Handwritten Digits Recognition Report

Pattern Recognition 2019 Assignment [INFOMPR]

Bjorn Bruns (6980341) Mikhail Ternyuk (6853390)

---

1. Exploratory analysis

For this pattern recognition assignment we used Python 3.7 with the following machine learning/data science packages: **`sklearn`**, **`tensorflow`**, **`keras`**, **`pandas`**, **`numpy`**, **`matplotlib`**, **`seaborn`**, **`scipy`** and **`statsmodels`**.

At the first step of the analysis we created a DataFrame from the 'mnist.csv' file. The dataset contains 42000 rows and 785 columns. The "label" column is our target for the predictions (the digit) and the other 784 columns are features (pixels). To check that all values are present, we printed a description of the created DataFrame:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
```

It shows that the DataFrame contains 785 columns (label, pixel0 to pixel783) and all values are of type int64. A few examples of the digits are:

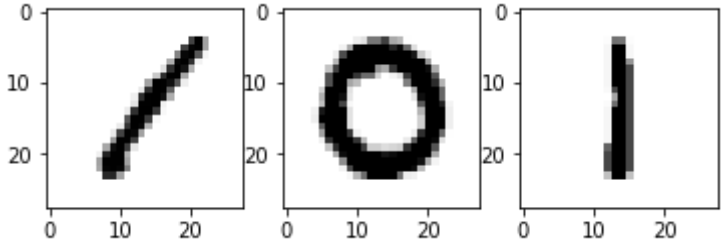


Figure 1. *Examples of handwritten digits of 28x28 pixels from the dataset.*

1.1 Spotting useless features

After a first glance at the pictures, we decided to check whether there are pixels (columns) that have a value of zero for all the rows in our dataset. Those features can be considered as useless. See below some examples of digits, where the black edges are useless features we found (i.e. these pixels aren't used in any of the rows):

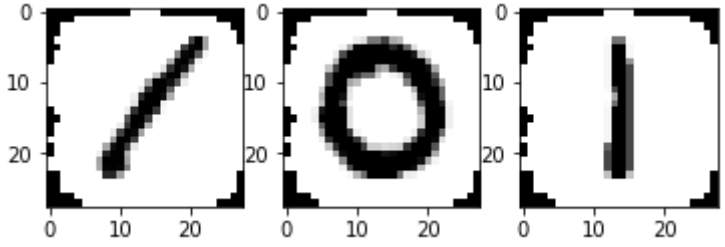


Figure 2. *Examples of handwritten digits with the useless features in black at the edges.*

In total we found 76 features which were useless. These were the following pixels:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 16, 17, 18, 19, 20, 21, 22, 23, 24,
25, 26, 27, 28, 29, 30, 31, 52, 53, 54, 55, 56, 57, 82, 83, 84, 85, 111,
112, 139, 140, 141, 168, 196, 392, 420, 421, 448, 476, 532, 560, 644, 645,
671, 672, 673, 699, 700, 701, 727, 728, 729, 730, 731, 754, 755, 756, 757,
758, 759, 760, 780, 781, 782, 783]
```

We can delete these features from the dataset and this will not affect the performance of our classifiers.

1.2 Class distribution

In order to see the distribution of the data over all classes (digits), we have created a histogram:

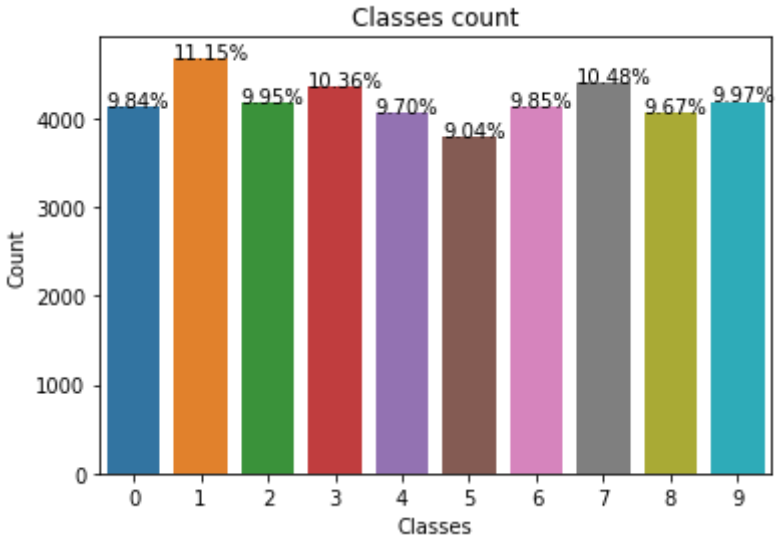


Figure 3: *Histogram with the frequency of all classes, and the percentage of this class as a part of the whole dataset.*

From the histogram we can see that label ‘1’ is the majority class in this dataset. This would mean that, if we simply predict the majority class, about 11.15% of all cases would be classified correctly.

1.3 Other findings of interest

1. Every pixel of a picture has an intensity scaled from 0 - completely white, to 255 - completely black. But at pictures we can also notice grey pixels, i.e. pixels with a value of neither 0 nor 255. We could consider the grey values as “noisy”. We can see the distribution of all nonzero features by building the distribution plot below.

