**Linnaeus University**
Introduction to Machine learning, 2DV516
*Jonas Nordqvist*
`jonas.nordqvist@lnu.se`

# Assignment 3: Supervised learning algorithms

**Document updated April 23rd**. Minor changes in Exercise 4.

## Introduction

In this Assignment you will use Python to handle a number of exercises related to various supervised learning methods: decision trees, support vectors machines and feed forward neural networks. The datasets used in the assignment can be downloaded in Moodle. The exercises are further divided into lectures.

## Submission instructions

All exercises are individual. We expect you to submit at least one py- file (or Jupyter Notebook) for each exercise and your submission should include all the datasets and files we need to run your programs. Execises A, B and C are just to get you started and should not be submitted. When grading your assignments we will in addition to functionality also take into account code quality. We expect well structured and efficient solutions. Finally, keep all your files in a single folder named as `username_A3` (e.g. ab123de_A3) and submit a zipped version of this folder.

Certain quantitive questions such as: *What is the MSE of the model?*, can simply be handled as a print statement in the program. More qualitative questions such as: *Motivate your choice of model.*, should be handled as a comment in the notebook or in a separate text file. (All such answers can be grouped into a single text-file.) The non-mandatory VG-exercise will require a separate report.

## 1 Support vector machines

### Exercise A: Testing the flexibility of SVM

Support vector machines may be very flexible, and may handle highly non-linear classification. In the following example you will utilize the `sklearn` classifier `svm.SVC` to build an SVM classifier for the data in the file `bm.csv`. This is a two-class problem and the data consists of 10,000 data points.

1. Create a dataset which consists of a random sample of 5,000 datapoints in `bm.csv`. To be able to compare with the subsequent results you can use the following code
   - `np.random.seed(7)`
   - `r = np.random.permutation(len(y))`
   - `X, y = X[r, :], y[r]`
   - `X_s, y_s = X[:5000, :], y[:5000]`
2. Use `sklearn` to create and train a support vector machine using a Gaussian kernel and compute its training error ($\gamma = .5$ and $C = 20$ should yield a training error of .0102, however note that these hyperparams are not optimized and the results may be improved).
3. Plot the decision boundary, the data and the support vectors in two plots, *c.f.* Figure 1. The indices of support vectors are obtained by `clf.support_`, where `clf` is your trained model.

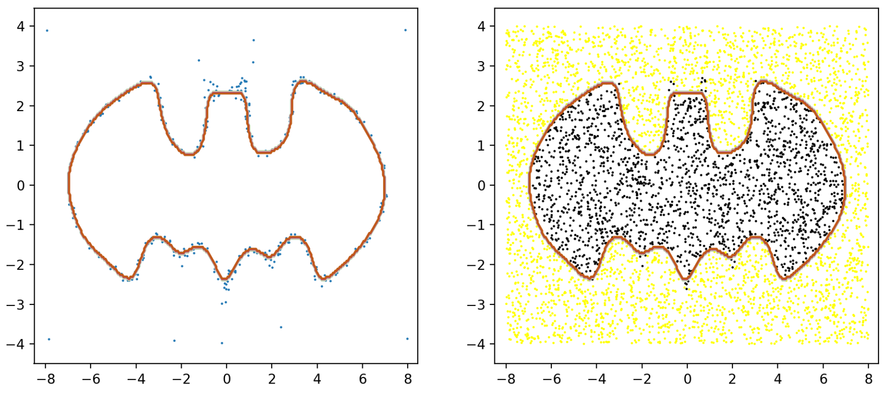

Figure 1: Support vector machine fitted on the BM dataset. To the left are the support vectors plotted together with the decision boundary and the to the right the data with the decision boundary.

## Exercise 1: Various kernels

Much of the flexibility in SVM lies in the *kernel*. In this exercise you will solve the same problem using three (or four) different kernels: linear, RBF, polynomial and a custom kernel (VG-exercise). The latter is of course not implemented in `sklearn`, and you will have to make your own implementation. These should later be compared in terms of predictive performance.

