Decision Making with Business Analytics - Assignment 1

Mike Weltevrede (ANR: 756479, SNR: 1257560)
Daniel Kolenbrander (ANR: 949231, SNR: 1273593)

October 1, 2019

1 Support Vector Machines

In this section, support vector machines $SVM_{(d_1,d_2)}$ for each digit pair (d_1, d_2) are created to classify every possible digit as either d_1 or d_2 (so also digits which are not d_1 or d_2 are being classified as one of the two). The SVMs are obtained using the radial kernel $K(u, v) = \exp\left(-\frac{\|u-v\|^2}{2\sigma^2}\right)$ and a grid search with $C \in \{10^{-3}, 10^{-2}, \ldots, 10^{3}\}$ and $\sigma \in \{10^{-8}, 10^{-7}, \ldots, 10^{2}\}$ with the `ksvm` function from the `kernlab` package in R. Via validation, the best SVM for each digit combination is selected as that SVM has the highest accuracy on the validation set (by construction made up of 25% of a sample from the training set). This first grid search used a random sample of 1,000 images from the training set. The parameters from this grid search are documented in Table A.1.

After this grid search, another grid search was run, using a random sample of 7,500 images from the training set. For this, the minimum and maximum values for C and σ found in the previous search were used and made the allowed parameters more granular. This lead to us using $C \in \{10^{-2}, 10^{-0.5}, \ldots, 10^{2}\}$ and $\sigma \in \{10^{-8}, 10^{-7.5}, \ldots, 10^{-6}\}$, with parameters as found in Table A.2. In the rest of this assignment, these SVMs with the corresponding results are used.

1.1 The digit 5

We would like to know which digit is the most and which digit is the least similar to the digit 5. For this, we consider the accuracy of the SVMs created for comparing 5 to other digits. Please consider Table 1.1 for the accuracies that we receive after predicting the test data with the SVMs created for digit 5 on the entire test set (not filtering on the digits specified for the SVM) and Table 1.2 for accuracies after filtering the test set (using the `confusionMatrix` function from the R package `caret`). By the filtered test set, we mean that for a certain $SVM_{5,d}$ c.q. $SVM_{d,5}$, we only take the test data with label 5 or d.

Other digit	0	1	2	3	4	6	7	8	9
Accuracy	0.1864	0.2025	0.1915	0.1885	0.1872	0.1838	0.1917	0.1857	0.1893

Table 1.1: Accuracy of SVMs on non-filtered test data

Other digit	0	1	2	3	4	6	7	8	9
Accuracy	0.9957	0.9990	0.9953	0.9911	0.9989	0.9935	0.9984	0.9952	0.9958

Table 1.2: Accuracy of SVMs on filtered test data

The reason why we have both of these accuracies is because we would like to know the effectiveness on the entire test set as well. However, this does not seem to make sense because the idea of creating separate SVMs is to, in the end, combine everything into the majority voting system, since individual SVMs cannot generalise outside of the two digits that they were trained on. As such, the explanations that follow will use the results from filtering the test data as per Table 1.2.

When the accuracy is high, this means that the corresponding digits are well distinguishable. The opposite holds for low accuracy. This means that the digit that is most similar to 5 is 3 with an accuracy of 99.11% and that the least similar digit is 1 with an accuracy of 99.90% (closely followed by 4, with an accuracy of 99.89%). However, note that all digits have an accuracy of over 99%, being extremely good.

1.2 The majority vote system

The majority vote system will look at the predictions of the 45 classifiers to predict one single digit d. Recall that our models $SVM_{(d_1,d_2)}$ will output whether the digit is d_1 or d_2, even if it is factually neither. Therefore, each model $SVM_{(d_1,d_2)}$ will "vote" for d_1 if d is most similar to that or d_2 otherwise. We will then add up the votes and our final prediction is the digit that received the most votes from these 45 classifiers. For this, we will use the test set.