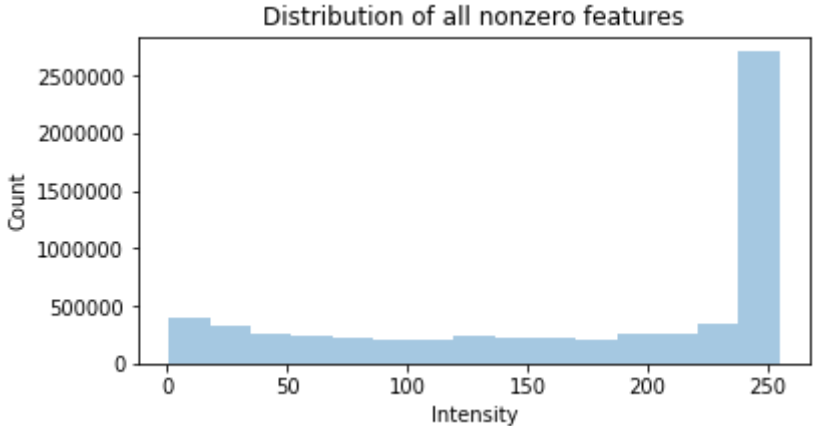


Figure 4. *Distribution of pixel occurence for nonzero values over all data points. N bins = 15.*

We can clean this “noise” by making all pixels binary (0 or 1), to improve distinguishment between values.

2. Some of the digits look symmetric (0, 1, 8). We can calculate the symmetry and use it as a new feature.

3. Also, we can calculate the average value for each pixel for a certain digit (class). These averaged images are called centroids. Then, when we have a new example, we can calculate the distance from this example to the centroids for all digits (classes). This would then create 10 new features, the distances to all digits (classes).

4. We notice that some digits contain circle-like shapes (e.g. 0, 8, 9), whether the digit contains this shape or not could be used as a new feature.

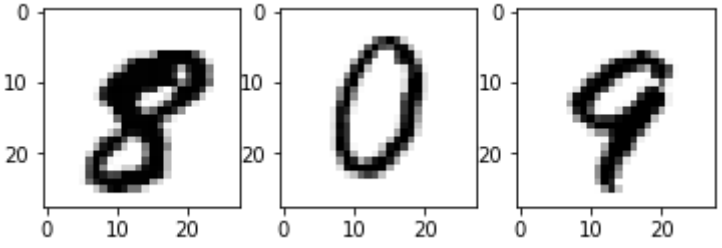


Figure 5. *Examples of handwritten digits with circle-like shapes.*

2. Density feature, multinomial logit model

For calculating the density of a digit we used the `astype` method on the DataFrame object, which turns all non-zero values into ones. After this, we added all values in the rows together with the function `sum()`:

```
df["density"] = df.iloc[:, 1:].astype(bool).sum(axis=1)
```

2.1. Average and standard deviation

For calculating the average and standard deviation of the density feature, we used the default functions `mean()` and `std()` on all the instances per class in the DataFrame:

Label (digit)	Mean	Standard deviation	Label (digit)	Mean	Standard deviation
0	191.74	33.70	5	152.76	33.77
1	85.73	20.01	6	158.21	32.66
2	168.97	33.06	7	131.42	27.03
3	163.52	33.64	8	173.97	32.84
4	141.50	27.92	9	143.25	28.54

Table 1. *Mean density and standard deviation for all classes.*

To visualise the difference of density in between classes, we created a barplot with the means of all digits and the standard deviation as error bars. We have also calculated the maximum and minimum differences between mean density values:

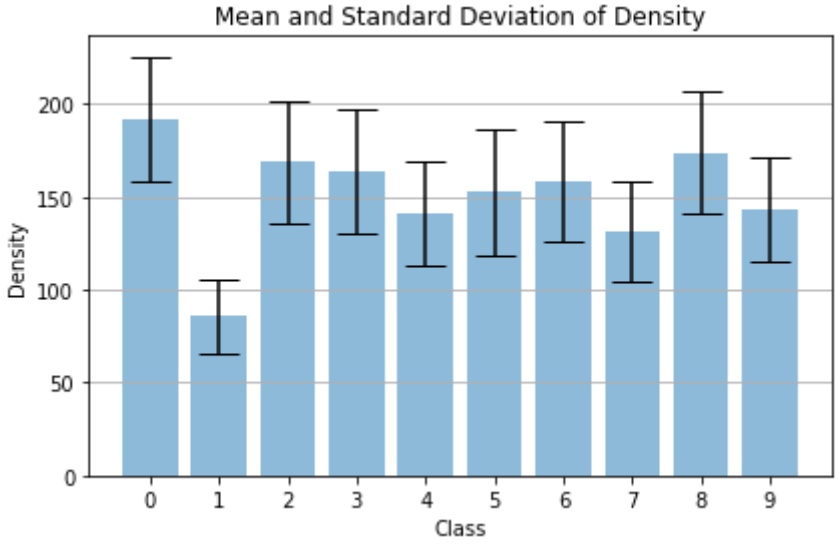


Figure 6. *Barplot with means of density and the standard deviation as an error bar.*

Maximum difference in mean density: 106.01 (0 and 1)
Minimum difference in mean density: 1.75 (4 and 9)

From these statistics we can assume that the density feature is able to distinguish the classes ‘0’ and ‘1’ quite well. However, distinguishing ‘4’ and ‘9’ would be challenging by using this feature.

2.2 Multinomial logit model

To build a multinomial logit model with `sklearn` we used the `LogisticRegression` function, with the `saga` solver. Firstly, we scaled all density values with `StandardScaler`, then divided all data in train and test sets (25% test, 75% train). After training the model, we achieved 26% accuracy on the test set.

