

Executive summary

Machine learning is a field in programming where you teach a computer how to do decisions based on training. In this postal office digit recognizer our goal is to find the best machine learning tool to teach the computer to read input and give a prediction to what digit it thinks it sees. Machine learning is a very relevant approach to solve this task. As long as we have access to enough training data then we will be able to create such prediction models. We are lucky enough being able to use the MINIST data set. My mission as a programmer in this project was to find an appropriate model to train the computer, then enabling it to predict new data. My main focus was finding a model type with tweaked parameters such that the model would perform with great accuracy, where we could train the model in reasonable time, and make predictions fast. These priorities was made to tailor the model for use in a postal system. Accuracy was of course essential so that letters are not sent to the wrong recipient.

The model that was chosen as the one for this task was a custom neural network. A neural network is computational structure/model highly inspired by the human brain. This model can predict images of digits with an accuracy of 99,6 % given thats the images of the digits is centered and rotated correctly, ie. uniform.

Technical report

I chose to experiment with the following classifiers. Bayesian, Decision tree, random forest (ensemble) and neural network.

For my testing i divided my dataset in to a test set and a train set. I used the split ratio of 70/30, with a random state seed of 42. For my models I stuck with the default random seed, which should use numpys standard seed. My code and results should be reproducible using the same split and random parametres. The full logs from my testing is located in the end of this document.

Reading input

The input was given as two csv files. To import this I used the built in functions in Python. First I opened the files, then I used the csv reader. Every line was put in a list. Every element in the lists was as type string, so it had to be parsed to ints. Also the lists was made in to Numpy arrays to make working with them more easy. These simple modifications was enough to use them with the models from Sklearn. For using Keras some additional changes was needed. The y data had to be made

categorical, and the x data had to be reshaped to arrays of array (from array of arrays of arrays).

Model selection

I decided to try out a few different approaches to our image recognition task. The first classifier I started to play with was KNN. It's the model that is easiest to understand in my opinion, it can also give fairly accurate results. It became clear very fast that the prediction time was way too long. This is because it has to calculate the distance to every other point when predicting. I abandoned KNN for that reason. I also did some testing with adaboost, but that was also abandoned because of accuracy problems. The next classifier I started looking into was the bayes classifier because it's very easy to implement, and is said to be fairly accurate. The next classifier I wanted to use was decision tree. The training here is also fairly fast, and predictions is very fast. Also it's quite easy to understand the logic behind it's workings. The next classifier I chose was random forest. I chose this to see how it could increase the accuracy compared to decision tree. The final classifier I implemented was a custom neural network. My hypothesis was that this would work very well for this task, both in terms of speed and especially accuracy. The following table shows a brief overview of key pros and cons of the main classifiers we have discussed in class that was considered by me for this project. This is what I used when deciding on classifiers.

Model	Cons	Pros
KNN	Slow prediction Slower the more dimensions Might perform bad with skewed data	Easy implementation Can give high accuracy Good for categorical data
Lin. reg.	Not well suited for categorical data Expensive training	Good for approximations Fast prediction
Log. reg.	Not designed for classification of more than two targets (but possible with sparisy) Expensive training Can have high bias	Fast prediction Provides probabilities
Decision Tree	Fairly expensive train Can provide poor accuracy Might perform bad with skewed data Small differences in data can provide very different trees Prove to over fit	Fast prediction Easy implementation Good for categorical data *Bad accuracy and overfitting can be solved with random forest

SVM	Mainly used for binary classification (but categorical possible) Does not provide probabilities (natively) Might overfit Computationally heavy	Can give fairly high accuracy Handles high dimensional data well
Neural network	Expensive training (depends on parameter choices) Can be hard to tweak optimal parameters	Known to perform well on image classifications Fast predictions Can provide very high accuracy CNN adds ability to see correlations in parts in images
Naive Bayes	Complicated to understand Builds on dependency assumptions, and will perform badly if assumption is not met	Very fast training and predictions

Multinomial Naive Bayes

Bayesian machine learning is an especially mathematical approach to machine learning. The core idea is to use Bayes formula to calculate probabilities for something to be of a class given parameters. I did a cross validation with three folds, and the following parameters using Sklearn GridSearchCV for the Multinomial Naive Bayes classifier. The ranges of these was qualified guesses. I had done some manual experimental testing already, and therefore I knew approx. where the values should lay.

```
parameters = {'alpha': [1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000], 'fit_prior': [True, False]}.
```

The following parameters gave the best model measured by accuracy.

```
parameters = {'alpha': 1e-06, 'fit_prior': True}
```

This table shows accuracy and timing for the different training and timing.

