Assignment 5 – CV1 WS14/15 26/01/2015

Prof. Stefan Roth
Jochen Gast
Stephan Richter

This assignment is due on February 2nd, 2015 at 13:00.

Please refer to the previous assignments for general instructions and follow the handin process from there.

Problem 1 - Linear SVMs (15 points)

From class we know that formulating support vector machines (SVMs) involves implementing and solving a constrained quadratic optimization problem. In this problem you will implement linear SVMs in the primal form. Conveniently, Matlab has a built-in solver which does essentially all of the work for you: `quadprog` solves arbitrary constrained quadratic optimization problems of the type

$$\begin{aligned} \arg\min_{\mathbf{z}} \quad & \frac{1}{2}\mathbf{z}^{\mathrm{T}}\mathbf{H}\mathbf{z} + \mathbf{f}^{\mathrm{T}}\mathbf{z} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{z} \leq \mathbf{b}. \end{aligned}$$

You will have to convert the SVM formulation below into the generic input format for the `quadprog` function, and then turn its output into a linear discriminant function. The outline for the problem is given in `problem1.m` and should be completed with the necessary function calls.

Task 1: Training a SVM for linearly separable training data

- In the `data` folder you will find a training data set called `separable.mat` containing 2D data points as well as their corresponding class labels. Implement the function `load_data.m` that loads the data and converts abstract class labels to numeric labels. Make sure that abstract labels "class_1/2" correspond to labels 0/1, respectively.
- To gather a little insight about the data structure, implement the function `show_before.m` which shows the 2D data points as a scatter plot. Please make sure that points from different classes have different colors and also include a `legend` indicating which classes are represented by the colors.
- Now train a support vector machine from the training data by solving the following quadratic optimization problem in `train_svm_separable.m`:

$$\begin{aligned} \arg\min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{s.t.} \quad & y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1, \quad \forall i. \end{aligned}$$

  To do this, you have to find out how to turn this constrained optimization problem into the generic form from above. Things you should ask yourself are: Which variable(s) does $\mathbf{z}$ need to represent? And what do you need to set $\mathbf{H}, \mathbf{f}, \mathbf{A}$, and $\mathbf{b}$ to?
  As a result you should return the learned weight vector $\mathbf{w}$ and the bias b. Since SVMs are sparse learning machines you will find that only a very few of your data points actually contribute to the solution. To find out these so-called support vectors you have to inspect the Lagrange multipliers returned by `quadprog`. Having done that you should also return the indices of the found support vectors, i.e. the columns of the support vectors within your feature matrix. Report the learned values of the weight vector $\mathbf{w}$ and bias b on the command window.

- Plot the decision boundary $y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0$ as well as the margin lines (see illustrations in the lecture slides) on top of a scatter plot of the data in `show_after.m`. Note that the margin lines are also contour lines $y(\mathbf{x}) = c$. Which values of c do you need? Also mark the support vectors in the plot (e.g. draw additional circles around the support vectors with a slightly larger radius than the other data points). Include a `legend` that indicates points with corresponding class label, the decision boundary, the margins and the support vectors.

Task 2: Training a SVM for non-separable training data

We now train a SVM for training data which is not linearly separable. From class we know that this can be achieved by including *slack variables* in the SVM formulation.

- Use your previously implemented function `load_data.m` to load the non-separable training data given in `nonseparable.mat`. Similarly, make use of your function `show_before.m` to visualize the non-separable training data. Although the data looks quite similar to the one given in the previous task, there is a crucial difference: you are not able to choose a hyperplane that perfectly separates points from both classes.

- Solve the following quadratic optimization problem in `train_svm_nonseparable.m` which now contains slack variables:

$$
\begin{array}{lll}
\underset{\mathbf{w},\xi,b}{\arg\min} & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i & \\
\text{s.t.} & y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1 - \xi_i, & \forall i, \\
 & \xi_i \geq 0, & \forall i.
\end{array}
$$

  Again, ask yourself what the various terms in the generic formulation need to represent and train a SVM from the non-separable training data (you may notice that the learning is not very sensitive for various choices of C). Please report the weight vector and bias value on the command window for $C = 5$.

- Finally, use `show_after.m` to visualize the learned decision boundary, margin lines and support vectors on top of the data.

Task 3: Categorize images with linear SVMs

Remember the categorization problem from assignment 4? Instead of using a Naive Bayes classifier we can instead use a SVM. Here each image is represented by a distribution over the trained codeword dictionary, which could be estimated by normalizing the number $n(i,m)$ of times a keypoint μ_i occurs in image I_m. We can then categorize images by separating these distributions:

- We included the inputs and histograms from assignment 4 in `asgn4data.mat`. Implement `load_asgn4.m` where you should both load the image labels and compute the feature vectors for training and testing sets, respectively. Here features are given by a probability distribution over the codeword dictionary, which you obtain by normalizing the corresponding histograms.

- Choose a weight C to train a non-separable SVM from the training data. Then implement `predict_svm.m` which computes your final predictions by means of the learned discriminant function. Finally, show the prediction error for the training and testing set on the command window. Try various weights for C and keep the one you find has the best overall performance.