# ICS 3206

# Machine Learning, Expert Systems and Fuzzy Logic

Assignment report

Lara Brockdorff

# Introduction

This is the report of the assignment of ICS 3206 Machine Learning, Expert Systems and Fuzzy Logic, titled : Using Support Vector Machines to Classify Hand-Written Digits.

All assigned work has been completed

| Item | Completed |
|---|---|
| | |
| SVM technical discussion | YES |
| A good comparison to alternative methods | YES |
| Artifact | YES |
| Experimentation with different SVM params | YES |
| Experiments and their evaluation | YES |
| Overall Conclusions | YES |

The contents and files relating to this assignment can be found in the folder titled **Project.**

# Contents

# SVM technical discussion

SVMs are a form of supervised learning machines that are binary classifiers (but can be adapted to multiclass classifiers). This algorithm works by finding a hyperplane suitable for separating the data into the either of the classes, with n- dimensional space, where n refers to the number of features being trained on.
In this model the right threshold/hyperplane (cut of point between the classification of classes ) is found by using a cross validation technique in order to find the right margin between the points in the dataset. This is therefore a non probabilistic model.
This makes this model particularly useful for classification tasks as it is are equipped to handle outliers overlapping classifications from the consideration of mislabelled data when finishing the correct hyperplane.

Binary SVMs can be used as multiclass classifiers using a one vs rest approach

What makes SVMs particularly useful, is their adaption to use kernels. Kernels are transformation functions that can be used to map the training data onto another plane, with the intention of making the data easier to separate using the mentioned hyperplanes.
Most SVM implementations use what is called a trick-Kernel. This reduces computation time, as only the relationship[ every pain of pointes is calculated, instead of transforming the whole dataset.

Polynomial kernel works by taking the data up to the given polynomial degree.  Radial Base function(RBF) works by using a technique similar to the weighted nearest neighbour model where the closest observations to the one observed have the greatest influence on that point.

## Comparison to alternative methods

In order to better understand the effectiveness of SVMs, we can compare it to alternative methods.

The k-Nearest-Neighbour (kNN) model is also commonly used in classification, which works by searching for the closest matching samples in the dataset. The number n refers to the number of neighbours considered. This techniques accounts for outliers less than the SVM model, as the SVM model consist of a hyperplanes best separating the classes, rather than the values of the immediate neighbours. In research carried out in [1] the authors also mention how kNNs are less computational intensive than SVMs and should be considered for multiclass data for this reason.

Artificial neural networks (ANN) models that are built out of multiple layers of cells, each made up of a classifier(perceptron). Each layer contains its corresponding weights which are responsible for determine the class. ANN models tend to cover local minima, and so have a less general view of the dataset. If training of an ANN is prolonged, it is more likely to consider noise as part of the dataset, and so be more prone to overfitting. To prevent this larger datasets are required.

# Implementation / Artifact

## Data Set used

For this project the MNIST data set was used [2], imported using the openML mnist_784 library.

This is made up of 70 000 instances,  784 features ( 28 X 28 pixel images), and 10 classes (digits 0-9).

Due to technical restrains (processing power on available machines), the algorithms described were trained and validated on only 10 000 of the samples, since not enough computing power was available to run the whole dataset. Tests were run on another 2000 clean (unseen samples)

## Libraries used

The SVM implementation was imported from the **sklearn** python library. The **pandas** library was also made use of to store and process the imported data.

## Implementation details

### Test, validation and test

Due to the large size of the dataset mentioned above, and the limited computing paper available on the personal machines being used, a total of 12000 samples were used. These were split up as follows.

| Type/ Use | Number of Samples |
|---|---|
| Training | 8000 |
| Validation | 2000 |
| Test | 2000 |

The training set was used to trained to models to obtain a predictive model. The validation set was used to tune the hyperparameters of the model, while the Test set was left *unseen*  to the model, and was only used when calculating the final predictive scores.

Different proportions of the data used were tested. It was noted that when the proportion of the validation set was increased substantially the level of accuracy on the testing set decreased. This can be used due to overfitting.

## Implementation Approach

After importing and splitting the data as described, a simple SVM was trained using a Linear SVC, with default parameters.  Following that Polynomial, RBF and Sigmoid SVCs were also set up with default parameters, and their accuracy scores were computed. Since the Polynomial and RBF had the highest accuracy on the test data, the tuning of their hyper parameters was then worked on.

