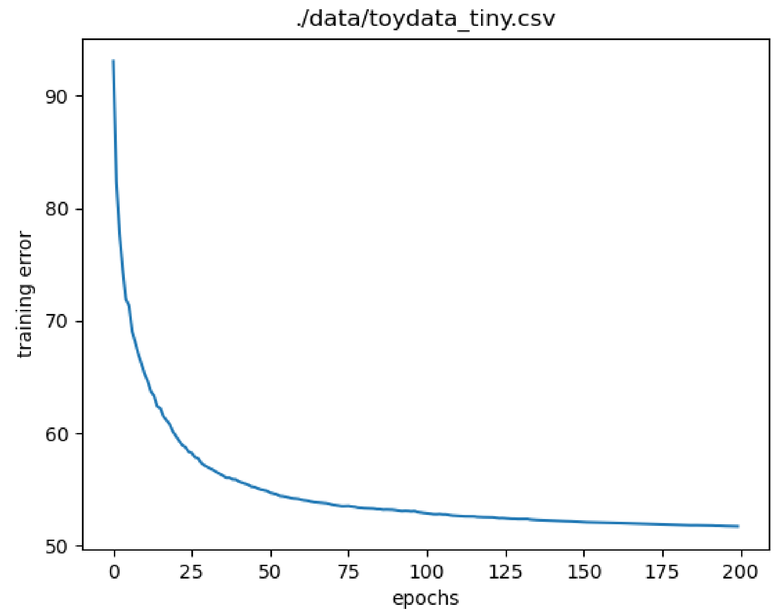
Mining Massive Data
Programming Assignment 2

Lukas Hinterleitner, Lukas Till Schawerda, Kilian Wohlleben

Linear SVM Model

We implemented a python class *CustomSVC* to take care of the linear support vector machine (SVM) model. Stochastic gradient descent applies the steps according to the gradient of the hinge loss and multi-class hinge loss respectively. Parameter tuning was done heuristically by testing various values in a way that maximizes the model performance. We use the same parameters for all data sets (learning rate = 0.1, regularization parameter C = 1.5). The model performance on the different data sets is pictured in the table below. The plot on the right shows the training process on the *toydata_tiny* data set.

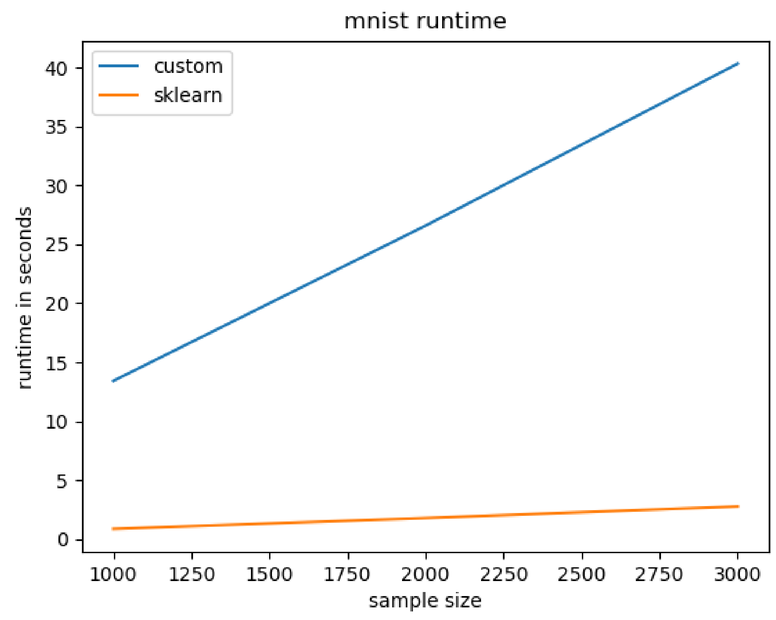
Data set	Accuracy	Runtime (seconds)
Toydata tiny	0.895	3.6
Toydata large	0.999	3440.3
MNIST	0.789	1268.3



