Robin Erd
Project II - Autumn 2021
University of Bergen – INF264

Report – Project 2: Digit recognizer

Summary

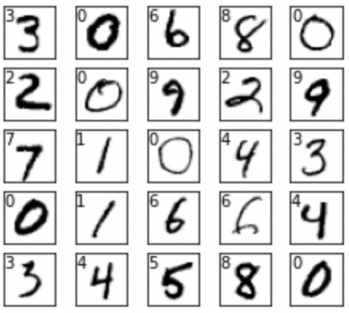
The goal of this project was to develop a computer program that recognizes handwritten digits by using machine learning techniques. The task of recognizing handwritten digits can be understood as a classification task. Every image of a handwritten digit is fully described by 27 times 27 integer values between 0 and 255, each of them corresponding to the brightness of one pixel. The objective is to assign each of these matrices a class, where each class corresponds to one of the numbers between 0 and 9.

There are several different classifiers (classification algorithms) that can be used for classification, each with its advantages and disadvantages. Most classifiers try to find a pattern in the matrices which distinguishes the classes from each other. The performance of the classifiers then depends on how well the patterns found by the classifiers can be generalized. During this project, Neural Networks, Decision Tree Classifiers, k-Nearest-Neighbour Classifiers, and Support Vector Machine Classifiers with different configurations were evaluated, and the Support Vector Machine with the configuration: *kernel coefficient (gamma) 'scale', regularization parameter (C) '3', rbf kernel* turned out to be the classifier with the best expected performance on unseen data, that is, in real-life application, among them. During final testing, it achieved an accuracy of 98.16%.

The results show that many configurations of the classifiers achieved high accuracy on the test data - almost all of them achieved an accuracy of over 85% and most achieved an accuracy of over 90%. This allows the conclusion that the problem is well suited for being approached with machine learning techniques and that handwritten numbers consist of patterns that are sufficiently easy to distinguish from each other for these models. Although the real-world performance can, due to the validation bias, not be expected to be as good as the performance on the test and validation data, it can, with reasonable confidence, be assumed to be no worse than 95%.

Still, it is to be noted that the performance is highly dependent on the quality of the input data. In some cases it is hard to assure the quality of data in real-world applications of machine learning, posing a challenge for any machine-learning model.

Figure 1: A sample of the images to classify with their correct labels



Technical Report

Preprocessing steps

The data is provided in two `*.csv` files, one containing the labels and one containing the images. Each row corresponds to one image (70000,784). The files can be read using `pandas` and then converted into NumPy arrays. The result is a 69999x784 size array with the input data which can be reshaped into a 69999x28x28 array and a 69999x1 size array containing the labels (0-9).

To optimize the performance of the classifiers the data should be normalized. This can be done by dividing the input array (containing the images) by 255. Furthermore, the array containing the labels has to be converted to a one-hot-encoded version for some classifiers, specifically the Decision Tree Classifier and the Neural Network. In addition to that, the Decision Tree Classifier and the Support Vector Machine also require the input data to be flat. Consequently, for these classifiers, the input data is flattened to an array of shape 69999x784.

Candidate algorithms and hyperparameters

Every machine learning algorithm used for classification tasks presented during the lecture is a possible candidate for this task. I chose to exclude ensemble methods from the candidate algorithms and instead included most other classifiers. Logistic Regression was excluded because it generally often delivers very similar performance to Support Vector Machines and I ran into issues regarding RAM usage during initial testing of Logistic Regression models. Clustering algorithms were excluded because they do not make use of the labeled data (unsupervised) and it makes sense to make use of any information available to get the best results. In conclusion, model selection will be performed on all classifiers covered in the lectures except for the ones excluded, specifically: Neural Networks, Decision Tree Classifiers, k-Nearest-Neighbour Classifiers, and Support Vector Machines.

For every candidate algorithm, there are multiple hyperparameters to choose and test. Due to limited resources (time and computing power), it is impossible to test every possible combination of all hyperparameters. Therefore the initial hyperparameters from which outward different variants were tested are close to the configurations which were recommended or preset during the exercises and homework.

The model selection on the Neural Network covers the train batch size as well as models with 1, 2, or 3 layers and 392 or 784 nodes on each layer (excluding the output layer). Apart from that, every model uses the ‘relu’ function as activation function (recommended during the lecture) is trained for 50 epochs (increased value from the homework exercise due to the problems' higher complexity). Only simple layers were used, where simple refers to the layers not being of more advanced types such as e.g. convolutional. Those are excluded because they were not covered by the lectures, exercises, or homework in detail, although they would probably lead to a significant increase in accuracy.

