Machine learning



**Group 40**

Name: Stefan Winter

Student number: 2067606

Name: Joost Oudesluijs

Student number: 2056392

Name: Joost Schutte

Student number: 2069625

[880083-M-6](https://tilburguniversity.instructure.com/courses/5509)

Dr. Güven Ç and Dr. Önal, I.

Department of Cogntive science and Artificial intelligence

School of Humanities and Ditital sciences

Tilburg University

June 2021

|  |
| --- |
|  |

# Task 1

**Model 1**

**Support vector machine**

Support vector machines is an algorithm that fits a hyperplane that classifies all datapoints to categories in an N-dimensional space (Ghandi, 2018). This hyperplane (black line) maximizes the distance or margin (light blue area) between different classes. The support vectors are the closest points to the classifier and determine the position of the hyperplane (see Figure 1).

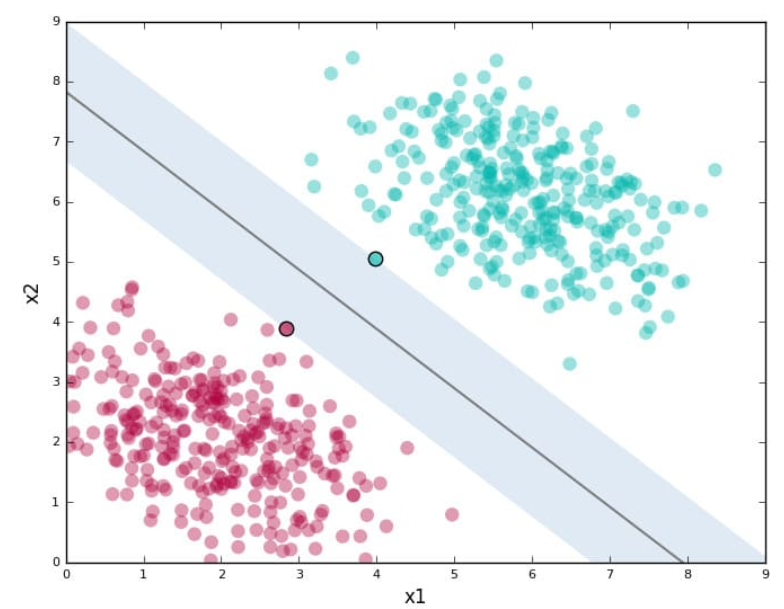


Figure 1: Visualized support vector machine.

To correctly classify the datapoints, a kernel function projects low-dimensional data into a higher-dimensional space. Enhancing the dimensions makes it possible to appropriately divide the observations into different classes. Therefore, support vector machines work more effective in high dimensional spaces, or datasets with a high number of input features (Ghandi, 2018). Since SVM can handle multiclass classification problems and works well with a high number of input features, this model is applied for task 1.

**Input to classifier**

To represent the image data from “train\_data\_label.npz” and “test\_data\_label.npz” the vectors are standardized with the StandardScaler function from sklearn. Hence the mean is subtracted from every observation and is divided by its standard deviation. Particularly for SVM, the algorithm assumes that all features are centered around zero and have variance in the same order (Scikit-learn, 2021). The data is not split into a training and a validation set, because we use cross-validation.

**Hyperparameter tuning**

To optimize this model a pipeline is created where multiple hyperparameter settings are evaluated. First, Principal Component Analysis (PCA) is applied to reduce the dimensionality and the noise (Husson et al., 2010). The n\_components function is applied from 75-100 percent with increments of 1 percent. E.g. 90 percent of all variance is explained with 58 features, which is a reduction of (784-58) 726 dimensions for model training. The cost parameter (C) is applied to find a good amount of samples allowed in the margin to find the lowest overall error (Scikit-learn, 2021). Values between 1-6 are applied with increments of 1. To transform low dimensional input to a higher dimensional space, multiple kernels are applied; polynomial, Radial Basis Function (RBF) and sigmoid. Lastly, to find the best parameter settings the GridsearchCV function from sklearn is applied with a cross-validation of five. This means that every part of the training data is used four times for training and one time for validation purposes (Souza, Matwin, Japkowicz, 2002).