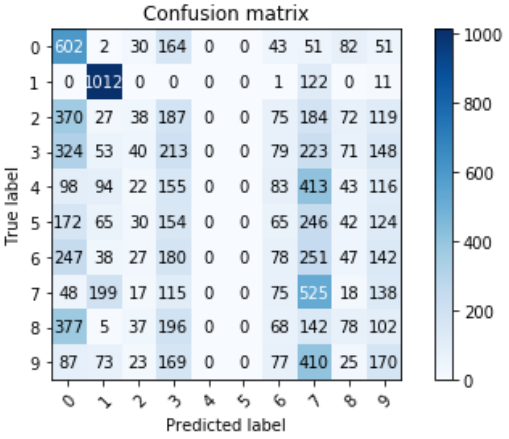


Figure 7. *Confusion matrix for true/predicted labels for all classes with the multinomial logit model using the density feature.*

Looking at the confusion matrix above, we can draw some interesting conclusions. We see that the model fails to predict the label ‘4’ and ‘5’. It performs very well on predicting digits with the label ‘1’, which was expected when we looked at the density means of the classes. It's also notable that ‘2’ is almost never predicted correctly. It is also interesting to see that the labels ‘0’ and ‘7’ are predicted a lot, for almost all classes.

The best distinguished pairs of classes were ‘1’ and ‘0’, only 2 examples were predicted wrong. The pair ‘1’ and ‘8’ also distinguished quite well, only 5 examples of the digit ‘8’ were classified as ‘1’. Please note that it distinguishes well between ‘1’ and ‘8’, but doesn't perform very well in predicting ‘8’ when the true label is ‘8’.

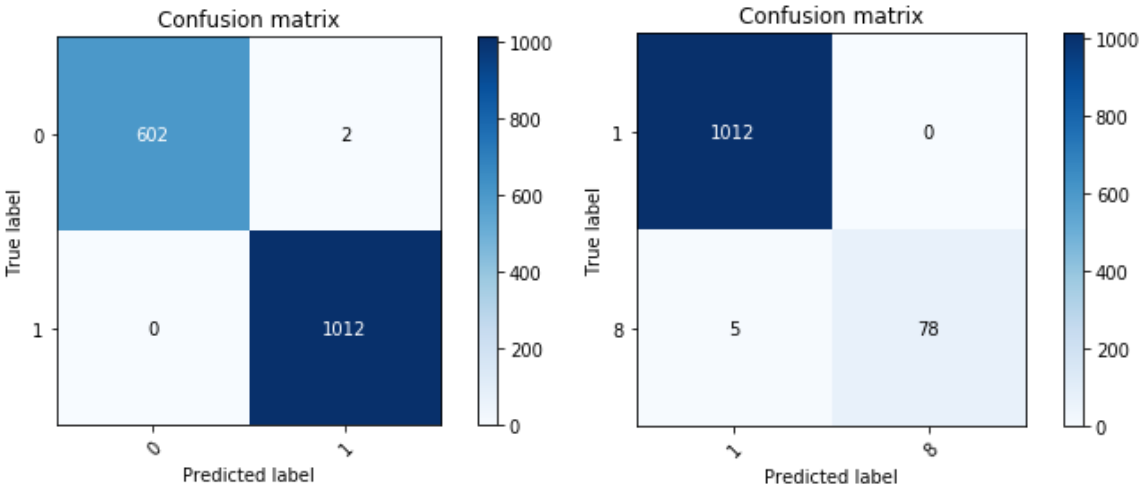


Figure 8 (left). *Confusion matrix for true/predicted labels for classes 0 and 1 with the multinomial logit model using the density feature.*

Figure 9. *Confusion matrix for true/predicted labels for classes 1 and 8 with the multinomial logit model using the density feature.*

At the same time, the classifier has difficulties with separating classes ‘8’ and ‘3’, almost 200 examples of ‘8’ were predicted as ‘3’.

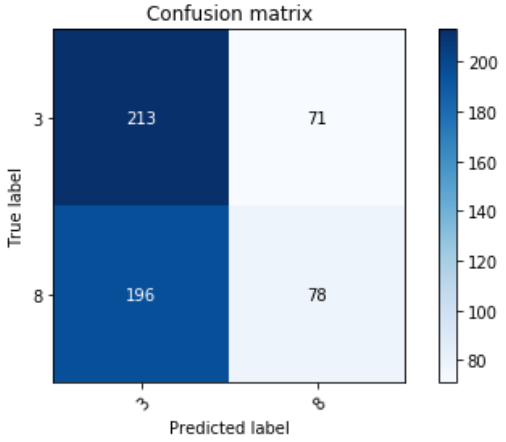


Figure 10. *Confusion matrix for true/predicted labels for classes 3 and 8 with the multinomial logit model using the density feature.*

3. New feature: symmetry, multinomial logit model

According to previous steps, density is quite a good feature to discriminate labels ‘1’ and ‘0’ or ‘1’ and ‘8’. Our goal now is to choose a feature that clearly distinguishes other groups of digits. This feature should not be correlated with the density feature. If we look again at the classes, we can see that the digits ‘1’, ‘8’ and ‘0’ have a mostly symmetric shape, where labels ‘7’ and ‘4’ do not have this.