The accuracy acc_{mvs} of the majority vote system can be easily computed by considering the number of correctly predicted digits $d_{correct}$ divided by the total number of digits predicted d_{total}, so

$$acc_{mvs} = \frac{d_{correct}}{d_{total}}.$$

For this instance, we find that the majority voting system correctly classifies **9,790** out of **10,000** digits, giving an accuracy of 97.90%.

1.3 The best and worst predictions

When evaluating each digit separately, we want to know which ones have the best and worst predictions. The results are found in Table 1.3.

Digit	0	1	2	3	4	5	6	7	8	9
MVS Accuracy	0.9908	0.9921	0.9777	0.9851	0.9796	0.9652	0.9812	0.9689	0.9805	0.9663

Table 1.3: Accuracy of Majority Voting System per digit

We see that the highest and lowest accuracy are achieved for the digits 1 and 5, respectively.

Apparently, the digit 5 is most difficult to classify, even though the model still boasts an impressive 96.52% accuracy for it. It is closely followed by the digits 9 (96.63%) and 7 (96.89%). A reason for this may be that, even though there are not many different ways in which one can write a 5 (while there are various ways to write a 7, for example by crossing it or making the top line wavy), if the lines are continued too long, it may resemble an 8, for example.

The digit 1 is easiest to classify, though it is closely followed by 0. This seems intuitive, since there are few ways in which one can write a 1; it is a relatively simple shape (a single line, possibly with a little bend at the top). Therefore, it is difficult to classify a digit that is truly a 1 as another digit.

1.4 Neural Network

We want to use the output of the majority voting system as an input to a neural network that will learn how to use this input to predict the digit. In essence, it decides some voting rule other than majority voting. Unfortunately, it is difficult to interpret a neural network's internal calculations so we do not know what this voting system actually looks like.

As an input to the neural network, we convert the output of the majority voting system to binary outputs. We will call this binary conversion $U \in \mathbb{R}^{45 \times N}$, with N being the amount of images in our dataset. For example, we currently have $SVM_{1,3}$ giving as output equal to 1 or 3. In U, this means that row 12 (corresponding to $SVM_{1,3}$) will have its 1s (the first class) converted to 0s and its 3s (the second class) to 1s. This is done for all 45 SVMs.

As output to the neural network (the labels which it needs to predict), we use the one-hot encoded versions of the labels, i.e. a binary matrix $V \in \mathbb{R}^{10 \times N}$ with a 1 in the spot corresponding to the relevant digit and 0 in all other spots.

This leads to an ANN with 45 input nodes and 10 output nodes. We place one hidden layer in between. This has H nodes, with $H \in \{5, 10, 15, 20\}$. For these values of H, we ran the neural network for 15 epochs with a batch size of 64 and looked at which value of H gave us the highest accuracy on the validation set. The results are found in Table 1.4.

H	5	10	15	20
Accuracy	0.9357	0.9852	0.9866	0.9877

Table 1.4: Accuracies for the ANN with H nodes in the hidden layer

From these results, we see that $H = 20$ has the highest validation accuracy but it is closely followed by $H = 15$. For good measure, we ran a grid search for $H \in \{15, 20\}$, $epoch \in \{11, 12, 13\}$, and $batch\_size \in \{64, 128\}$, of which the results are found in Table A.3. It follows that $H = 20$ makes up the top 3 in validation accuracies and that $batch\_size = 64$ makes up the top 5. The tie for number 1 with a validation accuracy of 98.76% between 11 and 13 epochs is broken by the one that has the lowest loss, which is 13 epochs. As such, our final model will have $H = 20$, $epoch = 13$, and $batch\_size = 64$.

2 Deep Learning - Auto encoders

2.1 Train an ANN

In order to train an ANN with the following typology we use the `keras` package in Python: an input layer of 784 nodes, 3 hidden layers of 200, 45 and 20 nodes respectively, and an output layer of 10 nodes, each representing a single digit. The network is created from several parts that are trained separately and then assembled together:

- An autoencoder that reduces from 784 nodes to 200 nodes
- An autoencoder that reduces 200 'basic' features to 45 'complex' features
- An ANN to reduce 45 features to 10 output nodes (akin to Section 1.4).

