HW5. Support Vector Machine

Jongmin Lee (jmlee@ai.kaist.ac.kr)

May 11, 2016

1 Introduction

In this assignment, you will design *support vector machine* on two image data sets which are same as before (i.e. handwritten digit images and face images). Don't be afraid. This assignment is super easy!

2 NumPy and SciPy

In this assignment, we will use NumPy(`http://www.numpy.org/`) and SciPy(`https://www.scipy.org/`) as before. Don't forget to use **Python 2.7** when scripting your code.

3 Project Instruction

3.1 Support Vector Machine (SVM)

3.1.1 Maximum Margin Classifiers

Let's start with the two-class classification problem using linear model when the training data set is linearly separable.

$$y(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

The training data consists of N input vectors $\mathbf{x}_1, ..., \mathbf{x}_N$ and corresponding target values $t_1, ..., t_N$ where $t_n \in \{-1, 1\}$, and new data points $\mathbf{x}$ are classified according to the sign of $y(\mathbf{x})$. Assume that training data set is linearly separable and there exists $\mathbf{w}$ and b that satisfy

$$\begin{cases} y(\mathbf{x}_n) > 0 & \text{if } t_n = +1 \\ y(\mathbf{x}_n) < 0 & \text{if } t_n = -1 \end{cases}$$

for every training instance, so that $t_n y(\mathbf{x}_n) > 0$ for all training data.

As usual, our goal is to find such hyperplane which separates the data with different classes. However, there may exist infinitely many solutions that separate the classes exactly, and if there are multiple solutions all of which classify the training data set exactly, we should try to find the one that gives the smallest generalization error. The support vector machine approaches this problem through the concept of the *margin*, which is defined to be the smallest distance between the decision boundary and any of the samples, as illustrated in Figure 1.

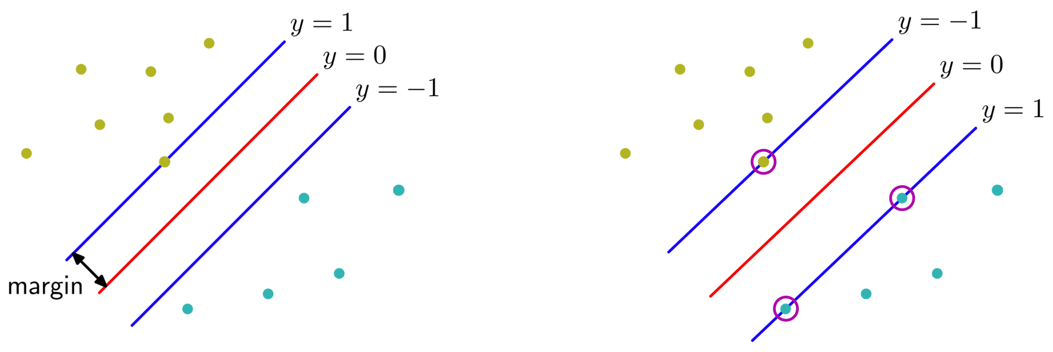

Figure 1: The margin is defined as the perpendicular distance between the decision boundary and the closest of the data points, as shown on the left figure. The location of the boundary is determined by a subset of the data points, known as support vectors, which are indicated by the circles on the right figure.

3.1.2 SVM Formulation: Lagrangian Primal and Dual

Recall that the perpendicular distance of a point $\mathbf{x}$ from a hyperplane defined by $y(\mathbf{x}) = 0$ where $y(\mathbf{x}) = \mathbf{w}^\top\mathbf{x} + b$ is given by $|y(\mathbf{x})|/\|\mathbf{w}\| = |\mathbf{w}^\top\mathbf{x} + b|/\|\mathbf{w}\|$. As mentioned, our goal is to maximize margin and the solution is given by:

$$\begin{aligned}\arg\max_{\mathbf{w},b}\,(margin) &= \arg\max_{\mathbf{w},b}\left(\min_n \frac{|\mathbf{w}^\top\mathbf{x}_n + b|}{\|\mathbf{w}\|}\right)\\ &= \arg\max_{\mathbf{w},b}\frac{1}{\|\mathbf{w}\|}\min_n\left[t_n(\mathbf{w}^\top\mathbf{x}_n + b)\right]\end{aligned}$$