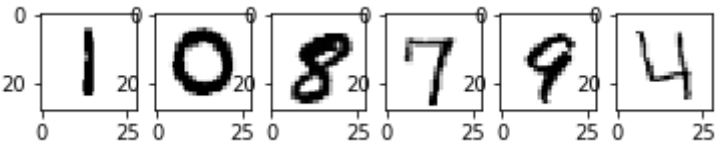


Figure 11. *Examples of handwritten digits with (non-)symmetric shapes.*

Symmetry seems to be a good discriminator for these kind of classes and supposedly improve our classifier.

3.1 Deriving new feature

There are several ways to determine symmetry in a picture. The most straightforward ways to do this are horizontal or vertical. However, with horizontal and vertical symmetry we didn't achieve the best performance. We also tested several other methods to calculate the symmetry and the one we found with the best performance was "mirrored horizontal" symmetry.

This method compares the two halves (horizontally divided) of the picture, where the bottom half is vertically mirrored. We compared pixels one-by-one starting from both sides of the array. When both pixels have a nonzero intensity, we count these as 2 matches. After doing this for all pixels, we divide the sum of these counted matches by the density of the digit (i.e. the sum of all nonzero pixels). This means that we're checking whether the top-left pixel (pixel0) matches with the bottom-right pixel (pixel783). Next, we compare the pixel at the right of the top-left pixel (pixel1) and at the left of the bottom-right one (pixel782), until all pixels are compared (at pixel391 and pixel392).

3.1.1 Calculation of symmetry

Please note that the 'index' is an instance of a digit (row in the DataFrame) for which we are calculating the symmetry.

```
i = 0 # Pixel counter from the start of pixels array
j = 783 # Pixel counter from the end of pixel array

digit = np.array(DF.iloc[[index], 1:])[0] # Isolating an instance
(index) of a digit from the dataset and creating array of pixels

digit_density = digit.astype(bool).sum() # Calculating density, i.e.
the number of nonzero pixels

# Main loop, where we check if both pixels have a non-zero value
while i < 392:
    if digit[i] != 0 and digit[j] != 0:
        symmetry_level += 2 # counting symmetric pixels
    i += 1
    j -= 1
# calculating symmetry feature score
symmetry = symmetry_level/digit_density
```

3.2 Analysis of symmetry feature

For calculating the average and standard deviation of the symmetry feature, we used the default functions `mean()` and `std()` on all the instances per class in the DataFrame:

Label (digit)	Mean	Standard deviation	Label (digit)	Mean	Standard deviation
0	0.60	0.14	5	0.49	0.14
1	0.64	0.16	6	0.48	0.10
2	0.51	0.14	7	0.34	0.11
3	0.51	0.12	8	0.60	0.11
4	0.52	0.12	9	0.52	0.09

Table 2. *Mean symmetry and standard deviation for all classes.*

To visualise the difference of symmetry in between classes, we created a barplot and also calculated the maximum and minimum differences between mean symmetry values:

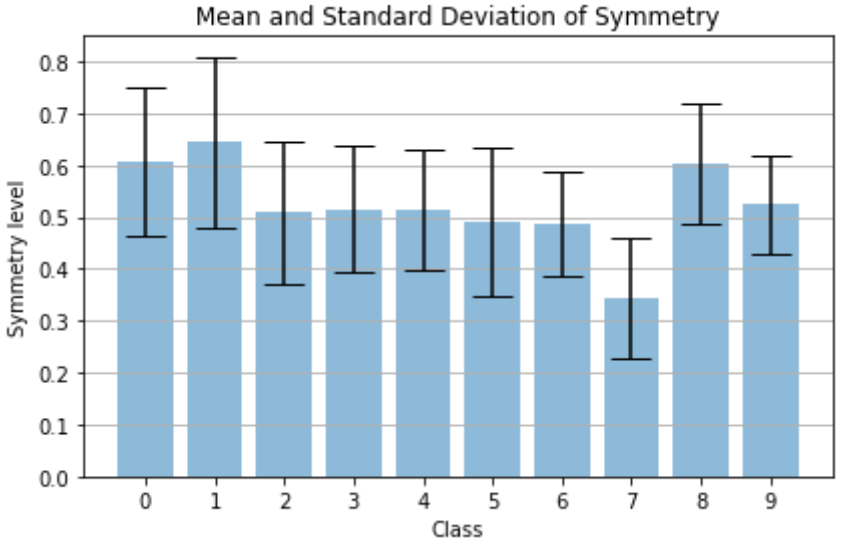


Figure 12. *Barplot with means of symmetry and the standard deviation as an error bar.*

Maximum difference in mean symmetry: 0.30 (7 and 1)
Minimum difference in mean symmetry: 0.002 (4 and 3)

From these statistics we can assume that the symmetry feature is able to distinguish the label '7' quite well from the labels '0', '1' and '8'. This is a good sign, because we created a feature which is not completely similar to the density feature. However, distinguishing '2', '3', '4', '5', '6', and '9' from one another would be harder by using this feature.

3.3 Multinomial logit model for symmetry

We used the same model we used for the density based classifier, the `LogisticRegression` function with the `saga` solver. We divided all data in train and test sets (25% test, 75% train). After training the model, we achieved a 20% accuracy on the test set.