After running the code for this neural network (for 19 epochs, where we see overfitting after 16 epochs) we get impressive accuracies of about 99.91%, 99.48%, 99.49% on the training, validation, and test set, respectively.

2.2 Encoding features in the first hidden layer

Unfortunately, we did not manage to plot the weights corresponding to some of the 200 nodes in the first hidden layer in a 28 by 28 format. However, when plotting the weights of the first hidden layer, we would expect that the weights of the first hidden layer correspond to several more basic features, such as vertical and horizontal lines. As such, it should be able to recognise top parts of fives and the stems of nines,

2.3 Encoding features in the second hidden layer

Unfortunately, the same problem as in Section 2.2 arose, as a result of which we were unable to take a look at the 45 nodes in the second hidden layer. However, we expect this layer to detect some higher-level features such as the detection of lines in specific positions in the image, as well as small circles and loops, such as for sixes, eights, and nines.

3 Dimension Reduction

An image in our dataset is made up of 784 pixels in total. We will try to see the effect of dimension reduction on efficiency and accuracy. We will consider two methods of dimension reduction: Random Projection and Average Pooling (with a stride of 4).

3.1 Random Projection

By the Johnson-Lindenstrauss Lemma, we know that a set of points in a high-dimensional space can be embedded into a space of much lower dimension in such a way that distances between the points are nearly preserved[1]. Note here, that there is a probability linked to it; this result happens with probability at least 0.90, which is not perfect.

The accuracies are found in the same way as in Section 1.2. Using this strategy, we find the overall accuracies below in Table 3.1 and accuracies per digit as in Table A.5. Here, we see that the Random Projection does not perform well. Its overall accuracy is only 11.53%, with the highest accuracy is for recognising twos (73.55%) while the second highest accuracy is only 39.57% for identifying fives. All other accuracies are below 3%, with most even being 0%.

	Original Data	Random Projection	Average Pooling
Accuracy	0.9792	0.1153 (11.77%)	0.9193 (93.88%)

Table 3.1: Overall accuracies from the majority voting system for original versus dimension reduced data

To see whether the code is more efficient, we need to compute the difference in computation time of running the same code for the "original" data and the reduced data. The outputs are found in Table A.4. In this table, we see that the random projection leads to a run-time (including creating the randomly projected data from the original data) of 398.33 seconds while the original run-time is 824.35 seconds. This is a decrease of about 51.68%, which is quite a lot. Nonetheless, we say that it is not worth it given the immense drop in accuracy.

[1]Johnson–Lindenstrauss lemma. (2019, September 7). Retrieved September 28, 2019, from `https://en.wikipedia.org/wiki/Johnson\OT1\textendashLindenstrauss_lemma`.

3.2 Average Pooling

We will reduce an image from 28×28 pixels to a 7×7 pixeled image. We do this by averaging the pixel values in every 4×4 sub-square. For this method of dimension reduction, we see that the complete run-time, including creating the pooled data from the original data, is 176.68 seconds. Recall that the original data had a run-time of 824.35 seconds. This is a decrease of about 78.57%, which is a substantial gain.

Regarding the accuracies, the Average Pooling method works a lot better than the random projection. Even though its overall accuracy is 91.93% (compared to an original accuracy of 97.92%), we can see that the individual digits are recognised really well. The lowest accuracies achieved are 85.43% and 88.50% for identifying fives and eights, respectively.