Use the dataset `mnistsub.csv`, which is a small, (and difficult) 2d dummy version of MNIST consisting only of examples from the digits `1, 3, 5` and `9`. Thus, each 2d coordinate represent a digit among the four classes. There are a total of 796 examples in the dataset. You should set aside a part of these for a validation set used in the hyperparameter search discussed below. Your task is to construct the three (or four) classifiers earlier mentioned. For each classifier you should do the following

1. Tune the necessary hyperparameters, by for instance grid search. In this exercise we are concerned with the hyperparameters given in Table 1. Every hyperparameter should be tested for at least 3 values but you are free to add more testings.[1]

| Kernel | Hyperparameters to tune |
|---|---|
| Linear | `C` |
| rbf | `C, gamma` |
| poly | `C, d, gamma` (optional) |
| (ANOVA) | `C, sigma, d` |

Table 1: Hyperparameters to tune in Exercise 1. Note that Anova is only necessary for students wanting a higher grade (*c.f.* Exercise 1.1)

2. Produce a plot of the decision boundary for the best models together with the data.

**Note:** If *not* using Jupyter notebooks all plots may ideally also be saved in a single pdf (preferred) or as images (wisely named) for the instructors as the reproduction of these plots may take some time to generate by code.

### Exercise 1.1 (VG-exercise): Implementing your own kernel

The VG-part of this exercise is to implement a kernel yourselves. In this exercise it will be executed by precomputing the kernel by means of a so-called *Gram matrix* explained further down in this text.

[1]On my Macbook the longest training time was for the ANOVA kernel and that was done in $\sim 6$ s. Hence, a grid search over $3^3$ different parameter values would amount to a total of less than 3 minutes. The others are done in negligible time, so it is easy to test for many values

The *ANOVA kernel* function can be computed as follows. Let $x, x'$ be two $\ell$-dimensional vectors, and denote by $x_i$ the $i$th component of $x$. Then we let

$$k(x, x') := \left( \sum_{j=1}^{\ell} \exp\left( -\sigma \left( x_j - x'_j \right)^2 \right) \right)^d ,$$

where $\sigma \in \mathbb{R}$ and $d \in \mathbb{N}$ are two hyperparameters to be tuned. In `sklearn` this kernel can be implemented as follows. Given two matrices $X$ and $Y$ of dimension $n \times p$ and $m \times p$. We compute the so-called Gram matrix $G$ of $X$ and $Y$, given by

$$G_{ij} = k(x^{(i)}, y^{(j)}),$$

where $x^{(i)}$ and $y^{(j)}$ are the $i$th and $j$th row of $X$ and $Y$. A correct implementation would thus yield a matrix $G$ of the size $n \times m$. To train a classifier you should instead of passing $X$ in the fitting pass the gram matrix of $X$ and $X$ and include `kernel='precomputed'` as an argument when constructing your classifier. When predicting you should instead pass the gram matrix of $X_p$ and $X$, where $X_p$ is the set of examples to be predicted.

**Bonus credit!** Prove that $k$ is a kernel. You may use any result without proof from the lecture.

### Exercise 2: One versus all MNIST

Support vector machines using rbf-kernels perform very well on the MNIST dataset. By tuning your parameters you should be able to get over 95% test accuracy. So, the first part of this exercise is to find `C` and `gamma` to obtain that kind of scores. You may use a smaller part of MNIST for training and still obtain good scores. Recall that the hyperparameters have to be found without laying your hands on the test set, *i.e.* use either cross-validation, a validation set or some other technique to distinguish between different models. Report in your code as comments, in your or in a separate document the grid (or whatever technique for hyperparameter search your are using) which was searched and the resulting best hyperparameters.

The second part of this exercise is to compare the built-in binarization scheme used for the `SVC` class, namely one-vs-one, against the one-vs-all scheme, which was discussed in Lecture 5. You should implement your own version of one-vs-all SVM and compare your results against the built in version. To make the comparison simple you should keep the same hyperparameters which you found in the first part of this exercise. Which was the best classifier? If studying the confusion matrix was there any apparent difference between the two methods in terms of misclassifications? Include your findings either as comments in your code, in your Jupyter notebook or as a separate text document.

The dataset is very common and easily obtainable online. Google is your friend here!

## 2 Decision trees

### Exercise B: Visualization

In this exercise we will use an artificial dataset with the inspiring and creative name `artificial.csv`. It is a 2d dataset which is very prone to overfitting.