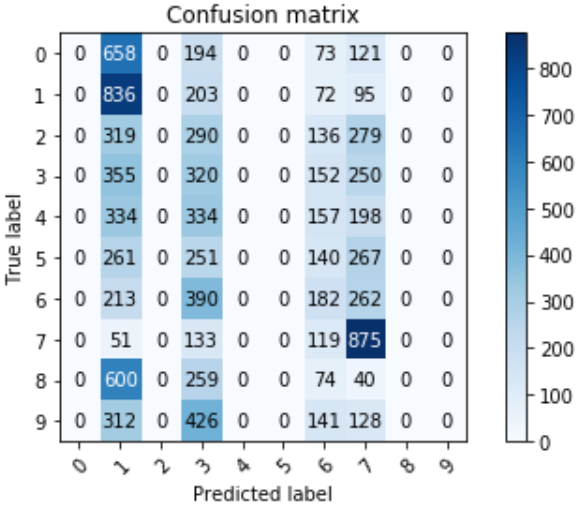


Figure 13. *Confusion matrix for true/predicted labels for all classes with the multinomial logit model using the symmetry feature.*

Looking at the confusion matrix above, we can draw some interesting conclusions. We see that the model fails to predict the label 0, 1, 4, 5, 8 and 9. It performs very well on predicting digits with the label 7, which was expected when we looked at the symmetry means of the classes. It is also interesting to see that the labels 1 and 3 are predicted quite a lot. Label 1 is predicted for the true labels 0 and 8 as well, most likely because all these digits are symmetric

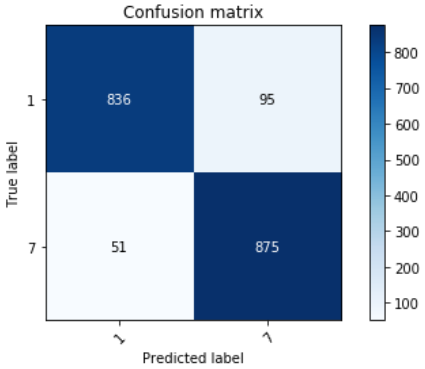


Figure 14. *Confusion matrix for true/predicted labels for classes 1 and 7 with the multinomial logit model using the symmetry feature.*

At the same time classifier has difficulties with separating classes 0, 1 and 8. They all predicted as class 1, but amounts of examples of each class almost the same (658 with label 0, 600 with label 8).

As a result, we achieved our main goal - discriminate label 7 from labels 0 and 1. We can distinguish between 0 and 1 by using density very well.

4. Density and symmetry combined, multinomial logit model

To build a multinomial logit model with `sklearn` we used the same `LogisticRegression` function as in the previous models, with the `saga` solver. Firstly, we scaled all density values with `StandardScaler`, then divided all data in train and test sets (25% test, 75% train).

Considering previous steps, our new classifier with both density and symmetry features is expected to show the best performance on pairs of labels 0-1, 1-7, 7-8 and 7-9:

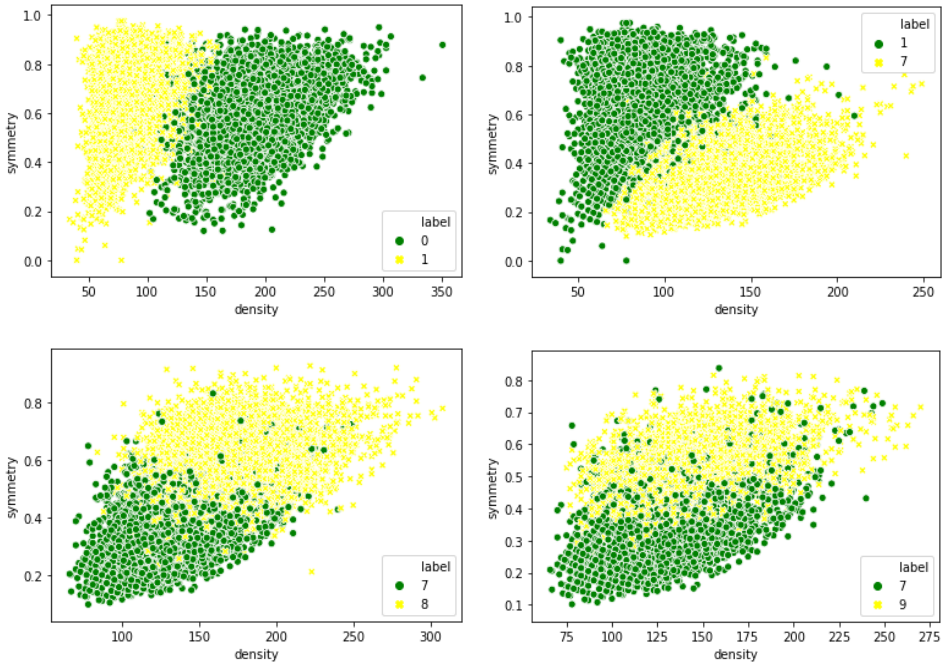


Figure 15. *Scatter plots showing both the symmetry and density feature values for different pairs of digits.*

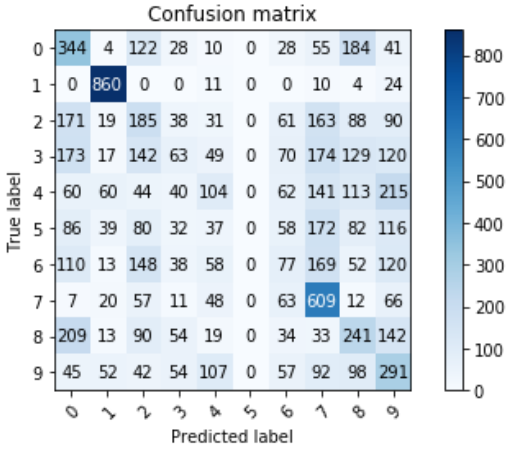


Figure 16. *Confusion matrix for true/predicted labels for all classes with the multinomial logit model using both the density and symmetry features.*

After training the model, we achieved a 33% accuracy on the test set. Compared to the model with only the density feature we have got a 7% (percentage point) increase and with only the symmetry feature we have a 13% higher performance.

