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

Assignment 3: Supervised learning algorithms

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 from 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, and B 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.

The following libraries are allowed to use in the assignment: `sklearn`[1], `pandas`, `matplotlib`, `scipy.stats`, `seaborn`, `numpy`, `tensorflow` and `pytorch` (the latter two only for the neural network exercise). However, note that we do not have time to discuss the latter two packages in class.

1 Neural networks

Exercise 1a: Fashion MNIST

Image classification is a classical task 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 clothing, and the 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 28×28 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 as good accuracy as you can. There are numerous hyperparameters that we discussed in class which you can tweak, 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). You should make a structured search for the best hyperparameters that you can find.

The data is available at *e.g.* `https://github.com/zalandoresearch/fashion-mnist`.

[1] Note that in some assignments `sklearn` is not allowed.

Exercise 1b: Fashion MNIST analysis and further extension (VG-exercise)

This exercise is optional for passing the assignment, but required to obtain grades A or B.

We will continue to work on the Fashion MNIST data (and our model) from Exercise 1a. Using the final model found in 1a you should go in and investigate the results, and carry the analysis further. The specifics of the exercise are listed in bullet form below. You should do the following:

- Plot the confusion matrix. Which are the easy/hard categories to classify? Are there any particular classes that often gets mixed together?
- Select two classes that exhibit some misclassifications between them, and make a visual inspection of the misclassifications (if there are too many a subset is sufficient) by means of plotting the misclassified examples. Is it reasonable that the model makes these misinterpretations? Are the examples difficult for human eyes? Comparing to the cases in which it correctly classifies are the difficult samples different? Write a short text describing your thought on the subject, and also include images as evidence supporting your viewpoints.

Exercise 2: Neural networks by hand (VG-exercise)

This exercise is optional for passing the assignment, but required to obtain grades A or B.

In this exercise you are only allowed to use `numpy`. Moreover, the assignment is to construct a small neural network 'by hand'. The input for the network are two features x_1 and x_2. The network should have a single hidden-layer with two nodes, and sigmoid activations, and the output layer should be a single node using a sigmoid activation. Bias terms should be included in both the input and the hidden layer.

Represent the weights as matrices, and bias vectors. $W^{(1)}$ and $b^{(1)}$ are the weights and biases going from the input to the hidden, and $W^{(2)}$ and $b^{(2)}$ comes from the hidden and goes to the output. In this exercise you'll have a Jupyter notebook-file (`nn_exercise2.ipynb`) the first part of the assignment is to 'fill in the blanks' in order to solve the problem. The problem at hand is the famous XOR-problem, where we want the network to learn the non-linear function that maps $(0, 0)$ and $(1, 1)$ to 0, and $(0, 1)$ and $(1, 0)$ to 1. In the notebook the backpropagation is already implemented but you need to provide the remainding code.

The second part of the exercise is to change the initialization of the weights in the exercise, by reducing them to 10% of their original value, and investigate what effects this has on the network. You should include your investigation as a short text.

Worth mentioning is that the loss function used in the exercise is the binary cross entropy previously discuss for logistic regression. However, for completeness we include it here again

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n} \left(y_i \log(f(\mathbf{x}_i)) + (1 - y_i) \log(1 - f(\mathbf{x}_i)) \right),$$

where f is the neural network, and $\theta := \{W^{(2)}, W^{(1)}, b^{(2)}, b^{(1)}\}$.

2 Ensemble methods and Decision Trees

Exercise A: Testing trees (Not to be submitted!!!)

In this exercise we will use an artificial dataset called `artificial.csv`.

1. Use `DecistionTreeClassifier` from sklearn to grow a decision tree with using `max_depth=3` in Python.
2. Implement your own method which plots the decision boundary for your decision tree model. Draw the decision boundary for your tree. The tree should look similar to the top image in Figure 1.
3. Using the `artificial.csv`-dataset we can in fact obtain zero training error for a decision tree. Fit a new tree for the data by removing the `max_depth`. This will grow a deeper tree. Visualize the decision boundary of the tree. They should look similar to the bottom boundaries in Figure 1.