Here, note that the scale of $\mathbf{w}$ and b doesn't change the value of margin, which enables us to set:

$$t_n(\mathbf{w}^\top\mathbf{x}_n + b) = 1$$

for the point that is closest to the surface and then all data points will satisfy the constraints:

$$t_n(\mathbf{w}^\top\mathbf{x}_n + b) \geq 1, \quad n = 1, ..., N \tag{1}$$

The optimization problem then simply requires to maximize $1/\|\mathbf{w}\|$ subject to the constraints given by Eq (1), which is equivalent to minimizing $\frac{1}{2}\|\mathbf{w}\|^2$.

$$\boxed{\begin{array}{ll}\underset{\mathbf{w}}{\text{minimize}} & \frac{1}{2}\|\mathbf{w}\|^2\\ \text{subject to} & t_n(\mathbf{w}^\top\mathbf{x}_n + b) \geq 1,\ n = 1, \ldots, N.\end{array}}$$

In order to solve this constrained optimization problem, we introduce Lagrange multipliers $a_n \geq 0$, with one multiplier a_n for each of constraints in (1), giving the following Lagrangian:

$$\mathcal{L}(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n\{t_n(\mathbf{w}^\top\mathbf{x}_n + b) - 1\}$$

where $\mathbf{a} = [a_1, ..., a_N]^\top$. By setting the derivatives of $\mathcal{L}$ with respect to $\mathbf{w}$ and b equal to zero, we obtain the following two conditions:

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \mathbf{x}_n \tag{2}$$

$$0 = \sum_{n=1}^{N} a_n t_n \tag{3}$$

Eliminating $\mathbf{w}$ and b from $\mathcal{L}$ using these conditions gives the *dual representation* of the maximum margin problem:

$$\boxed{\begin{array}{ll} \underset{\mathbf{a}}{\text{maximize}} & \tilde{\mathcal{L}}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m \mathbf{x}_n^\top \mathbf{x}_m \\ \text{subject to} & a_n \geq 0,\ n = 1, \dots, N. \\ & \sum_{n=1}^{N} a_n t_n = 0 \end{array}}$$

3.1.3 Kernel SVM and its Prediction

Thanks to dual representation of maximum margin problem, we can adopt *kernel trick* that avoids explicit feature mapping $\phi(\mathbf{x})$. What we should do is just to replace $\mathbf{x}_n^\top \mathbf{x}_m$ with $\phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m)$. Through this, we can deal with implicit high-dimensional feature mapping. The resultant optimization problem is simply given by:

$$\boxed{\begin{array}{ll} \underset{\mathbf{a}}{\text{maximize}} & \tilde{\mathcal{L}}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \\ \text{subject to} & a_n \geq 0,\ n = 1, \dots, N. \\ & \sum_{n=1}^{N} a_n t_n = 0 \end{array}}$$

For this assignment, we will use radial-basis function(RBF) kernel defined as:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

,where σ is free parameter. Theoretically, it is known that the feature space of RBF kernel has an infinite number of dimensions.

In order to classify new data points using the trained model $(a_1, ..., a_N)$, we should express $y(\mathbf{x})$ with respect to $\mathbf{a}$ and kernel function. This can be achieved by substituting $\mathbf{w}$ using Eq (2).

$$\begin{aligned} y(\mathbf{x}) &= \mathbf{w}^\top \mathbf{x} + b \\ &= \sum_{n=1}^{N} a_n t_n \mathbf{x}_n^\top \mathbf{x} + b \\ &= \sum_{n=1}^{N} a_n t_n k(\mathbf{x}_n, \mathbf{x}) + b \end{aligned} \tag{4}$$

Note that many data points may have $a_n = 0$, in which case they do not contribute to the predictive model Eq (4). Only the remaining points whose $a_n > 0$ contribute to the prediction and constitute the support vectors.

3.1.4 Soft-Margin SVM

So far, we only considered the case that training data points are linearly separable. Although SVM can give exact separation with non-linear decision boundaries using appropriate kernel function $k(\mathbf{x}, \mathbf{x}')$, this usually leads to poor generalization. We therefore modify SVM so as to allow some misclassification. To do this, we introduce *slack variables* $\xi_n \geq 0$ where $n = 1, \dots, N$, with one slack variable for each training data point.