## Tuning of hyper parameters

In order to tune the hyper parameters a grid search was performed using the GridSearchCV call from the sklearn library. This works by searching through combinations of the given

hyperparameters, searching for the combination with the highest score. This was tested on the validation set, and the grid search was run a number of times, each time reducing the range of the parameters to be closer to the rages of the best estimator.
The following grids were used to find the best estimators for the RBF and Polynomial SVCs respectively.

```
# defining parameter range
param_grid = {'C': [1,2,3],
              'gamma': [0.0000002, 0.0000003, 0.0000004, 0.00000044],
              'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

grid.fit(X_train, y_train)

predicted_yNew =grid.predict(X_val)
print(accuracy_score(y_val, predicted_yNew))
print("****** ------- ********")
print(classification_report(y_val,predicted_yNew))

print(grid.best_estimator_)
```

*Figure 1Tuning RBF hyper parameters*

```
# defining parameter range
param_gridPoly = {'degree': [2,3,4,5],
             'kernel': ['poly']}

gridP = GridSearchCV(SVC(), param_gridPoly, refit = True, verbose = 3)

gridP.fit(X_train, y_train)

predicted_yP =gridP.predict(X_val)
print(accuracy_score(y_val, predicted_yP))
print("****** ------- ********")
print(classification_report(y_val,predicted_yP))

print(gridP.best_estimator_)
```

*Figure 2 Tuning Polynomial parameter*

After the best estimator was found, the predictions were made on the unseen test data, and the accuracy scores were computed.

## Evaluation

In order to evaluate the model we must interpret the precision, recall and f1 scores of the model run on the test data, using the chosen hyper parameters.

Precision refers to how many of the samples predicted to be positive (in that class), are actually positive. Therefore :True positives / (True positives + false positives).
Recall refers to how many of the samples that where actually labelled positive (in that class) were actually true positive. Therefore :True positives / (True positives + false negatives)
The F1 value is the multiplication of the precision and recall *2, divided by the sum of the precision and recall. This gives an indication of balance between the 2 values

While tuning the model with the polynomial kernel, the degree with the highest score on the validation data found in the grid search was with degree 2. The following results were obtained when running the classification report on the test (unseen) data.

Poly Trained results

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.97 | 0.99 | 0.98 | 207 |
| 1 | 1.00 | 1.00 | 1.00 | 230 |
| 2 | 0.95 | 0.94 | 0.94 | 198 |
| 3 | 1.00 | 0.97 | 0.98 | 207 |
| 4 | 0.97 | 0.98 | 0.98 | 194 |
| 5 | 0.96 | 0.95 | 0.96 | 169 |
| 6 | 0.96 | 0.99 | 0.97 | 202 |
| 7 | 0.98 | 0.97 | 0.97 | 215 |
| 8 | 0.95 | 0.95 | 0.95 | 187 |
| 9 | 0.98 | 0.95 | 0.97 | 191 |
| accuracy | | | 0.97 | 2000 |
| macro avg | 0.97 | 0.97 | 0.97 | 2000 |
| weighted avg | 0.97 | 0.97 | 0.97 | 2000 |

From these results we can note that the digits 1 and 3 were calculated most accurately. We can also note that the digit 3 had more false negatives than it had false positives, since the recall was lower than the precision. The digit 2 had the overall worse predictive score of 95% precision and 94% recall.

When tuning the hyper parameters of the RBF kernel, the parameters founds as the best estimator in the grid search were the following:

| C value | Gamma value |
|---|---|
| 2 | 4.4e-07 |

The following results were obtained when running the classification report on the test (unseen) data.

RBG Trained results

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.98 | 0.99 | 0.98 | 207 |
| 1 | 1.00 | 1.00 | 1.00 | 230 |
| 2 | 0.95 | 0.97 | 0.96 | 198 |
| 3 | 1.00 | 0.98 | 0.99 | 207 |
| 4 | 0.96 | 0.97 | 0.97 | 194 |
| 5 | 0.99 | 0.96 | 0.97 | 169 |
| 6 | 0.97 | 0.99 | 0.98 | 202 |
| 7 | 0.98 | 0.97 | 0.97 | 215 |
| 8 | 0.96 | 0.97 | 0.97 | 187 |
| 9 | 0.98 | 0.96 | 0.97 | 191 |
| accuracy | | | 0.98 | 2000 |
| macro avg | 0.98 | 0.98 | 0.98 | 2000 |
| weighted avg | 0.98 | 0.98 | 0.98 | 2000 |

We can note an increase in both the precision and recall of each digit, and also the total overall performance. Similarly to the previous case, the digits 1 and 3 had the best performance, while the digit 2, had the lowest performance.

It is also worth mentioning that the tasks being performed where computationally intensive, and some training was time consuming. This can be due to the way the kernel performs a transformation of the data, and also due to the process of fitting the hyperplane between the margins as was described about.

## Running the program

### Running the SVM.py

The file titled SVM.py can be found in the folder titled **Project.**
This can be run from the terminal using the command:

```
python SVM.py
```

### Jupyter Notebook

An ipython jupyter notebook is also present in the projects folder. This is run by running the `jupyter notebook` command and running the notebook that launches.

A pdf of the run notebook has also be added to the folder for reference.

## Overall conclusions

After understanding the algorithm behind SVMs, their use as a classifier can be greater apricated as one can understand how the use of the kernel to map the data into a more separable form proves to be very useful in a classification task.

The best performance achieved in this project was an average of 98% with a RBF kernel with c=2, and gamma= 4.4e-07.

## References


[1] S. M. Amita Goel, "Comparison: KNN & SVM Algorithm," International Journal for Research in Applied Science and Engineering Technology (IJRASET), vol. 5, no. 7, pp. 165-168, 2017.

[2] L. Y. a. C. C. a. B. CJ, "MNIST handwritten digit database," *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist.*