**Model training**

The obtained training score after hyperparameter tuning is approximately 99 percent. The best settings are with a dimensionality of 91 percent variance explained. This reduces the 784 features to 65 features. The best parameter for C is 5. Lastly, the best SVM kernel is RBF. This makes sense since the RBF kernel and the Standardscalar both squeeze their observations to a (near) Gaussian distribution, therefore the RBF kernel could be the best fit (Scikit-learn, 2021).

**Results**

The created hyperplane from the training set, with the appropriate hyperparameter settings, gives an accuracy of 0.88 on the test set. This means that almost 90 percent of all test images are correctly classified based on the hyperplane from the trained model (see Appendix 2). The most difficult label to classify is the letter R. The letters V, Y, K, U and W were most often incorrectly classified as an R with 38, 35, 29, 27 and 21 times respectively. R incorrectly classified U and V most often, 35 and 21 times respectively. Therefore the main problem with R is with false positives, a precision problem.

**Model 2**

**Random forest**

Random Forest (RF) is built upon decision trees. Decision trees is a supervised learning model and is often used for classification tasks (James et al., 2013). The architecture of the decision tree is comparable with a regular tree; roots, nodes, branches and leaves. The root node is the starting point of the tree and splits every node with recursive partitioning based on the largest information gain (Sato & Tsukimoto, 2001). At each coming leaf node, the data splits into partitions based on the feature with the second largest information gain. This process is done until a base case is reached or when the label is small enough.

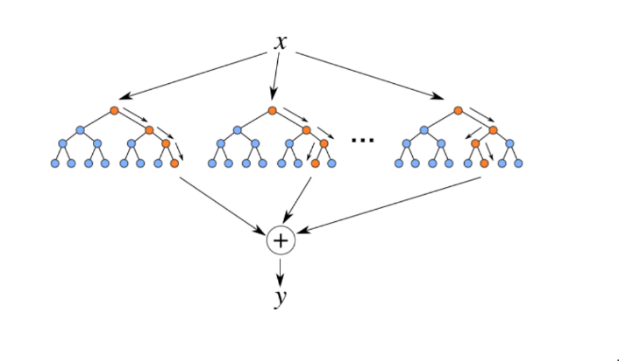
Although decision trees are transparent and easily interpretable, this model overfits the data quite often. To overcome this problem, RF is introduced. RF differentiate on decision trees by generating multiple trees. The process of generating multiple trees is done by random sampling with replacement of training data and random subsets of features when splitting nodes (Koehrsen, 2018). The random selection of features in each node allows variables to enter the model that otherwise would be overlooked (Strobl et al., 2008). The model is then trained with these random samples and subsets to decrease the total variance. However, different feature combinations could generate different predictions, therefore the final prediction is based on a majority vote.

Figure 2: Visualized random forest classifier.

**Hyperparameter tuning**

The input of the classifier is the same as from the former model, therefore it will not be discussed. To increase performance of the RF hyperparameters can be tuned in multiple ways. To set the number of decision trees that comprise the random forest the n\_estimator parameter is included in the pipeline. To prevent that every leaf is pure and prevent potential overfitting the max\_depth parameter is applied. To consider different splits in features the max\_features parameter is used. To set a penalty at the minimum number of samples used per leaf node the min\_samples\_leaf parameter is included. The parameter min\_samples\_split is also included to set a baseline for the minimum number of samples required to split an internal node. All hyper-parameters are set into a pipeline to find the best parameter settings from 1-20 with increments of 1 through the Gridsearchcv algorithm with a cross-validation of 5. PCA is included from 0.8-1.0 explained variance with increment of 0.01. Lastly, an untuned model was also executed.

**Model training**

The best settings from the RandomForestClassifier from Sklearn.tree was the untuned model. With a training set score of approximately 100 percent. The second best model is the tuned model, see Appendix 3 for the hyperparameter settings.