These are defined by $\xi_n = 0$ for data points that are inside the correct margin boundary, and $\xi_n = |t_n - y(\mathbf{x}_n)| > 0$ for other points. For example, a data point that is on the decision boundary ($y(\mathbf{x}) = 0$) will have $\xi_n = 1$, and the misclassified points will have $\xi_n > 1$. Then, the exact classification constraints Eq (1) are relaxed to:

$$t_n(\mathbf{w}^\top \mathbf{x}_n + b) \geq 1 - \xi_n, \quad n = 1, ..., N \tag{5}$$

, and at the same time, we penalize the points that lies on the wrong side of margin boundary. These gives the following modified optimization problem:

$$\boxed{\begin{array}{lll} \underset{\mathbf{w},\boldsymbol{\xi}}{\text{minimize}} & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n & \\ \text{subject to} & t_n(\mathbf{w}^\top \mathbf{x}_n + b) \geq 1 - \xi_n, & n = 1, \dots, N \\ & \xi_n \geq 0, & n = 1, \dots, N \end{array}}$$

where the parameter $C > 0$ controls the trade-off between the slack variable penalty and the margin. Like before, we can construct Lagrangian and get its dual form by eliminating $\mathbf{w}$, b and $\{\xi_n\}$. Finally, we obtain the following optimization problem:

$$\boxed{\begin{array}{ll} \underset{\mathbf{a}}{\text{maximize}} & \tilde{\mathcal{L}}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \\ \text{subject to} & 0 \leq a_n \leq C, \; n = 1, \dots, N. \\ & \sum_{n=1}^{N} a_n t_n = 0 \end{array}}$$

, which is identical to separable case except that the first constraints are somewhat different. Predictions for new data points are again made by using Eq (4).

3.2 What to Do

To simplify the problem, we serve many parts of the whole algorithm as skeleton code. First, we tackle the multi-class classification task through one-vs.-one reduction, which trains $C(C-1)/2$ binary clasifiers for a C-way multiclass problem. This is already implemented so you don't have to care about the details of one-vs.-one reduction and only focus on the implementation of two-class SVM. Second, we serve the quadratic programming solver **(self.quadraticProgrammingSolver)** which solves the following form:

$$\boxed{\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \frac{1}{2}\mathbf{x}^\top \mathbf{P}\mathbf{x} + \mathbf{q}^\top \mathbf{x} \\ \text{subject to} & \mathbf{G}\mathbf{x} \leq \mathbf{h} \\ & \mathbf{A}\mathbf{x} = \mathbf{b} \end{array}}$$

Third, maintaining support vectors only and discarding the others after the training, which is for efficient prediction, is already implemented. Your prediction will be performed based only on support vectors and the corresponding ones. Lastly, the computation of bias term is also served.
You will fill in portions of **svm.py**. Specifically, you have to code the following functions:

- *trainSVM*
- *predictSVM*

In *trainSVM* function, you should solve the dual form of soft-margin SVM described in Chapter 3.1.4 and return obtained Lagrange multipliers $\{a_i\}$. Note that the labels of training input are binary ($t_n \in \{-1, +1\}$), and also the quadratic programming solver is given. What you should do is only writing the quadratic programming problem description, call the QP solver and return the output values. That's it! In *predictSVM*, you should return prediction results based on bias term b, support vectors $\{\mathbf{x}_n\}$ and their labels $\{t_n\}$, and the corresponding Lagrange multipliers $\{a_n\}$. The comments of the methods describe the data structures of the input and output.

- We are using dimensionality-reduced real-valued feature vectors obtained by PCA as input parameters.
- Do **NOT** use SVM function which are implemented in other scientific packages(i.e. scikit-learn) directly.

3.3 What to Submit

Please submit **svm.py** file only. Any late submissions will not be accepted.

3.4 How to Run the Code

To try out the classification pipeline, run **dataClassifier.py** from the command line. This will classify the digit data using the default classifier (mostFrequent) which blindly classifies every example with the most frequent label.

```
python dataClassifier.py
```

To activate the Support Vector Machine, use -c SVM:

```
python dataClassifier.py -c SVM
```

To run on the face recognition dataset with different training data size, use -d faces and -t numTraining

```
python dataClassifier.py -c SVM -d faces -t 300
```

3.5 Some Hints

- Using SVM with 100 training examples on digits dataset, you should get a validation accuracy of about 73% and a test accuracy of 69%.
- Using SVM with 100 training examples on faces dataset, you should get a validation accuracy of about 79% and a test accuracy of 84%.