For better understanding how exactly single-feature models were improved by adding a second feature, we have to compare all 3 models. The original model with density feature showed 25% accuracy. The classifier distinguished the numbers 1 and 0 very well, and we also managed to achieve a high accuracy when distinguishing the pair 1 and 8. We made an attempt to choose another feature that would allow us to distinguish digits that were different in structure. Our model with the symmetry feature made it possible to isolate class 7 with respect to 1, 0 and 8, which increased the performance of the original model:

Density feature model **Two-features model**

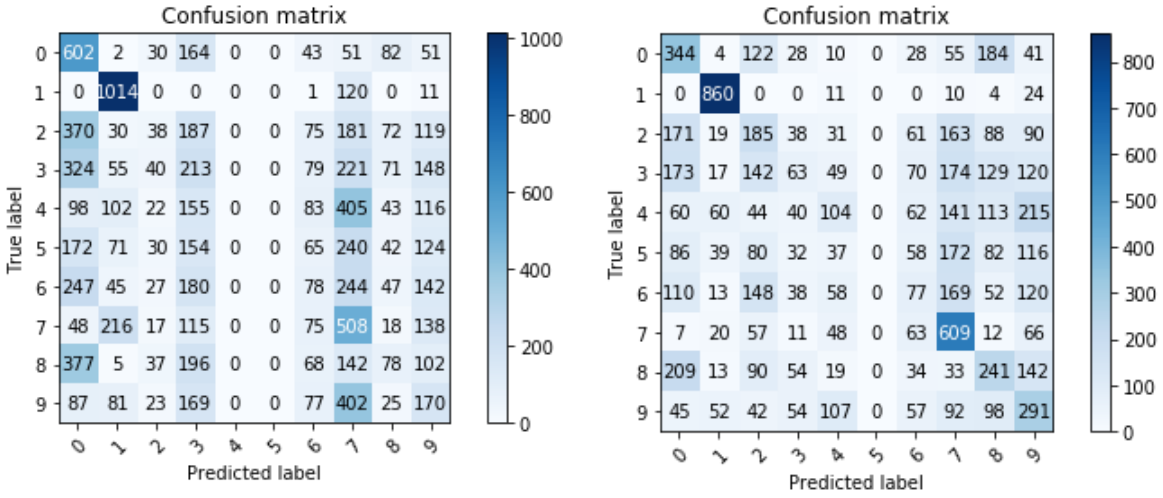


Figure 17. *Confusion matrices for true/predicted labels for all classes with the multinomial logit model using the density feature (left) vs. symmetry and density features (right).*

Symmetry feature model **Two-features model**

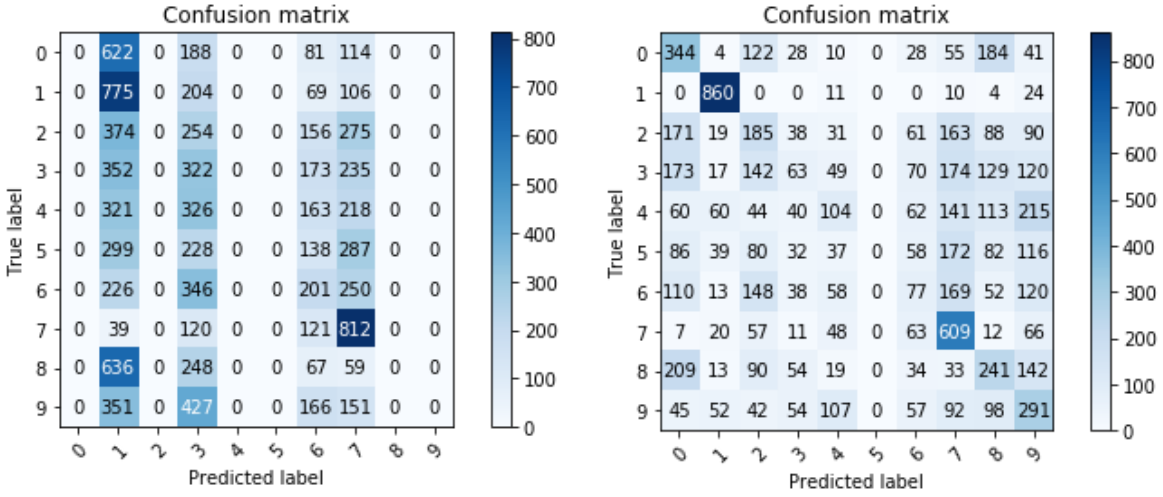


Figure 18. *Confusion matrices for true/predicted labels for all classes with the multinomial logit model using the symmetry feature (left) vs. symmetry and density features (right).*

The main achievement of the model with 2 features is preservation of the initial results of the density model. Adding the symmetry feature doesn't decrease the ability to distinguish the pairs '1' and '0' or '1' and '8', which we were able to distinguish with the density feature alone. It also achieved good results at distinguishing class '7'. It should also be noted that, in the combined model, pairs of digits '9' and '7', '9' and '1' are quite well distinguishable. Altogether that resulted in a 33% accuracy.