Model selection on the Decision Tree Classifier covers two different splitting criteria; entropy and Gini as well as two different settings for splitting; best and random. Pruning is not performed because of the way it is implemented in `sklearn`. Performing model selection on pruning would require manual extra steps (for determining the right alpha) which could not be executed automatically for every configuration. This is further discussed in the *Results* section.

The k-Nearest-Neighbour classifier's most important hyperparameter is the number of neighbors to take into account during classification. For that reason, this parameter is varied the most during model selection (32, 64, 512, 1024). For this sequence, I chose a large starting value (32) because of the size of the dataset and rapidly increased it (due to the size of the dataset). In addition to that (because of time constraints) the model selection covers only the weight parameter, which determines whether all neighbors are weighted uniformly or according to their distance. For the algorithm parameter, the 'auto' configuration is used and the p-value parameter also remains at the default value, that is 2.

The Support Vector Machine model selection covers different kernels (kernel, 'linear', 'poly', 'rbf'), different regularization parameters (C, 1,2,3), different gammas (gamma, 'auto', 'scale'), different degrees for the polynomial kernel (degree, 2,3) and different constant terms for the polynomial and sigmoid kernel (coef0, 0,1,2). These hyperparameters cover a wide range of possible configurations, allowing for an initial orientation and providing a direction in which the results could be further improved. The sigmoid kernel is excluded because a test run on the validation data revealed very bad performance (70% accuracy) and the value 1 for the degree parameter is excluded. After all, this results in the same computations as the linear kernel. In general, the degree and coef0 parameters are only taken into account for the poly kernel. Furthermore, the gamma-parameter is not taken into account by the linear kernel. For the parameters degree, coef0, and C the chosen range of values to test is based on a stepwise increase from 0. The model selection covers all possible values of gamma and kernel, except for the excluded sigmoid kernel.

Performance measure

For this task which is a multi-class classification, different metrics can be used to evaluate the performance of a model, such as recall, precision, F1 score, and accuracy. The F1 score is best chosen in cases where high recall and high precision are equally important. One could argue that this is the case for the handwritten digit classification task, but there is a simpler solution; the accuracy metric. The accuracy metric is perfectly suited for multi-classification tasks in which all classes are equally important and the distinction between true positives and true negatives or false positives and false negatives is not necessary and arbitrary. Still, the F1 score can prove useful if the training data is imbalanced as this impacts the expressiveness of the accuracy measure, but this is not the case. The distribution of our training data is 6902, 7877, 6990, 7141, 6824, 6313, 6876, 7293, 6825, 6958 which is close (enough) to a uniform distribution to justify using accuracy over the F1 score.

Model selection

For model selection, I chose the Cross-Validation model selection scheme. This allows for a better estimate of how well the model is expected to perform in real-world applications and on unseen data. Cross-Validation provides the opportunity to be trained and tested on multiple-train-test splits and therefore makes better use of the data by delivering more data to evaluate the model configuration. In this project, I will take the average over the accuracies the model achieved on the different splits. Regular validation only depends on one train-test split. The main disadvantage of using Cross-Validation is that it takes significantly more time to perform the model selection as every model has to be trained and tested multiple times instead of just once for every configuration. For this reason, I chose three-fold splits for Cross-Validation.

Result

The tables and figures below contain and illustrate the average accuracy that was achieved with the given configuration of the given classifier when trained on the training data and evaluated on the validation data (3-Fold-Cross-Validation). For each classifier, the best-performing configuration is highlighted.

Table 1: NN performance during three-fold Cross-Validation (averaged)

Configuration (batch size, nodes, layers)	average accuracy (on the validation folds)
500, 392, 1	0.90119
500, 392, 2	0.91121
500, 392, 3	0.91904
500, 784, 1	0.90404
500, 784, 2	0.91614
500, 784, 3	0.92299
1000, 392, 1	0.88337
1000, 392, 2	0.89354
1000, 392, 3	0.90023
1000, 784, 1	0.88709
1000, 784, 2	0.89806
1000, 784, 3	0.90435

Figure 2: NN performance during three-fold Cross-Validation (averaged)