Type	Time	Accuracy
Cross validation	43 sec	0.828

Train best model with train set (length 49000)	0.311 sec	-
Predict best model with test set (length 21000)	0.1077 sec	0.823

We can see that this models is very fast at both fitting and predicting, the accuracy however is not that great. For the postal system the training time doesn't need to come close to this training speed. The sacrifice to this speed is that it will probably make 18 character misprediction for every 100 character it tries to predict. All in all this classifier is very fast and easy to implement, but it comes at the cost of accuracy. The accuracy is too low to perform in a good way for the postal system.

Decision Tree

Decision tree machine learning functions in the way that is makes a tree based on decisions. It makes decision nodes based on what separates the data in the best way (splits most even). I did a cross validation with three folds, and the following parameters using sklearn's GridSearchCV. The ranges of these was qualified guesses. I had done some manual experimental testing already, and therefore I knew approx. where the values should lay.

```
parameters = {'criterion': ['gini', 'entropy'], 'max_features': ['auto', 'log2', None], 'max_depth': [1, 10, 20, 30, None]}
```

The following parameters gave the best model measured by accuracy.

```
parameters = {'criterion': 'entropy', 'max_depth': None, 'max_features': None}
```

This table shows accuracy and timing for the different training and timing.

Type	Time	Accuracy
Cross validation	5 min 2 sec	0.864
Train best model with train set (length 49000)	16 sec	-
Predict best model with test set (length 21000)	0.0818 sec	0.875

We can see that this models trains the model in a very tolerable time, and predicts extremely fast. It's accuracy is still not great. It took a lot of parameter tweaking getting it to perform this good. All in all this model has an okay training time, with a very fast prediction. However it requires some tweaking getting it to provide acceptable accuracy. The will still make approximately 13 mispredictions for every 100 prediction attempts. This is still not good enough for the postal system.

Random Forest (ensemble)

A random forest classifier is in essence a classifier that creates a number of decision trees with different part of the training data. The data these are made of, the number of these and the voting system are the most important parameters that can be tweaked in this model. I did a cross validation with three folds, and the following parameters using sklearn's GridSearchCV. The ranges of these was qualified guesses. I had done some manual experimental testing already, and therefore I knew approx. where the values should lay.

```
parameters = {'n_estimators': [10, 30], 'max_features': ['auto', None], 'max_depth': [50, None], 'bootstrap': [True, False], 'oob_score': [False]}
```

The following parametres gave the best model measured by accuracy.

```
parameters = {'bootstrap': False, 'max_depth': None, 'max_features': 'auto', 'n_estimators': 30, 'oob_score': False}
```

This table shows accuracy and timing for the different training and timing.

Type	Time	Accuracy
Cross validation	2 hours 7 min 34 sec	0.963
Train best model with train set (length 49000)	18 sec	-
Predict best model with test set (length 21000)	0.4936 sec	0.965

This model does training in a fairly short time, and predicts quite fast. Also the accuracy is quite good. The model has a lot of parameters one can tweak, but if you focus on the most important features such as the number of trees and their max

depth then the model is easy to implement. This could be a good alternative for the postal system.

Custom Neural Network

A neural network is a model strongly inspired by how a human brain works. It creates layers of multiple neurons, which is capable of seeing correlations between features and target values, and this is done where some features is more important than other.

This neural network was implemented using the Keras package. This made the testing procedure slightly different from the Sklearn based models. I wanted to implement a cross validation function for the neural network so that I could compare these values with the ones from the Sklearn results. Therefore I ended up making my own cross val. structure, also using a dictionary for testing multiple parameters together in one go. I later found out that parts of this would be possible when wrapping a Keras model and passing it to some Sklearn package that could perform cross val. scores. However I had already made my testing pipeline, and decided to stick with it.

I did a lot of testing with different values using cross val. on the neural network. I sat up computer to run these overnight, but it did unfortunately crash around 4 AM, one of the last nights I had. This was due to some deadlock/infinite loop in my code or an error/bug in the system or Python. There was no error code in terminal, and the program appeared to still be running, but not producing any results during 6 hours of running. It probably ran for around 4-5 hours before the crash. My test results does for that reason not contain all the results for all parameters, but I got a cross val. score of 0.9945 and decided this was good enough.