5. All pixel features, multinomial logit model, SVM, feed-forward NN

We have analysed the full dataset (5000 train, 37000 test data points) with 784 pixel values as features with a regularized multinomial logit model with LASSO penalty, a support vector machine and a feed-forward neural network. Before doing this, we have standardised all features by removing the mean and scaling to unit variance with the `StandardScaler` from the `sklearn` package.

5.1 Regularized multinomial logit model (using the LASSO penalty) - logistic regression

We trained a multinomial logit model with the `LogisticRegressionCV` function of `sklearn`, using the `saga` solver. The advantage of this function is that it uses cross-validation (with `StratifiedKFold`) to find the best complexity penalty setting C for the LASSO penalty. After finding this, it trains the model on all training data with the best setting found. We have used 10 folds, and set the Cs feature to 100, meaning that a 100 C values are chosen in a logarithmic scale between 10^{-4} and 10^{4}.

The C setting we found that performed best with the cross-validation was 200.92. The accuracy on the train set was 94.36%. The accuracy on the test set was 89.82%.

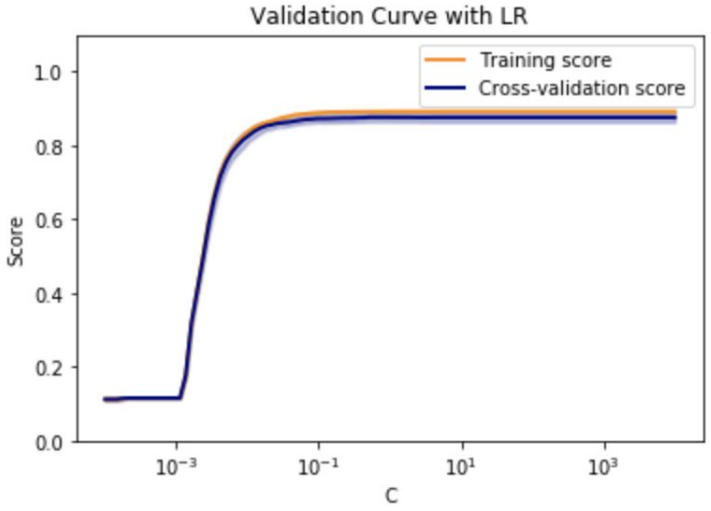


Figure 19. *Validation curve showing the training score and the cross-validation score for different values of C using a multinomial logit model with LASSO penalty.*

5.2 Support vector machine

We trained a support vector machine, using `SVC` with `rbf` (radial basis function) kernel from `sklearn`. Unfortunately, this model did not have a cross-validation tuning mechanism built in, therefore we used `GridSearchCV` from `sklearn` to find the best parameters for the support vector machine. We performed a grid search with 10 folds on the gamma and complexity penalty parameters of the SVC. We tested the following parameter settings:

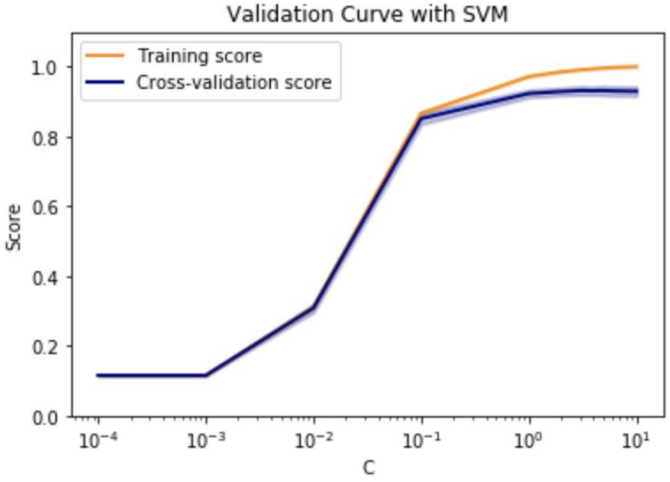
```
Cs = [0.0001, 0.001, 0.01, 0.1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
gammas = [0.0001, 0.001, 0.01, 0.1, 1]
```

The grid search found that a C of 3 and gamma of 0.001 performed best. We then trained the model again with these settings on all training data. The accuracy on the train set was 99.04%. The accuracy on the test set was 93.22%.

C	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1	2	3	4	5
gamma									
0.0001	0.1162	0.1162	0.1162	0.6124	0.8886	0.9030	0.9100	0.9152	0.9176
0.001	0.1162	0.1162	0.3092	0.8510	0.9228	0.9284	**0.9312**	0.9308	0.9312
0.01	0.1162	0.1162	0.1162	0.3130	0.7242	0.7410	0.7410	0.7410	0.7410
0.1	0.1162	0.1162	0.1162	0.1162	0.1778	0.1810	0.1810	0.1810	0.1810
1	0.1162	0.1162	0.1162	0.1162	0.1162	0.1162	0.1162	0.1162	0.1162

C	6	7	8	9	10	10^{2}	10^{3}	10^{4}
gamma								
0.0001	0.9186	0.9202	0.9206	0.9222	0.9228	0.9158	0.9116	0.9116
0.001	0.9302	0.9298	0.9296	0.9292	0.9294	0.9294	0.9294	0.9294
0.01	0.7410	0.7410	0.7410	0.7410	0.7410	0.7410	0.7410	0.7410
0.1	0.1810	0.1810	0.1810	0.1810	0.1810	0.1810	0.1810	0.1810
1	0.1162	0.1162	0.1162	0.1162	0.1162	0.1162	0.1162	0.1162