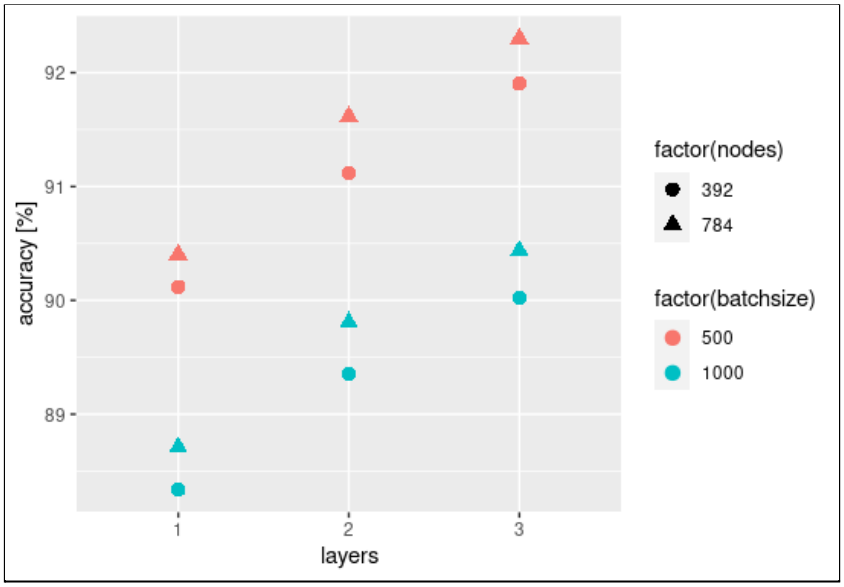


Table 2: DecisionTree performance during three-fold Cross-Validation (averaged)

Configuration (measure, split)	average accuracy (on the validation folds)
gini, best	0.84816
gini, random	0.84449
entropy, best	0.85371
entropy, random	0.85190

Figure 3: DecisionTree performance during three-fold Cross-Validation (averaged)

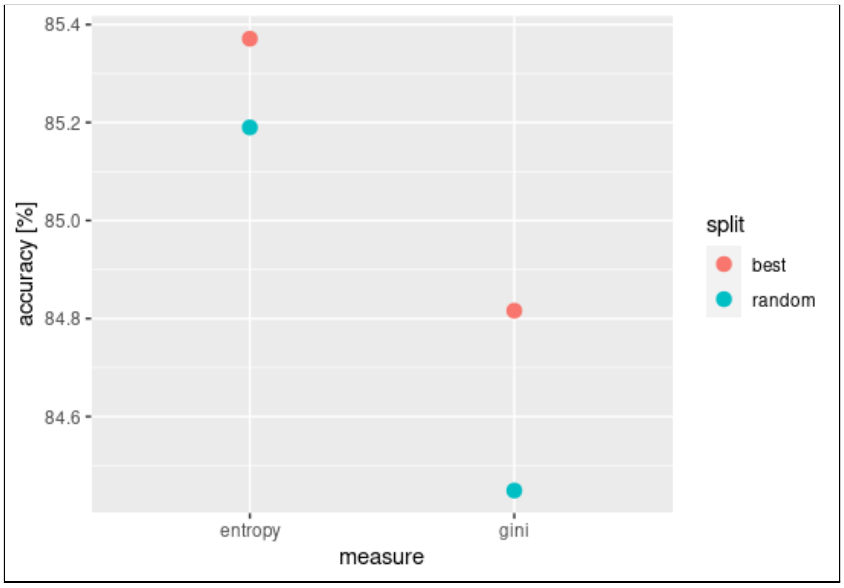


Table 3: kNN performance during three-fold Cross-Validation (averaged)

Configuration (k, weight)	average accuracy (on the validation folds)
32, uniform	0.94617
32, distance	0.94776
64, uniform	0.93376
64, distance	0.93569
512, uniform	0.86038
512, distance	0.86564
1024, uniform	0.80907
1024, distance	0.81816

Figure 4: kNN performance during three-fold Cross-Validation (averaged)