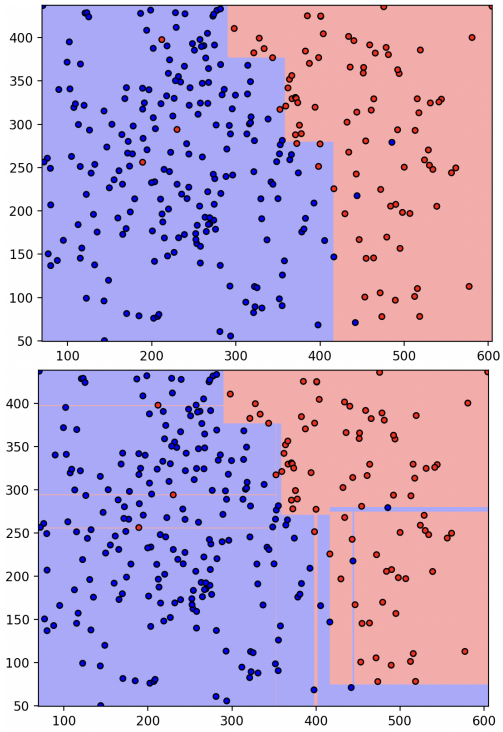

Figure 1: Decision boundary for the trees, shallow tree on top, deep tree in the bottom.

Exercise 3: Ensemble of Batman Trees

In this exercise, we'll revisit the Batman dataset from Exercise A. Start by splitting the data into a training of 9000 images and test set of 1000 examples each at random.

We will work with decision trees and the sklearn implementation of those.[2] In this assignment we will create our own simplified version of a random forest. We will construct 100 decision trees. Train these on different subsets for each classifier, and then combine their predictions in a majority vote to use as a classifier.

The subsets should be constructed using the bootstrap technique, which in short means that we should create new training sets for each individual decision tree in the following manner: sample 5000 training samples with replacement from the training data (that means any sample may be chosen more than once). This means that for instance X_1 is a subset of 5000 training samples, but most likely there are several which are duplicates. Thus, you will create $X_1, X_2, \ldots, X_{100}$ and the corresponding $y_1, y_2, \ldots y_{100}$. Then each of the 100 decision trees should be trained on the corresponding dataset. Some pointers on a suggestion on how to create these datasets is given in the code below.

```
# Assuming that X is the data matrix in the exercise, the following representation could be
    used for the 100 bootstrapped training data matrices for each decision tree

from numpy.random import default_rng
rng = np.random.default_rng()

n = 5000
r = np.zeros([n,100], dtype=int)
XX = np.zeros([n,2,100])

for i in range(100):
    r[:,i] = rng.choice(n, size = n, replace=True)
    XX[:,:,i] = X[r[:,i], :]
```

Listing 1: Ideas for generating bootstrap samples for the different models

For the sake of the assignment there is no need to work further with the different individual models in searching for hyperparameters. The default values are good enough!

The output of this exercise should be the following:

[2] The implementation of decision trees is the only place you should use sklearn for this exericse.

Figure 2: Decision boundaries for 99 decision trees on the Batman-dataset (the 100th model is removed for visual purposes) and also including the ensemble voting model (down-right corner)

a) The estimate of the generalization error using the test set of the ensemble of 100 decision trees.

b) The average estimated generalization error of the individual decision trees.

c) A plot of the decision boundaries of all the models, and including the ensemble model (*c.f* Figure 2).

d) Finally, a short comment on the results. Was it expected? Surprising? Do you see any benefits, downsides with this method?

3 Support vector machines

Exercise B: Testing the flexibility of SVM (Not to be submitted!!!)

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.

```
n_s = 5000
np.random.seed(7)
r = np.random.permutation(len(y))
X, y = X[r, :], y[r]
X_s, y_s = X[:n_s, :], y[:n_s]
```

Listing 2: Settings for the Batman dataset

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 3. The indices of support vectors are obtained by `clf.support_`, where `clf` is your trained model.