Table 3. *Cross-validation of support vector machine, showing the mean validation score (10 folds) for different values for gamma and C. The highest score is highlighted in green.*



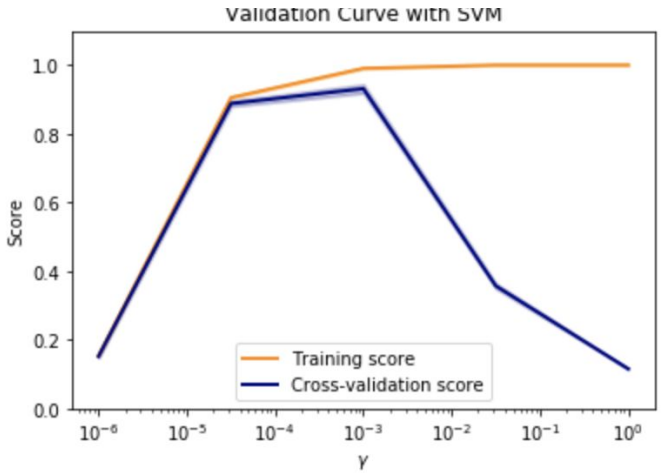


Figure 20 (left). *Validation curve showing the training and cross-validation score for different values of C using a support vector machine. Gamma is kept constant at 0.001.*
Figure 21. *Validation curve showing the training and cross-validation score for different values of gamma using a support vector machine. C is kept constant at 3.*

5.3 Feed-forward neural network

We trained a feed-forward neural network, using the `Sequential` model from the `keras` package. The model had the following characteristics: an input layer of shape 784 (all features), a hidden layer with an N number of neurons and a `ReLU` activation function and a L2 regularization with setting C. The output layer had a `softmax` activation function and a shape of 10 (the number of digits). The labels of the data were categorized/binarized using the `to_categorical` function of `keras`.

We used the same `GridSearchCV` function as for the SVM to find the best settings for the number of neurons and the regularization setting, again with 10 folds. We tested the following parameter settings:

```
neurons (N) = [10, 16, 20, 32, 64, 128, 256, 512, 1024]
regularization (C) = [0, 0.001, 0.002, 0.003, 0.004, 0.005, 0.006,
0.007, 0.008, 0.009, 0.01, 0.02, 0.03, 0.04, 0.05]
```

The grid search found that an N of 512 and C of 0.001 performed best. We then trained the model again with these settings on all training data, with a batch size of 128 and 20 epochs. The accuracy on the train set was 100%. The accuracy on the test set was 94.02%. Seeing these scores, we thought that the model was overfitting. However, these settings were obtained by using cross-validation. We later confirmed that increasing the C did not improve the accuracy on the test set. Higher accuracy could be reached by adding multiple hidden layers, dropout and other ways of improvement. However, we used a single-hidden-layer neural network, as was used by the other students using R.

6. Model comparison

We have compared the test accuracy of the three models we created in part 5 below:

Model	Test accuracy
Logistic regression	0.8982
Support vector machine	0.9322
Feed-forward neural network	0.9402

Table 4. *Comparison of test accuracies of the three models used.*

However, accuracy alone doesn't provide enough information to decide whether a model is better than another model. Therefore, we have used the McNemar's test statistic to determine whether the differences in accuracy were significant or not. We used the `mcnemar` function from the `statsmodels.stats.contingency_tables` package to calculate this. The McNemar's test statistic returns a P-value. We set the α to 0.01, therefore, if the P value is less than 0.01, we can conclude that the models perform significantly different.

6.1 Logistic regression vs. support vector machine

	SVM correct	SVM incorrect
LR correct	32579	656
LR incorrect	1914	1851

Table 5. *Contingency table for the logistic regression vs. support vector machine.*

Statistic=656.000, p-value=0.000
Different proportions of errors, significant difference between models.

When we look at the statistics above, we can conclude that there is a significant difference between the accuracies of the logistic regression (89.82%) and the support vector machine (93.22%), as the p-value 0.000 is less than α (0.01).

6.2 Logistic regression vs. feed-forward neural network

	NN correct	NN incorrect
LR correct	33006	229
LR incorrect	1785	1980

Table 6. *Contingency table for the logistic regression vs. feed-forward neural network.*

Statistic=229.000, p-value=0.000
Different proportions of errors, significant difference between models.

When we look at the statistics above, we can conclude that there is a significant difference between the accuracies of the logistic regression (89.82%) and the neural network (94.02%), as the p-value 0.000 is less than α (0.01).

6.3 Support vector machine vs. feed-forward neural network

	NN correct	**NN incorrect**
SVM correct	33944	549
SVM incorrect	847	1660

Table 7. *Contingency table for the support vector machine vs. feed-forward neural network.*

Statistic=549.000, p-value=0.000
Different proportions of errors, significant difference between models.

When we look at the statistics above, we can conclude that there is a significant difference between the accuracies of the support vector machine (93.22%) and the neural network (94.02%), as the p-value 0.000 is less than α (0.01).

In conclusion, we can say that the feed-forward neural network performs best on predicting new handwritten digits from the test set. Therefore, this is the best model we have created for this assignment.