1. Use `DecistionTreeClassifier` in `sklearn` and grow a decision tree and plot its decision boundary. It should be similar to the left boundary in Figure 2.

2. The boundary indicates that we have some overfitting at hand. We can solve this by restricting the depth of the tree. Pass the argument `max_depth=3` and refit the tree. This will instead grow a shallow tree. Visualize the decision boundary of the tree. It should look similar to the right boundary in Figure 2.

### Exercise 3: Counting Facebook comments

The dataset for the currect exercise is taken from the UCI Machine learning Repository.[2] It consists of little over 40,000 samples with more than 50 features. The target variable is the amount of Facebook

[2]`http://archive.ics.uci.edu/ml/datasets/Facebook+Comment+Volume+Dataset`

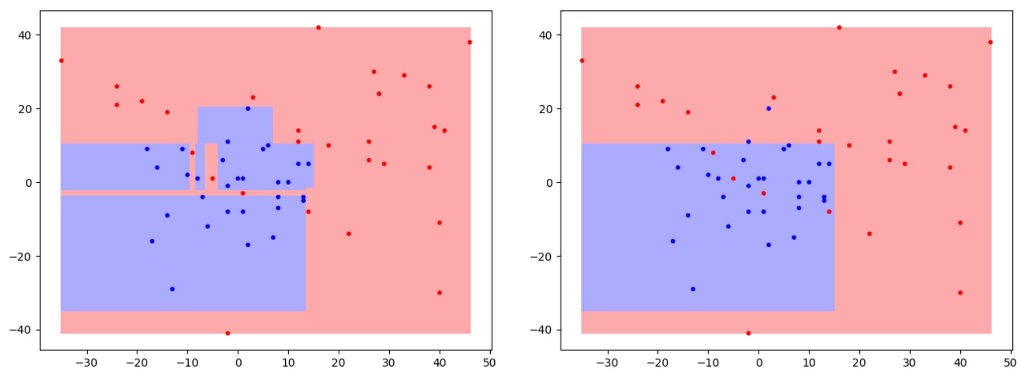

Figure 2: Decision boundary for the decision trees in Exercise B, deep tree to the left, shallow tree to the right

comments a post will generate in a $h$ hours, where $h$ is a feature in the dataset. The features are a variety of properties of the post and the page at which it is posted. You will be given both a training and a small test set found in the files `fbtrain.csv` and `fbtest.csv`.

1. Build a regression tree using `DecisionTreeRegressor`. What is the training MSE? What is the test MSE? Can this be improved by not allowing the tree to grow too deep in the training process?
2. Repeat 1) using `RandomForestRegressor`.
3. One problem is that the feature $h$ is almost always set to 24 in the training set, but rarely in the test set. Since this is feature is of great importance for the outcome restrict your $X$ and $y$ to those cases where $h = 24$ (this is the 39th feature in $X$). Repeat 1) and 2). Did your results improve?

## 3 Neural networks

### Exercise C: Yes or no but not both

Train a multilayer perceptron (MLP) to learn the XOR function, and print its parameters. Confirm for yourself that the network learned the function correctly by studying its weights.

### Exercise 5: ML in Fashion

Image classification is one of the fundamental tasks in machine learning. MNIST serves as a good starting point for testing out algorithms and learning. Some criticism that has been pointed out is that it is fairly easy to get good (well above 95%) accuracy. In this exercise you will perform image classification on a similar dataset known as Fashion MNIST. The dataset consists of 10 categories of different clothes, and your overall objective is to find a feed-forward neural network which can distinguish images on the different sets of clothes. The dataset contains 60,000 images for training and 10,000 for testing just as the ordinary MNIST. The images are $28 \times 28$ pixels.

The following elements should be included and printed/plotted in your code.

1. Plot 16 random samples from the training set with the corresponding labels.
2. Train a multilayer perceptron to achieve at least 82% test accuracy. There are numerous hyperparameters that we discussed in class which you can twerk, for instance: learning rate, number of and size of hidden layers, activation function and regularization (*e.g.* Ridge (known here as L2), and early stopping).
3. Plot the confusion matrix. Which are the easy/hard categories to classify? Are there any particular classes that often gets mixed together?