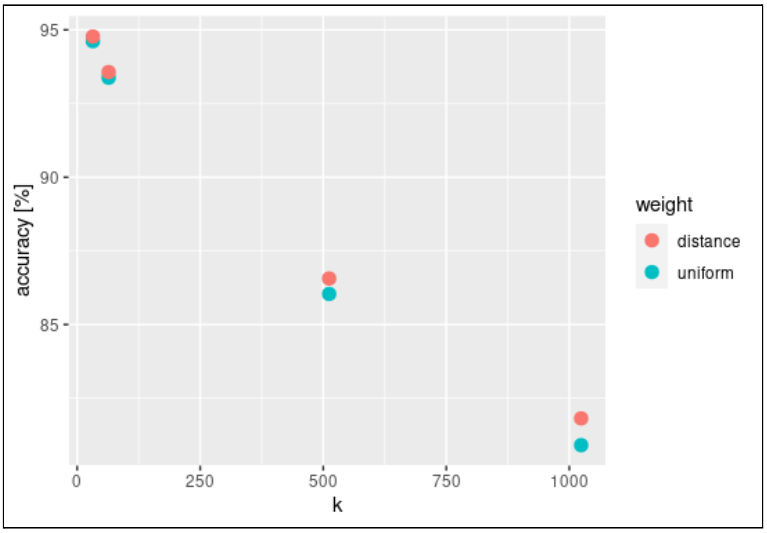
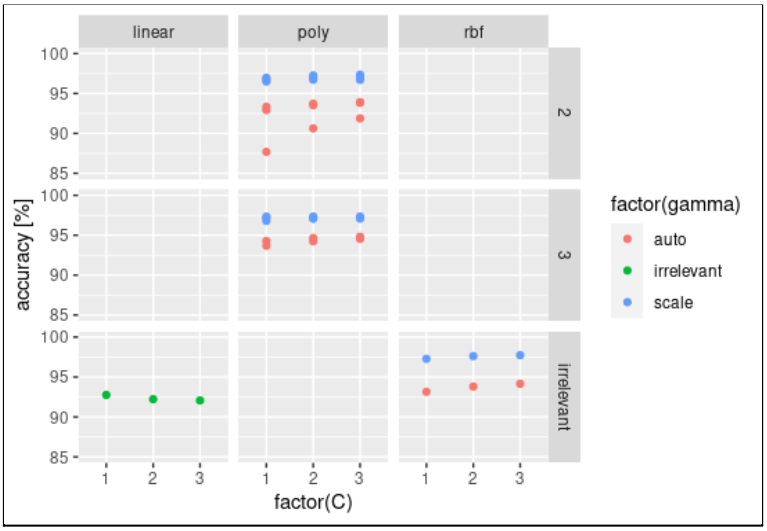


Figure 5: SVM performance during three-fold Cross-Validation (averaged)[1]



[1] Note: coef0 is not part of the figure as it would have decreased readability even further and the y-axis facet corresponds to the degree parameter while the x-axis facet corresponds to the kernel parameter, outliers were excluded for readability (axis scaling)

Table 4: SVM performance during three-fold Cross-Validation (averaged)

Configuration (gamma, coef0, degree, kernel, C)	average accuracy (on the validation folds)
irrelevant, irrelevant, irrelevant, linear, 1	0.92752
irrelevant, irrelevant, irrelevant, linear, 2	0.92221
irrelevant, irrelevant, irrelevant, linear, 3	0.92071
auto, 0, 2, poly, 1	0.87688
auto, 0, 2, poly, 2	0.90626
auto, 0, 2, poly, 3	0.91876
auto, 1, 2, poly, 1	0.92921
auto, 1, 2, poly, 2	0.93531
auto, 1, 2, poly, 3	0.93826
auto, 2, 2, poly, 1	0.93336
auto, 2, 2, poly, 2	0.93717
auto, 2, 2, poly, 3	0.93919
auto, irrelevant, irrelevant, rbf, 1	0.93150
auto, irrelevant, irrelevant, rbf, 2	0.93802
auto, irrelevant, irrelevant, rbf, 3	0.94149
auto, 0, 3, poly, 1	0.33584
auto, 0, 3, poly, 2	0.54025
auto, 0, 3, poly, 3	0.64628
auto, 1, 3, poly, 1	0.93681
auto, 1, 3, poly, 2	0.94240
auto, 1, 3, poly, 3	0.94545
auto, 2, 3, poly, 1	0.94276
auto, 2, 3, poly, 2	0.94654
auto, 2, 3, poly, 3	0.94817
scale, 0, 2, poly, 1	0.96969
scale, 0, 2, poly, 2	0.97267
scale, 0, 2, poly, 3	0.97348
scale, 1, 2, poly, 1	0.96752
scale, 1, 2, poly, 2	0.96978
scale, 1, 2, poly, 3	0.97043
scale, 2, 2, poly, 1	0.96493
scale, 2, 2, poly, 2	0.96724
scale, 2, 2, poly, 3	0.96705
scale, irrelevant, irrelevant, rbf, 1	0.97280
scale, irrelevant, irrelevant, rbf, 2	0.97617
scale, irrelevant, irrelevant, rbf, 3	0.97731
scale, 0, 3, poly, 1	0.96793
scale, 0, 3, poly, 2	0.97086
scale, 0, 3, poly, 3	0.97150
scale, 1, 3, poly, 1	0.97336
scale, 1, 3, poly, 2	0.97345
scale, 1, 3, poly, 3	0.97343
scale, 2, 3, poly, 1	0.97183
scale, 2, 3, poly, 2	0.97121
scale, 2, 3, poly, 3	0.97102

