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

Assignment 3: Decision trees and support vector machines

Introduction

In this Assignment you will use MATLAB to handle a number of exercises related to decision trees, maximal margin classifiers and support vector machines. The datasets used in the assignment can be downloaded in Moodle. The exercises are further divided into lectures. That is, for example, the first set of exercises related to decision trees are suitable to handle after Lecture 6.

Submission Instructions

All exercises are individual. Most exercises can be handled with a single .m-file. Certain quantitative questions such as: *What is the expected number of stories in a 900 ft building?*, can simply be handled as a print statement in the .m-file. More qualitative questions such as: *Motivate your choice of model.*, should be handled in a text file. (All such answers can be grouped into a single text-file)

Finally, keep all your m-, mat-, csv-, and text-files in a single folder named as `username_A3` and submit a zipped version of this folder.

Decision trees (Lecture 6)

Exercise 1: Visualization and cross-validation

In this exercise we will reuse data from assignment 1, the generated dataset `data1.mat`.

1. Use `fitctree` to grow a decision tree called `mdl` using the provided dataset with default arguments. Furthermore, use `view(mdl)` to view a text description of your tree. You should also let MATLAB visualize your tree as a graph. This can by done by also passing the named-valued pair `'Mode','graph'`.

2. Implement your own method which draws the decision boundary for your decision tree model. Draw the decision boundary for your grown tree. What is characteristic for the decision boundary of a tree? Conceptually, can you obtain any kind of decision boundary using decision trees?

3. To estimate the performance we can use k-fold cross-validation. Either build your own cross-validation implementation or check the documentation for cross validation on decision trees in MATLAB. Select an appropriate value for k and evaluate the models estimated classification error.

4. Using our dataset we can in fact obtain zero training error for a decision tree. Fit a new tree for the data, by also passing the argument `'MinParentSize', 1` (default is 10). This

will grow a deeper tree. Visualize the tree both as a graph, and its decision boundary. Test the k-fold cross validation error for the new deeper tree. Is the result expected? Motivate your answer.

Exercise 2: Depth control and pruning for Titanic data

In order to prevent overfitting of decision trees you can either impose a condition not allowing the tree to grow too deep, i.e. depth control, or you can build a full tree, which you later prune. In this exercise you will try and compare the results of both these approaches for the Titanic survival data.[1] The dataset contains only a training set and thus you must extract a part of it to dedicate as test set for comparing your approaches.

Besides the m-files you should also submit a short report on your work in this exercise, which should be **at most** 2 pages. In this report you should document your results, motivate your design decisions (data preprocessing etc.), and how you obtained them.

1. Some of the data points are missing some information you will have to choose what to do with these points. Also, note that some features might have to be excluded.

   I suggest that you import the csv-file using the import through the `Import data` in the Home tab in MATLAB to get a good visual of the problems with the data.

   Document and motivate in text what you did in the preprocessing and importing step of this exercise.

2. You should train and compare two models, one in which you apply depth control to make a smaller tree, and one where you use pruning.

   In both cases you should find the optimal models parameters `MaxNumSplits` for depth control and α for the pruning using some validation technique. In the Compendium there is a short text on how to obtain the pruned trees from a `fitctree`.

   Store the final models in a mat-file, and create a script which loads them and showcase their performance.

Support vector machines (Lecture 7)

Exercise 3: Maximal margin classifier

In this first exercise you will be given the code for the function `mmcPlot.m`. Missing in the code is the distance M, i.e. the perpendicular distance from the support vectors to the hyperplane. It is used to plot the slabs[2] and you should complete the code.

1. Complete the code in `mmcPlot.m` by computing the correct vaule for M.

2. Append to your data (X, y) the point `[-1 2]` which should be classified as -1. Rerun the `mmcPlot`. What happens?

Exercise 4: Classifying the BM-dataset

To showcase the flexibility of a support vector machine using RBF-kernels, we will train an SVM on the BM-dataset. The dataset is included in the zip-file for the exercise.

[1]Description and link to the source is found here `https://www.kaggle.com/c/titanic/data`

[2]Or the margin if you will.

The objective for this exercise is merely to fit an SVM using RBF-kernel for the dataset, find its training error-rate and plots its decision boundary.[3] You should use the MATLAB built-in functions for Support vector machines, and the training of such are available through `fitcsvm`. Note that you might have to tune the hyperparameters C, γ to obtain really nice boundaries. The following name-valued pairs can be passed to `fitcsvm` for this exercise

- `'KernelFunction', 'gaussian'`
- `'BoxConstraint'` this is the C parameter
- `'KernelScale'` this is the γ parameter.

Exercise 5: Hand-written digit recognition

Some of you have already given this problem a try in the previous assignment. This exercise concerns handwritten digit recognition using the MNIST dataset. This dataset contains 60,000 images of handwritten digits for training and 10,000 for testing. Each image is 28-by-28 pixels, and each pixel is represented by an integer 0-255. The data is found in `mnist.mat`, and it contains 2 cell matrices `img`, `img_test` containing the training and test data respectively, and the corresponding labels. The objective is to create a SVM-model which, with high accuracy, predicts the correct labels in the test set. Unfortunately, `fitcsvm` doesn't support multi-class classification but there is an alternative provided by MATLAB using `fitcecoc`. The syntax for `fitcecoc` is as follows

```
params = templateSVM('KernelFunction', K, 'KernelScale', γ, 'BoxConstraint', C);
mdl = fitcecoc(X,y,'Learners', params, 'Coding', 'onevsall');
```

Possible kernels K are `linear`, `polynomial`, and `gaussian`. For the polynomial kernel you can instead of `KernelScale` pass the argument `PolynomialOrder`.

Preprocess the data

There are a couple of preprocessing steps needed to be done before.

- Vectorize the input making it suitable for the `fitcecoc`. Hence, the objective is to obtain a 60000-by-784 (note that $28^2 = 784$) matrix where each row represent one image. The commands `cell2mat` and `reshape` are useful here.[4]
- Normalize to $[0, 1]$ interval.

Also, to visualize the images you can use `imagesc([img{n}])`, which extends to several images by, e.g. `imagesc([img{1} img{2}; img{3} img{4}])` or once you've obtained your data matrix `X` you can use `imshow` on an example which you reshape into a suitable size.

1. Train a linear SVM using a subset[5] of the training examples and evaluate its performance versus the test set. What is the accuracy of the model? Which of the digits does the method have most trouble predicting correctly?
2. At this stage you will try to find an optimal model for this problem. You are free to choose any kernel and any set of hyperparameters, however there should be some tuning involved illustrating why this particular set of hyperparameters were prefered over others.

[3]You can probably use your decision boundary-function from the Decision trees right away depending on how you implemented it.

[4]I suggest that you do not perform these manipulations using for-loops as it will take long time.

[5]This is due to the otherwise long training time.

Divide the training and test data in any way you want in order to obtain a (training, validation, test)-triplet. However, note that the test set must not be used until estimating the error of the final model.

Describe your effort in a report. Again, the report should be **at most** 2 pages. The output of this exercise should (at least) be the accuracy of the model, and the confusion matrix of the test set, together with explanations of the results, and some illustration of misclassified examples. Also, store the output model file in a .mat-file which you include in your submission, and create a script which loads this file and showcase its performance (without having to retrain the model).

On average humans perform at an accuracy of 98.84 % on classification of this data. Hence, to perform better than that is said to be superhuman. Does your model reach superhuman level? Do you have any suggestions on how to improve your model?

On a final note: training the full model may take some time, why it is perfectly fine to only do things on smaller sets of this data, as long as you write this in your documentation.