4 Conclusions

4.1 Best performing predictor

When assessing the running times of the Majority Voting System on the original data, Random Projection and Average Pooling we see that Average Pooling is clearly the best predictor (see Table A.4). It showed a decrease of 78.57% in the running time compared to the original data and still recognized the individual digits very well with an overall accuracy of 91.93% compared to an original accuracy of 97.92%. However, since many of the applications for handwritten digit recognition (e.g. package delivery) have a very high costs associated with a wrongly delivered package the main focus could lay at the accuracy side. Therefore, when comparing the different predictors based on the accuracy we see that the Majority Voting System on the original data is the better predictor compared to Random Projection and Average Pooling, 97.92% versus 11.53% versus 91.93% respectively. However, compared to the ANN with three hidden layers the Majority Voting System is outperformed, since the ANN has an accuracy of 99.91%, 99.48%, 99.49% on the training, validation, and test set, respectively. Therefore the ANN is the best performing predictor based on our assumption that accuracy should be the most important aspect to take into account when deciding on which predictor to use.

Of course management should, always decide for themselves what weight they want to give the accuracy compared to the running time, since this differs based on the sector the company is in. For a company that handles same day deliveries, or maybe even within a few hours, the running time could be of more importance and management could choose a predictor based on the least amount of running time, instead of the highest accuracy.

4.2 Possible improvements

When looking at the Majority Vote System based on the SVM we could improve the predictor by removing the columns in the data that only have zero's. By deleting these columns we decrease the noise for the SVMs and they should therefore learn faster. Therefore, we expect the accuracy and efficiency of the Majority Voting System to increase when implementing this idea. Furthermore, we expect this change not to be too difficult since the removal of the columns that only have zeros is a relatively simple change.

A Tables

A.1 Support Vector Machines

		Second digit								
First digit		1	2	3	4	5	6	7	8	9
0	C	1	1	10	100	1	1	1	1	10
	σ	10^{-7}	10^{-7}	10^{-7}	10^{-8}	10^{-6}	10^{-7}	10^{-7}	10^{-6}	10^{-7}
1	C		10	0.1	1	1	1	10	10	1
	σ		10^{-8}	10^{-7}	10^{-7}	10^{-7}	10^{-7}	10^{-7}	10^{-7}	10^{-7}
2	C			10	10	1	1	10	1	10
	σ			10^{-7}	10^{-6}	10^{-7}	10^{-7}	10^{-7}	10^{-6}	10^{-8}
3	C				1	10	10	10	10	10
	σ				10^{-7}	10^{-7}	10^{-7}	10^{-7}	10^{-7}	10^{-7}
4	C					0.1	0.1	10	1	10
	σ					10^{-7}	10^{-7}	10^{-7}	10^{-7}	10^{-7}
5	C						1	0.1	1	10
	σ						10^{-6}	10^{-6}	10^{-6}	10^{-7}
6	C							0.1	1	0.1
	σ							10^{-7}	10^{-6}	10^{-7}
7	C								1	1
	σ								10^{-7}	10^{-6}
8	C									1
	σ									10^{-7}

Table A.1: Optimal values for $C \in \{10^{-3}, 10^{-2}, \ldots, 10^{3}\}$ and $\sigma \in \{10^{-8}, 10^{-7}, \ldots, 10^{2}\}$ according to a grid search

