support-vector-machines

Release 1.0

Galvanize DSI

Aug 10, 2017

CONTENTS

1 Suggested prework 1

CHAPTER

ONE

SUGGESTED PREWORK

There are 25 minutes work of video.

- ISLR: Max margin classifier
- ISLR: Support vector classifier

Main contents:

1.1 Introduction

The support vector machine (SVM) is a classification method that attempts to find a hyperplane that separates classes of observations in **feature space**.

In contrast to some other classifications methods we have seen (*e.g.* Bayesian), the SVM does not invoke a probability model for classification; instead, we aim for the direct caclulation of a **separating hyperplane**.

Consider the *logistic regression model*, which transforms a linear combination of predictors with the logistic function.

$$g_\theta(x) = \frac{1}{1+\exp(-\theta' x)}$$

Notice that when our response is $y = 1$, we want the product $\theta' x$ to be a very large, positive value so that $g_\theta(x) \to 1$, and when $y = 0$, we want this product to be a very large, negative value, so that $g_\theta(x) \to 0$.

See Chris Fonnesbeck's lecture for more

1.2 Kernels

Classic videos:

1. Kernels 1
2. Kernels 2

1.3 SVM Basics

Following the flow of ideas in the Bishop book [1] (Chapter 7) here we discuss the use of support vector machines using the Python programming language. The determination of model parameters in SVMs are an example of a convex optimization problem so it is important to remember that local solutions are also global optimums. We begin with the two-class problem

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

with $\phi(\mathbf{x})$ representing a transformation of the feature space. Training data come in the form of N input vectors $(\mathbf{x}_1 \ldots \mathbf{x}_N)$ where each has an accompanying vector t_n such that $t_n \in \{-1, 1\}$. To assign classes to a new input vector we evaluate the sign of $y(\mathbf{x})$ from Eqn (1.3): if $y(\mathbf{x}) > 0$ then the class is 1 otherwise the class will be -1. In order to find the decision hyperplane based on $\mathbf{w}$ and b that best explains the training data we use the constraint

$$t_n(\mathbf{w}^T \phi(\mathbf{x}_n) + b) = 1, n = 1, \ldots, N$$

The parameters $\mathbf{w}$ and b are selected in order to maximize the distance (also called a *margin*) between the decision boundry $\mathbf{w}^T \phi(\mathbf{x}) + b = 0$ and the closest constraints or *active* costraints. The decision boundry may be a line in two dimensions, a plane in three or a hyperplane in cases with more than three dimensions. The optimization problem is simply the maximization of $\|\mathbf{w}\|^{-1}$ which is the same as minimizing

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \frac{1}{2} \|\mathbf{w}\|^2$$

subject to the constraint Eqn (1.3). For the optimization problem the apparent disappearance of b and the use of $\frac{1}{2}$ is further explained in the text (p328). Presently we have a constrained optimization problem so we use *Lagrangian Multipliers* (one a_n per t_n). Specifically, the problem is an example of a quadratic programming (QP) problem because we are trying to minimize a quadradic function that is subject to a set of linear inequality constraints. We can then define the Lagrangian function

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

where we are minimizing w.r.t. $\mathbf{w}$ and b and maximizing w.r.t. $a = (a_1, \ldots, a_N)^T$. Setting the derivatives of Eqn (1.3) equal to zero w.r.t. $\mathbf{w}$ and b and a bit more math yeilds the *dual representation* of the maximal margin problem.

$$\tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

This representation is subject to the constraints

$$\begin{aligned} a_n &\geq 0. \quad n = 1, \ldots, N \\ \sum_{n=1}^{N} a_n t_n &= 0 \end{aligned}$$

Here the kernel function is defined by $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x})'$. Then by plugging in multipliers and the kernel function the objective function Eqa (1.3) can be rewritten as

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}_n, \mathbf{x}_m) + b$$

In Eqa (1.3) there was a dot product which sums over the M dimensions. In the newly expressed version of this funciton Eqa (1.3) we sum over the N points. This is called the kernel trick, which enables SVMs to handle non-linear problems by mapping in to higher dimensional space without directly computing $\phi(\mathbf{x})$. Often only a small number of multipliers will be non-zero and we are only required to store those traning samples.

1.3.1 Resources

- Mathieu Blondel's blog
- Wiki - SVMs

1.4 Lagrangian Multipliers

From Bishop book [1] (Appendix E).

If we want find the maximum of a function $f(x_1, x_2)$ subject to the constraint $g(x_1, x_2) = 0$ the problem can be solved by optimizing the Lagrangian function

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$$

The Lagrangian multiplier $\lambda \neq 0$ in Eqn (1.4) can have either sign. In calculus a common problem is the identification of extrema and when the problem is subject to a constraint Lagrangian multipliers are a common tool used for this optimization problem.

1.4.1 Resources

- Wiki Lagrange Multipliers

1.4.2 Bibliographic notes

1. Bishop, C. M. Jordan, M. I.; Kleinberg, J. & Schölkopf, B. (ed.) Pattern Recognition and Machine Learning Springer, 2006

Supplemental

1.5 Works cited