The plan was for it to execute cross val. for the following parameters. The ranges of these was qualified guesses. I had done some manual experimental testing already, and therefore I knew approx. where the values should lay. I could also included other parameters such as activation and drop outs, but I saw these as less important.

```
parameters = {'n_layers': [2, 6, 10, 14], 'epochs': [10, 20, 30], 'batch_size': [80, 100, 120], 'n_neurons': [10, 80, 150, 250]}
```

This results in 144 cross validations, with each build spending a significant amount of time. That explains why I was satisfied with the results I got. The following parameters is what gave me the cross val. score of 0.9945.

```
parameters = {'n_layers': 2, 'epochs': 20, 'batch_size': 80, 'n_neurons': 250}
```

Type	Time	Accuracy
Cross validation	unknown (crash after 4-5 hours)	0.994
Train best model with train set (length 49000)	4 min 09 sec	-
Predict best model with test set (length 21000)	2 sec	0.996

As we can see this model takes some time to train, but predictions is fairly fast. The train time is very acceptable considering the size of our data set. The accuracy is also very good, almost as if it is overfitting. However I would assume that it isn't overfitting too bad, as both the cross val. score is consistently high, and the predictions on the test set is also very good. This model can for these reasons fit our postal office system very well.

Choosing the best model for the postal system

All of the model reports contains a small summary for it's performance. It's pretty clear that the bayes model and the decision tree model are not performing well enough for this project. They are simply not accurate enough. They are however very fast to predict, and the bayes model is also extremely fast at training. The accuracy tradeoff might be accepted for some tasks where speed is very important and accuracy not as critical. In this postal system however the accuracy is not good enough, and we don't need prediction time this fast. Both the random forest model and the neural network offer speeds well within what the task requires. When we compare these two models then we see that the neural network performs slightly better at accuracy, but slightly worse in terms of speed. The neural network is still the best candidate for the postal system because of its great accuracy.

How well will the neural network perform in the real world?

This task has been done using the MNIST dataset. It contains a lot of diversity in how the digits are written, and can be considered real world data. It's also very big, so the models has had a lot of data to train on. I'm therefore quite convinced that the neural network would also perform well with new unseen data. After all it did very well on the test set, which was unseen.

The main concern for how the model would function in the real world is rotation and locating the digits in a larger image than 28 x 28 pixels. The model has after all mostly seen digits centered and with correct rotation. If you for example try to predict a slightly tilted "1" then it will, with great probability, predict it to a "7". One way to solve this problem would be to train the model on digits that are tilted and uncentered (brute force method). These data could be generated by modifying some of the data from MNIST. The big problem with this is that it would make the model a lot more advanced than it needs to be, thus increasing computation time. The other main way to solve this problem is to ensure that the input data is of uniform measures. I will not go deep into how this could be achieved, but there exists methods for achieving this, many involving additional machine learning (on a higher level for example). Therefore I'm quite convinced that my neural network would perform well if it's given uniform data.

Even though the neural network can predict digits with fairly high accuracy there will always be digits it can't predict right. I envision that such a postal system could have a failsafe method for character predictions. It could search its register over addresses and people, and if no hits were found then it would lead to a new prediction using either human prediction or using another prediction model. It would still not be perfect as we might not have all people in our registry (example impossible to keep track of all the world's population is the system should work international).

What would I change if there was more time

There are a few changes that I would have made if I had more time. Some of the aspects could also be solved with greater computational power.

I would like to implement CNN in addition to my normal neural network. It's known to increase the accuracy in image recognition. It uses a technique called filtering which makes it better at recognizing features isolated from each other in pictures. I managed to tweak the normal neural network to give a high accuracy, but a CNN might have performed better in the real world on aspects discussed in the section about performance in real world use.