		Second digit								
First digit		1	2	3	4	5	6	7	8	9
0	C	$10^{-0.5}$	$10^{0.5}$	$10^{0.5}$	1	1	$10^{0.5}$	1	$10^{0.5}$	1
	σ	10^{-7}	$10^{-6.5}$	$10^{-6.5}$	$10^{-6.5}$	$10^{-6.5}$	$10^{-6.5}$	$10^{-6.5}$	$10^{-6.5}$	$10^{-6.5}$
1	C		1	1	1	$10^{1.5}$	$10^{-0.5}$	10	$10^{0.5}$	$10^{0.5}$
	σ		$10^{-6.5}$	$10^{-6.5}$	$10^{-6.5}$	10^{-7}	$10^{-6.5}$	$10^{-6.5}$	$10^{-6.5}$	$10^{-6.5}$
2	C			$10^{0.5}$	$10^{1.5}$	1	1	$10^{0.5}$	$10^{0.5}$	10
	σ			$10^{-6.5}$	$10^{-7.5}$	10^{-6}	$10^{-6.5}$	$10^{-6.5}$	$10^{-6.5}$	$10^{-6.5}$
3	C				1	10	1	$10^{1.5}$	$10^{0.5}$	$10^{0.5}$
	σ				10^{-6}	$10^{-6.5}$	$10^{-6.5}$	10^{-7}	$10^{-6.5}$	$10^{-6.5}$
4	C					1	$10^{0.5}$	$10^{0.5}$	$10^{0.5}$	10
	σ					10^{-6}	$10^{-6.5}$	10^{-6}	$10^{-6.5}$	$10^{-6.5}$
5	C						$10^{0.5}$	$10^{0.5}$	$10^{0.5}$	$10^{0.5}$
	σ						$10^{-6.5}$	$10^{-6.5}$	$10^{-6.5}$	10^{-7}
6	C							1	$10^{0.5}$	1
	σ							10^{-6}	$10^{-6.5}$	$10^{-6.5}$
7	C								1	$10^{0.5}$
	σ								$10^{-6.5}$	$10^{-6.5}$
8	C									$10^{0.5}$
	σ									$10^{-6.5}$

Table A.2: Optimal values for $C \in \{10^{-2}, 10^{-0.5}, \ldots, 10^{2}\}$ and $\sigma \in \{10^{-8}, 10^{-7.5}, \ldots, 10^{-6}\}$ according to a grid search

H	Epochs	Batch size	Validation accuracy	Validation loss	Test accuracy	Test loss
15	11	64	0.9871	0.0704	0.9765	0.1240
15	11	128	0.9830	0.0823	0.9728	0.1294
15	12	64	0.9867	0.0725	0.9765	0.1278
15	12	128	0.9852	0.0744	0.9741	0.1255
15	13	64	0.9870	0.0701	0.9769	0.1260
15	13	128	0.9858	0.0751	0.9752	0.1264
20	11	64	**0.9876**	0.0694	0.9769	0.1271
20	11	128	0.9854	0.0727	0.9763	0.1203
20	12	64	0.9872	0.0697	0.9774	0.1246
20	12	128	0.9859	0.0708	0.9749	0.1238
20	13	64	**0.9876**	0.0683	0.9769	0.1256
20	13	128	0.9868	0.0688	0.9765	0.1239

Table A.3: Accuracies following a grid search for $H \in \{15, 20\}$, *epoch* $\in \{11, 12, 13\}$, and *batch_size* $\in \{64, 128\}$

A.2 Dimension Reduction

Operation		Original Data	Random Projection	Average Pooling
Creating reduced data	Training data		12.39	33.65
	Test data		2.49	7.14
Creating SVMs		603.08	290.81	86.53
Majority Voting		221.27	92.64	49.36
Total time		824.35	398.33	176.68

Table A.4: Run times comparing original and pooled data (in seconds)

	Digits									
	0	1	2	3	4	5	6	7	8	9
Original Data	0.9918	0.9903	0.9826	0.9842	0.9817	0.9720	0.9812	0.9708	0.9723	0.9633
Random Projection	0.0000 (0.00%)	0.0000 (0.00%)	0.7355 (74.85%)	0.0297 (3.02%)	0.0000 (0.00%)	0.3957 (40.71%)	0.0000 (0.00%)	0.0049 (0.50%)	0.0000 (0.00%)	0.0059 (0.61%)
Average Pooling	0.9714 (97.94%)	0.9877 (99.74%)	0.9089 (92.50%)	0.9040 (91.85%)	0.9257 (94.30%)	0.8543 (87.89%)	0.9353 (95.32%)	0.9056 (93.28%)	0.8850 (91.02%)	0.9009 (93.52%)

Table A.5: Accuracies for individual digits from the majority voting system for original versus dimension reduced data, percentage of the original data accuracy in parentheses