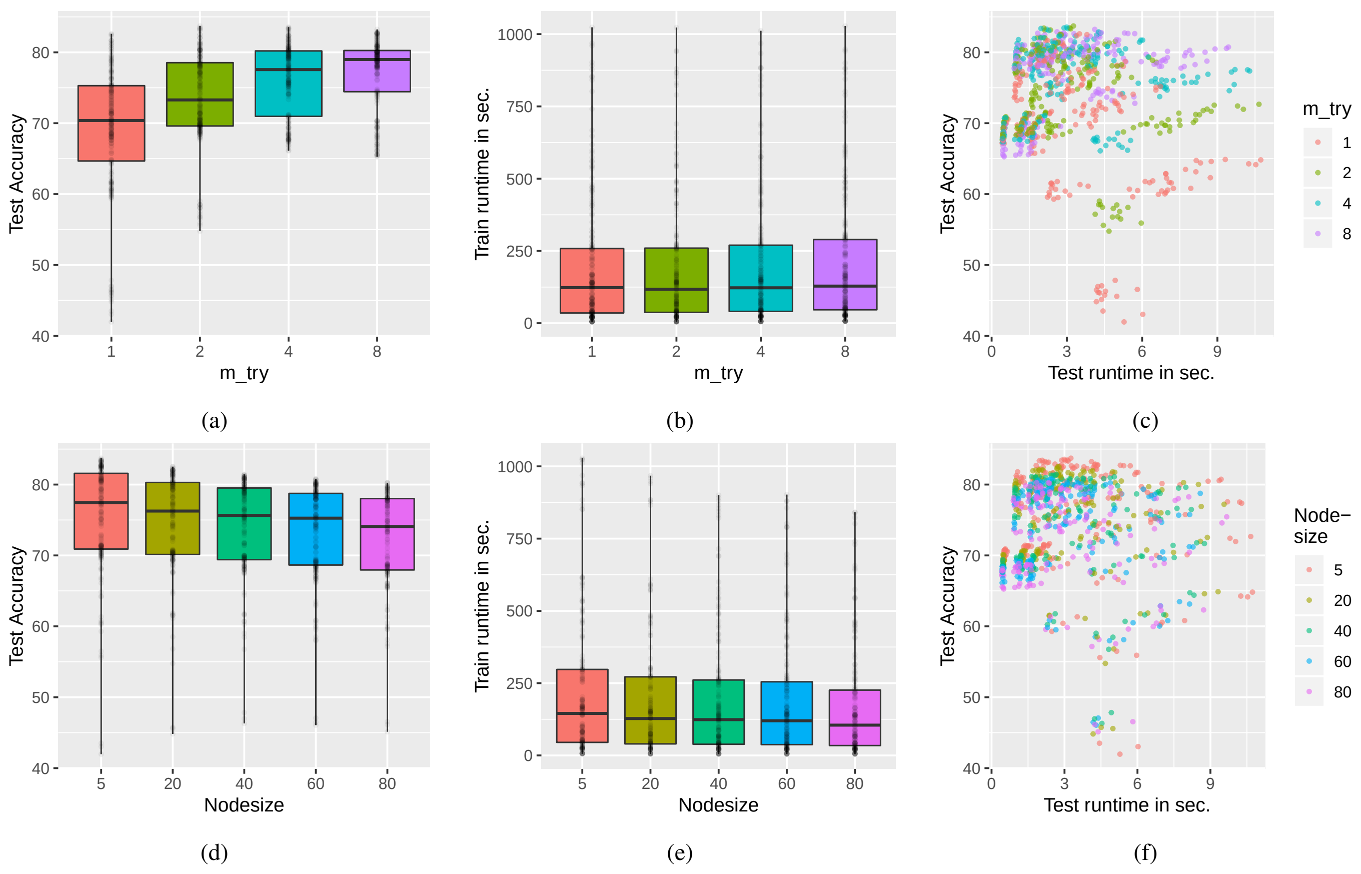

Fig. 3: (a,b,c) respectively to Fig. 2(a,b,c) but for amount of features to try m_{try} instead of p; (d,e,f) respectively to Fig. 2(a,b,c) but for $nodesize$ instead of p;

Random Forest still very fast (around 3 seconds) while also archiving it's best accuracy for our problem (See cluster of green dots in upper left corner of Figure 2c). Hence we took the 50 PCs and manually fine tuned our model further. Finally we came up with a final value of 40 PCs for our Random Forest PCA preprocessing.

2) Random Forest n_{trees}: As we would expect it from the computational complexity and as we can see it in our test results the amount of various trees n_{trees}, which the Random Forest algorithm growth, heavily effects train runtime (see Fig. 2e)) as well as test runtime (see Fig. 2f)). We therefore looking for the lowest value of n_{trees} which still achieves a good accuracy. As we can see in Figure 2d) the accuracy does not vary a lot between 200 or 300 Trees, so we decided on a final n_{trees} value of 200, because of the faster test and train runtime.

3) Random Forest m_{try}: In Random Forest training, the amount of features tried (m_{try}), controls how many features are checked for finding the best decision point for a new tree branch. Therefore it has no influence on the testing time. As we can see in Figure 3, all values of m_{try} (different colors) are evenly spread along the test runtime axis which confirms this assumption. Since we reached the highest mean test accuracy at the border of our m_{try} grid domain, we manually tested even higher values for m_{try} for our final model, but all of the higher values had even worse results, so we finally choose a m_{try} value of 4 which yielded the best test accuracy.

4) Random Forest $nodesize$: The hyper parameter $nodesize$ controls the minimum amount of samples in the leaf nodes and therefore also controls the depth of the tree. By decreasing this value our tree will grow deeper and therefore the test and train time will increase. We can also see the increasing train (see Fig. 3e) and test time (slight shift of red points to the right in Fig. 3f) in our hyper grid search results. Even if it increases test and train time we choose 5 as our final $nodesize$, since it brought the best accuracy (see Fig. 3d). During manual fine tuning we tried even smaller values but had worse results, probably because of overfitting.

5) Random Forest $sampsize$: During the grid search we tried out 3 different values for $sampsize$. This value will define how many samples from the input data are taken into consideration to generate the decission trees. Our grid search results did not showed a significant difference in accuracy between all of the 3 tested values. We therefore decided to use the default value of this hyperparameter.

III. Evaluation

Do a proper cross validation and indicate also mean and variances for all problems. Describe results on test and train-

TABLE III: Intermediate Results after hyperparameter tuning

Result name	Dataset split in students	Train time	Test time	Test accuracy
Random Forest best gridsearch	disjunct 30 / 17	171 sec.	3.26 sec.	83.72%
Random Forest + manual finetuned	disjunct 30 / 17	127 sec.	7.27 sec.	84.15%
Random Forest + manual finetuned	all-in 30/17	129 sec.	7.18 sec.	93.97%
Random Forest + manual finetuned + image wise min-max-norm	disjunct 30/17	138 sec.	6.37 sec.	85.68%
Random Forest + manual finetuned + image wise min-max-norm	all-in 30/17	139 sec.	7.46 sec.	93.75%

ings set and reflect on overfitting. (remeber: Give information about the computational time required.) (remeber: Analyze the results and give proper explanations)

A. *Random Forest*

=¿ use 10 fold crossvalidation with 43 to 4 with bigger set (remeber: Give information about the computational time required.) (remeber: Analyze the results and give proper explanations)

IV. Future Work

Give an indication what could be further improved.

optimize Random forst model also on corner dataset and see how it works