With more time (and significantly more computational power) then I would have tweaked the models with more variables for the parameter for each model when performing the cross validation. When finding approx. where the parameters should lay, then it would also be cool to interpolate around these areas, and in this way tweak the model even more. It was mentioned by one of the TAs during class that testing out five different values for each parameter would be a good starting point when tweaking the models. This however would generate enormous amounts of training and prediction test. Let's say for example that a model has four important parameters, and each of them should be tested with five different values. That would lead to $5^4 = 625$ different tests, again multiplying with n folds in

the cross val. I concluded that it was not relevant in this project to do that many tests as it would take an enormous amount of time to conduct. I stuck with the parameters stated in each report (for each model). With significantly more computational power or a lot more time then it would be cool to do this many tests. One could decrease the dataset size when doing the parameter tuning. I considered this, but decided to do the tests with the whole train set anyway. I made that decision because I wanted my models to be as accurate as possible (even in the cross validation). In some of the models there is also a strong correlation between what parameters performs the best and the size of the training set (example random forest n estimators). Also I stuck with three folds while cross validation. I decided this was adequate here, but with more power and time I might have increased this to 4-5 folds for a more precise cross val score.

With more time I would also do deeper research in what role the different parameters for the respective models play, and go deeper into the correlation between them. In this project I had some idea to what values would perform well. Then I did some additional experimentation before I decided on what parameter values to test for in the grid search cross validation. One can say that my experimentation was in a way based on empirical data when I was deciding on values for the different values for the parameters, and what parameters to tweak in the first place.

Model testing logs

Multinomial Naive Bayes

 bayes_scores - Notisblokk

Fil Rediger Format Vis Hjelp

Time started 2018-10-24 13:10:38.196998

Time ended 2018-10-24 13:11:21.262760

Time elapsed 0:00:43.065762

Best score 0.8284081632653061 with following parameters {'alpha': 1e-06, 'fit_prior': True}

MultinomialNB(alpha=1e-06, class_prior=None, fit_prior=True)

All tests

```

0.828 (+/-0.009) for {'alpha': 1e-06, 'fit_prior': True}
0.828 (+/-0.009) for {'alpha': 1e-06, 'fit_prior': False}
0.828 (+/-0.009) for {'alpha': 1e-05, 'fit_prior': True}
0.828 (+/-0.009) for {'alpha': 1e-05, 'fit_prior': False}
0.828 (+/-0.009) for {'alpha': 0.0001, 'fit_prior': True}
0.828 (+/-0.009) for {'alpha': 0.0001, 'fit_prior': False}
0.828 (+/-0.009) for {'alpha': 0.001, 'fit_prior': True}
0.828 (+/-0.009) for {'alpha': 0.001, 'fit_prior': False}
0.828 (+/-0.009) for {'alpha': 0.01, 'fit_prior': True}
0.828 (+/-0.009) for {'alpha': 0.01, 'fit_prior': False}
0.828 (+/-0.009) for {'alpha': 0.1, 'fit_prior': True}
0.828 (+/-0.009) for {'alpha': 0.1, 'fit_prior': False}
0.828 (+/-0.009) for {'alpha': 1, 'fit_prior': True}
0.828 (+/-0.009) for {'alpha': 1, 'fit_prior': False}
0.828 (+/-0.009) for {'alpha': 10, 'fit_prior': True}
0.828 (+/-0.009) for {'alpha': 10, 'fit_prior': False}
0.828 (+/-0.009) for {'alpha': 100, 'fit_prior': True}
0.828 (+/-0.009) for {'alpha': 100, 'fit_prior': False}
0.826 (+/-0.010) for {'alpha': 1000, 'fit_prior': True}
0.826 (+/-0.010) for {'alpha': 1000, 'fit_prior': False}
0.820 (+/-0.009) for {'alpha': 10000, 'fit_prior': True}
0.820 (+/-0.009) for {'alpha': 10000, 'fit_prior': False}
0.790 (+/-0.010) for {'alpha': 100000, 'fit_prior': True}
0.790 (+/-0.010) for {'alpha': 100000, 'fit_prior': False}

```

Data for best model:

Train data fit time 0:00:00.311167 train data length 49000

Test data predict time 0:00:00.106714 test data length 21000

Accuracy was 0.8233333333333334

Report

	precision	recall	f1-score	support
0	0.91	0.91	0.91	2008
1	0.89	0.93	0.91	2369
2	0.88	0.83	0.86	2103
3	0.79	0.82	0.80	2103
4	0.84	0.74	0.78	2105
5	0.85	0.65	0.73	1950
6	0.88	0.91	0.90	2070
7	0.95	0.83	0.88	2139
8	0.66	0.77	0.71	2064
9	0.67	0.82	0.74	2089
avg / total	0.83	0.82	0.82	21000