**Results**

The untuned model gives an accuracy of approximately 81 percent on the test set, which is 11 percent better than the best tuned model. Again, the most difficult label to classify is the letter R. The same letters are most often incorrectly classified for R as with the SVM model. However, the order only differs. Furthermore the letters S and W are also difficult to classify but still performs slightly better than the letter R (see Appendix 4). All these three letters have in common that they show higher false positives rates, therefore performing worse on precision compared to recall.

**Comparison between models**

The SVM model shows a better test performance compared to the RF model. The accuracy scores from both models show a difference of 8 percent in total. This difference is mainly caused by the letters S and W that perform approximately 20 percent worse on the RF model. Other letters stayed roughly the same; within 5 percent of each other. A remarkable finding is that both models seem to have problems to correctly classify the letter R. Furthermore, both models tend to show higher rates of false positives, and performs less on precision, on the letters with low test performance. A last comparison is runtime. The runtime to complete the trained model contains only contains 15 seconds for the RF model, while the SVM model needs four times as much. However, this difference is only a small absolute difference. Therefore, the SVM model is the overall best model.

# Task 2

The used model for task 2 is the SVM model. This model has 61 features (0.91 explained variance), uses a rbf kernel and scores 0.88 accuracy on the test set from task 1. To accurately predict the class labels from the test set from task 2, several preprocessing steps need to be done.

Firstly, the function “hand\_finder” is created to extract the top-left corner index of each hand picture per image. Therefore the Canny edge detection method is used. In this method every pixel is transformed into values of 255 or 0. The value of 255 is generated when the edges are predicted as noise and generated a 0 otherwise. When 28 values of 0 were detected, the first index of those 28 indexes were added to a list. Then the function iterates 28th indexes, and as long as the function does not find 28 values of 0, the function iterates 1 index further. This is done for every observation (image). This function returns a nested list of every left-corner index per observation. E.g. if there are five pictures in the first observation, the function extracts five left-corner indexes for that row in the list.

Secondly, the function “hand\_locator” is created to reshape the images back to a 28\*28 representation starting at the left-corner index for every picture. These index values were obtained from the first function (hand\_finder), and were transform back to their actual (initial) values. When plotting the new, denoised images, some pictures still contained some noise on the left or right sides of the pictures. The starting indexes were changed with a range of 5 (-3,2) to generate the appropriate images. This was done via trial and error.

Thirdly, the function “predict” is created to generate five predictions for each hand. Therefore if the first row contains five images, every image gets five predictions. However, if a prediction is correctly classified the function breaks the for loop. This break in the loop saves running time, therefore there is a higher efficiency.

Fourthly, the function “load\_true\_labels” is created to save the true label of each image to a list. These true labels were then compared to the predicted labels. If the true label was equal to one of the predicted labels, the counter for correct predictions increments by 1. The total correct predictions were than divided by the total length (amount of pictures) of the true labels.

The accuracy of the SVM model, with the aforementioned parameter settings, generated a score of 0.44 on the training set. This means that 11 images were correctly classified and 14 were not. The aforementioned trial and error approach for the function hand\_locator reduced noise into the picture. Therefore the accuracy enhanced to 60 percent. This means that 14 images were correctly classified and 11 images were not. Although this accuracy is still not as high as the accuracy from task 1, our model outperforms a simple coin toss. A simple coin toss would be a probability 0.042 for each image label times the power of the amount of each observation. Therefore this model could be enhanced with 28 percent points, but has a proper starting point already.

Lastly, the predictions for all 10,000 images from the test set were made and saved into a csv file. This process of predicting the images took approximately 5 minutes. To make a final comparison, the RF model, with the appropriate parameter settings, generated an accuracy of 60 percent as well. However, the running time took 6 times as long as the SVM model. Therefore, looking at efficiency the SVM model outperforms the RF model once again.

# Reference

Breiman, L. (2001). Random forests. Machine learning, 45, 5-32.