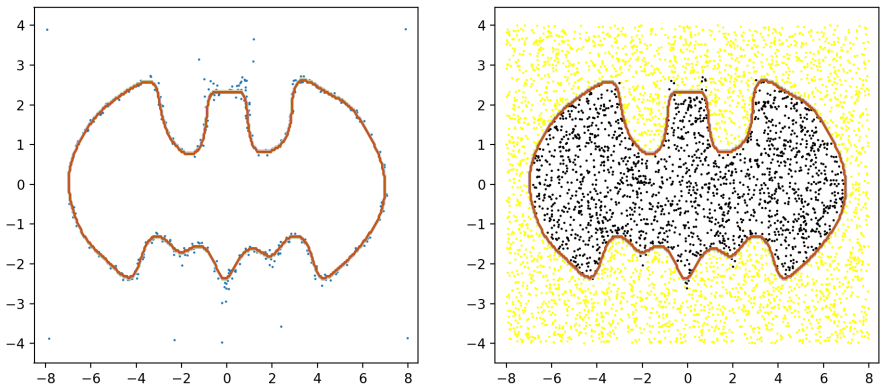

Figure 3: Support vector machine fitted on the BM dataset. To the left are the support vectors plotted together with the decision boundary and the to the right the data with the decision boundary.

Exercise 4: Various kernels

Much of the flexibility in SVM lies in the *kernel*. In this exercise you will solve the same problem using three different kernels: linear, RBF, and polynomial. These should later be compared in terms of predictive performance.

Use the dataset `dist.csv`, which is a small constructed toy set, it is visualized in Figure 4. The data is Gaussian mixture data, so there is a well-defined true decision bound. For $x \in [-6, 10]$, this is approximated by

$$d(x) = \begin{cases} 0.5\left(18 - 2x - \sqrt{-724 + 256x - 16x^2}\right), & \text{if } x > 3.94 \\ 0.071\left(174 - 22x - \sqrt{23123 - 6144x + 288x^2}\right), & \text{otherwise.} \end{cases}$$

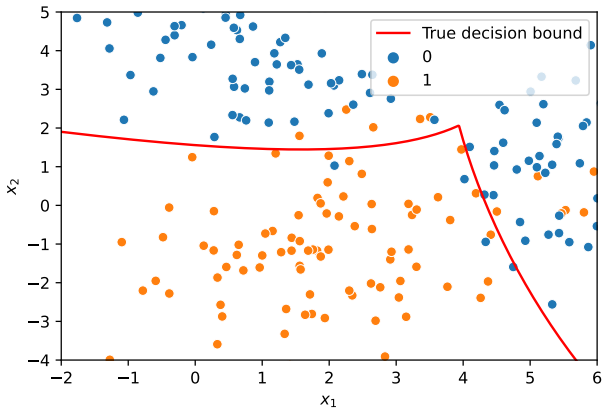

Figure 4: The dataset dist.csv and its true decision bound.

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. There is a desginated validation set that can be used for the validation of the hyperparameters `dist_val.csv`.
2. For each kernel, produce a plot of the decision boundary for the best models together with the data. If you want you can also include the true decision boundary as a comparison.

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.

Kernel	Hyperparameters to tune
Linear	`C`
rbf	`C, gamma`
poly	`C, d, gamma` (optional)

Table 1: Hyperparameters to tune in Exercise 4.

Exercise 5: Implicit versus explicit (VG-exercise)

This exercise is optional for passing the assignment, but required to obtain grades A or B.

In this exercise you should revisit the microchips (also available among the data files for this assignment). We saw that this problem can be well-described by a polynomial decision boundary. In this exercise you are going to solve this by means of a support vector classifier with explicit and implicit mapping of the features. In both cases you can use the standard hyperparameters except in the case of implicit mappping with polynomial kernel where you should set the degree as described in the exercise.

In the explicit model you should use `svm.SVC(kernel = 'linear')` and you should use the `mapFeature` function from Assignment 2 to get the various degrees of the data (explicitly). In the implicit model you should use `svm.SVC(kernel = 'poly', degree = d)`.

The objective of the exercise is to produce the content of the Table 2. The table consists of the training times for the two different models. The measuring of the training time should in the case of explicitly mapped features include the mapping of the features. Timing your training in Python may be done as describded in code below. The timing should be done at least 3 times per degree so that what you report is an average over these 3 (or more) runs.

```
import time

tic  = time.time()
# do what you want to do!
toc = time.time()
print(toc-tic)
```

Listing 3: A suggestion for time measurements in Python.

	Linear kernel (explicit)	Poly kernel (implicit)
Degree 2	·	·
Degree 8	·	·
Degree 15	·	·

Table 2: Average training time for a linear kernel SVC with explicit mapping of the features versus average training time for a SVC with polynomial kernel and thus implicit mapping of features.

Exercise 6: One versus all MNIST

Students taking 2DT916 need not to hand in this exercise.

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, 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!