Decision tree

dtree_scores - Notisblokk

Fil Rediger Format Vis Hjelp

Time started 2018-10-24 13:34:11.417596

Time ended 2018-10-24 13:39:13.762941

Time elapsed 0:05:02.345345

Best score 0.8644693877551021 with following parameters {'criterion': 'entropy', 'max_depth': None, 'max_features': None}

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                        splitter='best')
```

All tests

```
0.191 (+/-0.008) for {'criterion': 'gini', 'max_depth': 1, 'max_features': 'auto'}
0.195 (+/-0.018) for {'criterion': 'gini', 'max_depth': 1, 'max_features': 'log2'}
0.198 (+/-0.007) for {'criterion': 'gini', 'max_depth': 1, 'max_features': None}
0.788 (+/-0.002) for {'criterion': 'gini', 'max_depth': 10, 'max_features': 'auto'}
0.689 (+/-0.069) for {'criterion': 'gini', 'max_depth': 10, 'max_features': 'log2'}
0.846 (+/-0.003) for {'criterion': 'gini', 'max_depth': 10, 'max_features': None}
0.816 (+/-0.008) for {'criterion': 'gini', 'max_depth': 20, 'max_features': 'auto'}
0.765 (+/-0.014) for {'criterion': 'gini', 'max_depth': 20, 'max_features': 'log2'}
0.856 (+/-0.003) for {'criterion': 'gini', 'max_depth': 20, 'max_features': None}
0.817 (+/-0.006) for {'criterion': 'gini', 'max_depth': 30, 'max_features': 'auto'}
0.764 (+/-0.014) for {'criterion': 'gini', 'max_depth': 30, 'max_features': 'log2'}
0.855 (+/-0.006) for {'criterion': 'gini', 'max_depth': 30, 'max_features': None}
0.812 (+/-0.005) for {'criterion': 'gini', 'max_depth': None, 'max_features': 'auto'}
0.773 (+/-0.010) for {'criterion': 'gini', 'max_depth': None, 'max_features': 'log2'}
0.855 (+/-0.007) for {'criterion': 'gini', 'max_depth': None, 'max_features': None}
0.203 (+/-0.006) for {'criterion': 'entropy', 'max_depth': 1, 'max_features': 'auto'}
0.180 (+/-0.016) for {'criterion': 'entropy', 'max_depth': 1, 'max_features': 'log2'}
0.206 (+/-0.001) for {'criterion': 'entropy', 'max_depth': 1, 'max_features': None}
0.789 (+/-0.005) for {'criterion': 'entropy', 'max_depth': 10, 'max_features': 'auto'}
0.699 (+/-0.022) for {'criterion': 'entropy', 'max_depth': 10, 'max_features': 'log2'}
0.859 (+/-0.010) for {'criterion': 'entropy', 'max_depth': 10, 'max_features': None}
0.823 (+/-0.006) for {'criterion': 'entropy', 'max_depth': 20, 'max_features': 'auto'}
0.776 (+/-0.004) for {'criterion': 'entropy', 'max_depth': 20, 'max_features': 'log2'}
0.863 (+/-0.009) for {'criterion': 'entropy', 'max_depth': 20, 'max_features': None}
0.823 (+/-0.012) for {'criterion': 'entropy', 'max_depth': 30, 'max_features': 'auto'}
0.767 (+/-0.013) for {'criterion': 'entropy', 'max_depth': 30, 'max_features': 'log2'}
0.864 (+/-0.008) for {'criterion': 'entropy', 'max_depth': 30, 'max_features': None}
0.820 (+/-0.006) for {'criterion': 'entropy', 'max_depth': None, 'max_features': 'auto'}
0.772 (+/-0.012) for {'criterion': 'entropy', 'max_depth': None, 'max_features': 'log2'}
0.864 (+/-0.005) for {'criterion': 'entropy', 'max_depth': None, 'max_features': None}
```

Data for best model:

Train data fit time 0:00:16.254643 train data length 49000

Test data predict time 0:00:00.081782 test data length 21000

Accuracy was 0.8748571428571429

Report

	precision	recall	f1-score	support
0	0.93	0.93	0.93	2008
1	0.95	0.96	0.95	2369
2	0.87	0.85	0.86	2103
3	0.82	0.84	0.83	2103
4	0.87	0.87	0.87	2105
5	0.82	0.82	0.82	1950
6	0.90	0.90	0.90	2070
7	0.91	0.91	0.91	2139
8	0.83	0.81	0.82	2064
9	0.82	0.83	0.83	2089
avg / total	0.87	0.87	0.87	21000

Random Forest