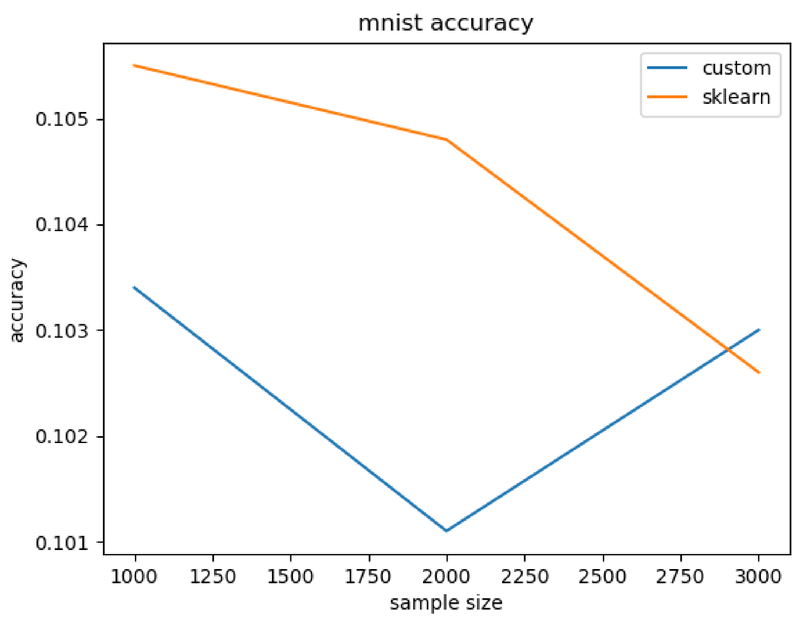
Random Fourier Features

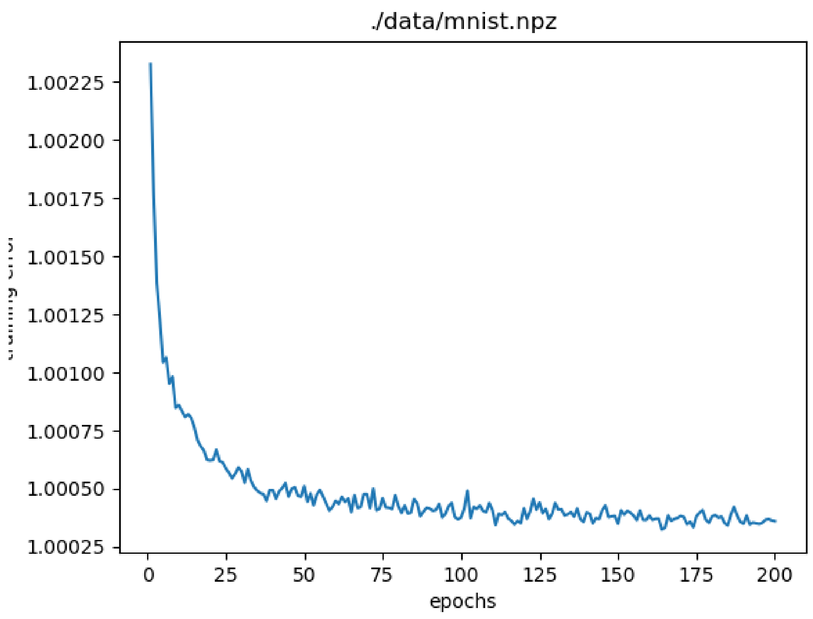
Random Fourier features (RFFs) were implemented following the paper by Rahimi & Recht 2007 (see https://proceedings.neurips.cc/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf). This is the method we discussed in class. Learning rate and regularization parameter were selected through experimenting with different values and finding the ones that yielded the best results. We found that using the same parameters for the different data sets worked well. The runtime of the computation is short. It increases slightly with more RFFs, with the most computationally intensive models taking at most 1 minute in our tests. The accuracy of the various models can be seen in the table.

Number	Toydata tiny	Toydata large	MNIST
100 RFFs	0.965	0.535	0.099
200 RFFs	0.980	0.508	0.098
500 RFFs	0.990	0.534	0.098
1000 RFFs	0.995	0.553	0.104



The plot below shows the learning process of the model and the convergence of SGD with 1000 RFFs on the mnist data set. The plots on the right display runtime and performance when training on 1000, 2000 and 3000 traning samples.





Parallel Implementation

We implemented parallelized SGD based on the procedure desribed in the paper by Zinkevich et al. 2010 (see https://proceedings.neurips.cc/paper/2010/file/abea47ba24142ed16b7d8fbf2c740e0d-Paper.pdf) The python *threading* library was used to apply actual parallel computation as requested for the bonus point exercise. We also used the *sklearnex* library, which improves performance when using Intel processors. It can be installed as described here: https://intel.github.io/scikit-learn-intelex/installation.html. We also implemented threading using the library proposed in the task description, so that each machine represents a single thread. A thread lock is used when appending the calculated weight vector to an array to guarantee thread-safety.
The parameters were again selected manually by testing various candidates to optimize model performance. The results can be seen in the table below.
The runtime becomes less the more machines are used. However, when the number of machines increases drastically, the accuracy of the model will be worse. In our case, the accuracy was just a little bit less then the standard SGD implementation. Additionally, the data has to be sufficiently large, otherwise the parallelization algorithm would lead to a bad accuracy.

Data set	Accuracy	Runtime (seconds)
Toydata tiny	0.790	3.1
Toydata large	0.999	15.0
MNIST	0.817	13.9