From these tables and figures, we can see that the Support Vector Machine with the configuration: *kernel coefficient (gamma) 'scale', regularization parameter (C) '3', rbf kernel* performed best among all tested configurations and models. This classifier with the corresponding configuration consequently is chosen as the final classifier because it will, with high probability, also perform best on the test data and in a real-world application.

Support Vector Machines work by trying to find a hyperplane that best separates the different labels. For multi-class labeling, this at first sounds unintuitive but works the same way, except that some additional steps are necessary - the given multi-class classification problem dataset is split into multiple binary classification problems which are solved, and in the end, the label with the most votes is assigned to the data point. Fortunately, this already is implemented in the Support Vector Machine part of `sklearn` and can be used by passing 'ovo' (one for one) as the decision_function_shape parameter.

To avoid overfitting of the Support Vector Machine I made use of the regularization parameter 'C', which defines the strength of the regularization proportional to C and uses a squared l2 penalty. During model selection, the values 0, 1, and 2 were tested for C.

For the Neural Networks, the measures taken to avoid overfitting are: using simple models with only one layer and 392 nodes in the model selection process, and: training for a low number of epochs. Weight regularization was not used due to time constraints as every additional parameter doubles the time required for model selection. While the Neural Networks performed reasonably well with an accuracy of about 90% they only made the second last place. The results do not show a significant variation in accuracy with respect to the used configuration which hints at a possible general upper bound for this classifier with basic layers and the given data.

The Decision Tree Classifier achieved an accuracy of about 85% which is the worst result. This can probably be traced back to the lack of pruning as the Decision Tree Classifier is naturally prone to overfitting. Pruning was not used during this project because numerous different methods can be used for pruning and most of them require manual testing and additional research. I tried to keep the effort put into the different models balanced so this was not an option and unfortunately, it seems like sklearn does not provide a function to apply reduced-error-pruning.

The k-Nearest-Neighbour Classifier performed surprisingly well if one considers the number of dimensions the data has (784 pixels = dimensions). The performance can most probably even be improved by adding additional investigation during the model selection phase about the parameter k. The k-Nearest-Neighbour Classifier can be prevented from overfitting by choosing a large enough k. The model selection involved as many different values for k as possible given the time and computing power limitations.

Outlook

As the final model achieved an accuracy of 98.168% on the test data I think it can be reasonably assumed that the performance on new, similar, unseen data will be slightly worse (due to the validation bias) but most probably no worse than 95% accuracy. However, the performance in a real-world application may vary strongly because of very inconsistent quality in the input data.

Given more resources I would try to improve the solution by delving deeper into more advanced configurations of neural networks such as convolutional neural networks which are designed specifically to work with images and I would perform further testing concerning the k-Nearest-Neighbour Classifier and its parameter k. Besides that, I would collect information concerning the different methods that can be used to avoid overfitting the Decision Tree Classifier and test them because I believe that it could perform much better (it has the most potential for improvement).

I would not try to improve the performance of the Support Vector Machine because I think that there is not enough room for improvement. Most of the configurations were varied extensively during the model selection in this project already, only the parameter C can be tweaked and one must also consider the outliers which quite probably are part of the dataset used for training and testing which prevent any classifier from achieving an accuracy of 100%.

If my objective continued to be to find the best possible classifier, I would most certainly focus on the convolutional neural networks which are used in most image-classification tasks in real-world applications that I know of. Trying to improve the performance of the other classifiers would be of interest just to know what their best performance on (this) image classification task is.