Gandhi, R. (2018b, July 5). Support Vector Machine — Introduction to Machine Learning Algorithms. Retrieved from https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

Koehrsen, W. (2018, December 1). An Implementation and Explanation of the Random Forest in Python. Retrieved from <https://towardsdatascience.com/an-implementation-and-> explanation-of-the-random-forest-in-python-77bf308a9b76

Sato, M. and H. Tsukimoto. 2001. Rule extraction from neural networks via decision tree induction. In Proceedings of the International Joint Conference on Neural Networks.

Strobl, C., Boulesteix, A. L., Kneib, T., Augustin, T., & Zeileis, A. (2008). Conditional variable importance for random forests. BMC bioinformatics, 9, 307.

Scikit-learn (2021). [sklearn.preprocessing](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing).StandardScaler. Retrieved from <https://scikit-learn.org/stable/modules/preprocessing.html>

Souza, J., Matwin, S., Japkowicz, N. (2002). *Evaluating Data Mining Models: A Pattern Language* School of Information Technology and Engineering University of Ottawa.

# Appendices

## 1 Group work

Stefan Winter

* Explaining Sourcetree
* Generating SVM and pipeline
* Feedback on description task 1 and task 2

Joost Oudesluijs

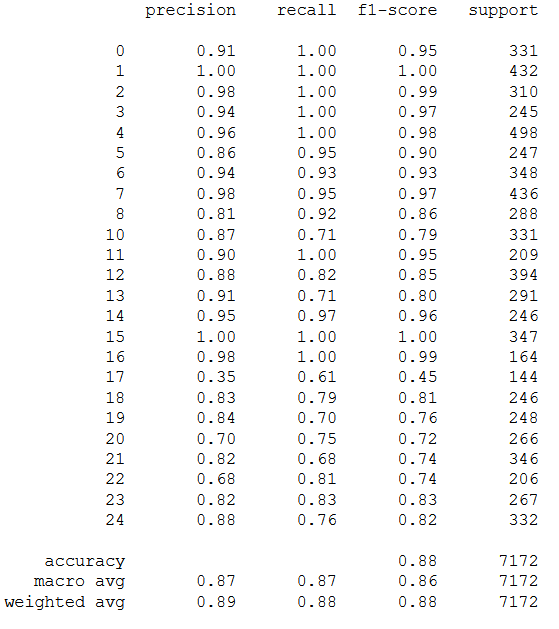
* Generating Neural network
* APA referencing
* Describing task 1 on paper
* Describing task 2 on paper

Joost Schutte

* Generating random forest and gridsearch
* Generating code for task 2
* Feedback on description task 1 and task 2

## 2 Classification rapport SVM

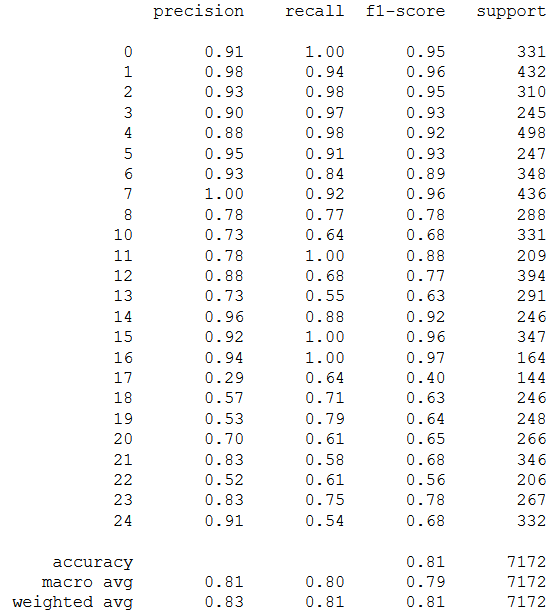
Table 1: Visualized classification report with the best hyperparameter settings.



## 3 Results tuned model RF

|  |  |  |
| --- | --- | --- |
| **Parameters** | **Tuned parameter scores** | **Test set score** |
| n\_components (PCA)  max\_depth  max\_features  min\_samples\_leaf  min\_samples\_split  n\_estimators | 0.85  12  5  1  2  19 | 0.70 |

## 4 Classification report RF



## 5 Images



Image 1: Images of example sign language letters and their labels