```

randomtree_scores - Notisblokk
Fil Rediger Format Vis Hjelp
Time started 2018-10-25 10:41:03.467509
Time ended 2018-10-25 12:48:37.606473
Time elapsed 2:07:34.138964
Best score 0.9629183673469388 with following parameters {'bootstrap': False, 'max_depth': None, 'max_features': 'auto', 'n_estimators': 30, 'oob_score': False}
RandomForestClassifier(bootstrap=False, class_weight=None, criterion='gini',
                        max_depth=None, max_features='auto', max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=30, n_jobs=1,
                        oob_score=False, random_state=None, verbose=0,
                        warm_start=False)

All tests
0.938 (+/-0.005) for {'bootstrap': True, 'max_depth': 50, 'max_features': 'auto', 'n_estimators': 10, 'oob_score': False}
0.957 (+/-0.005) for {'bootstrap': True, 'max_depth': 50, 'max_features': 'auto', 'n_estimators': 30, 'oob_score': False}
0.924 (+/-0.013) for {'bootstrap': True, 'max_depth': 50, 'max_features': None, 'n_estimators': 10, 'oob_score': False}
0.941 (+/-0.010) for {'bootstrap': True, 'max_depth': 50, 'max_features': None, 'n_estimators': 30, 'oob_score': False}
0.940 (+/-0.003) for {'bootstrap': True, 'max_depth': None, 'max_features': 'auto', 'n_estimators': 10, 'oob_score': False}
0.958 (+/-0.005) for {'bootstrap': True, 'max_depth': None, 'max_features': 'auto', 'n_estimators': 30, 'oob_score': False}
0.929 (+/-0.004) for {'bootstrap': True, 'max_depth': None, 'max_features': None, 'n_estimators': 10, 'oob_score': False}
0.940 (+/-0.007) for {'bootstrap': True, 'max_depth': None, 'max_features': None, 'n_estimators': 30, 'oob_score': False}
0.947 (+/-0.005) for {'bootstrap': False, 'max_depth': 50, 'max_features': 'auto', 'n_estimators': 10, 'oob_score': False}
0.962 (+/-0.003) for {'bootstrap': False, 'max_depth': 50, 'max_features': 'auto', 'n_estimators': 30, 'oob_score': False}
0.867 (+/-0.003) for {'bootstrap': False, 'max_depth': 50, 'max_features': None, 'n_estimators': 10, 'oob_score': False}
0.870 (+/-0.002) for {'bootstrap': False, 'max_depth': 50, 'max_features': None, 'n_estimators': 30, 'oob_score': False}
0.946 (+/-0.003) for {'bootstrap': False, 'max_depth': None, 'max_features': 'auto', 'n_estimators': 10, 'oob_score': False}
0.963 (+/-0.003) for {'bootstrap': False, 'max_depth': None, 'max_features': 'auto', 'n_estimators': 30, 'oob_score': False}
0.867 (+/-0.002) for {'bootstrap': False, 'max_depth': None, 'max_features': None, 'n_estimators': 10, 'oob_score': False}
0.869 (+/-0.005) for {'bootstrap': False, 'max_depth': None, 'max_features': None, 'n_estimators': 30, 'oob_score': False}

Data for best model:
Train data fit time 0:00:18.108575 train data length 49000
Test data predict time 0:00:00.493646 test data length 21000
Accuracy was 0.9648095238095238
Report

```

	precision	recall	f1-score	support
0	0.97	0.99	0.98	2008
1	0.98	0.98	0.98	2369
2	0.96	0.97	0.96	2103
3	0.96	0.96	0.96	2103
4	0.96	0.97	0.96	2105
5	0.97	0.95	0.96	1950
6	0.98	0.97	0.98	2070
7	0.98	0.96	0.97	2139
8	0.96	0.95	0.95	2064
9	0.94	0.94	0.94	2089
avg / total	0.96	0.96	0.96	21000

Custom Neural network

The full cross val. test report is too big for this pdf, but can be found as an additional file called *customNN_report.txt*. The following report is for the model based on the best parameters.

```

Training a model with trainset using following parameters, and testing with test set
parameters = {'n_layers': 2, 'epochs': 20, 'batch_size': 80, 'n_neurons': 250}
Time fit 0:04:09.495405
Time evaluate 0:00:01.946680
Score for best model 0.9960333281017485

```