Project 3

Pooya Taherkhani

CS 6830 – Machine Learning

Professor Razvan Bunescu

Textbook: PRML by Bishop

June 2018

Kernel Techniques
Kernel Nearest Neighbor
Distance-weighted Nearest Neighbor
Matrix Properties
Max Margin Hyperplanes
SVM Regression
Large Margin Perceptron
Feature Selection (Document Classifier)

1. Kernel Techniques

Show that if $k_1(x, y)$ *and* $k_2(x, y)$ *are valid kernel functions, then* $k(x, y) = k_1(x, y)k_2(x, y)$ *is also a valid kernel.*

Assume

$$k_1(x, y) = \phi(x)^T\phi(y) \; ; \quad \phi(x) = \begin{bmatrix}\phi_1(x) & \dots & \phi_n(x)\end{bmatrix}^T$$
$$k_2(x, y) = \psi(x)^T\psi(y) \; ; \quad \psi(x) = \begin{bmatrix}\psi_1(x) & \dots & \psi_m(x)\end{bmatrix}^T$$

We need to show that there exists a vector function ω such that

$$k(x, y) = \omega(x)^T\omega(y)$$

Thus, we want to separate x and y in the function k to produce the vector function ω,

$$\begin{aligned} k(x, y) &= k_1(x, y)k_2(x, y) \\ &= \phi(x)^T\phi(y)\psi(x)^T\psi(y) \\ &= (\phi_1(x)\phi_1(y) + \cdots + \phi_n(x)\phi_n(y))\,(\psi_1(x)\psi_1(y) + \cdots + \psi_m(x)\psi_m(y)) \end{aligned}$$

We can proceed with our argument regardless of the relative size of n and m. That is, it can be the case that $n < m$, $n = m$, or $n > m$. By expanding the last equation above, we have

$$\begin{aligned} k(x, y) &= \sum_{i=1}^{n}\sum_{j=1}^{m} \phi_i(x)\phi_i(y)\psi_j(x)\psi_j(y) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{m} \phi_i(x)\psi_j(x)\phi_i(y)\psi_j(y) \quad \text{scalar multiplication is commutative} \end{aligned}$$

Therefore, ω is

$$\omega(x) = \begin{bmatrix}\phi_i(x)\psi_j(x)\end{bmatrix}_{nm\times 1} \qquad \begin{array}{c}\text{for all combinations of } i \text{ and } j \\ \text{where } i = 1, \dots, n \text{ and} \\ j = 1, \dots, m\end{array}$$

2. Kernel Nearest Neighbor [Exercise 6.3, page 320]

Nearest Neighbor Classification:

- find the nearest data point $x_{n'}$ to the new input x
- then $y(x) = t_{n'}$

where $y(x)$ is the predicted label for x and $t_{n'}$ is the true label of the nearest point $x_{n'}$.

$$x^T = \begin{bmatrix}x_1 & \dots & x_k\end{bmatrix}_{1\times k} \qquad \text{new input with unknown label}$$
$$x_n^T = \begin{bmatrix}x_{n1} & \dots & x_{nk}\end{bmatrix}_{1\times k} \qquad n^{th} \text{ example with known label, } n \in 1, \dots, N$$

The distance d (used to find the nearest point), at its simplest form, is defined as the Euclidean distance

$$\begin{aligned} d^2 = \|\mathbf{x} - \mathbf{x}_n\|^2 &= (x_1 - x_{n1})^2 + \cdots + (x_k - x_{nk})^2 \\ &= x_1^2 - 2x_1x_{n1} + x_{n1}^2 + \cdots + x_k^2 - 2x_kx_{nk} + x_{nk}^2 \\ &= \left(x_1^2 + \cdots + x_k^2\right) + \left(x_{n1}^2 + \cdots + x_{nk}^2\right) - 2\left(x_1x_{n1} + \cdots + x_kx_{nk}\right) \\ &= \mathbf{x}^\mathsf{T}\mathbf{x} + \mathbf{x}_n^\mathsf{T}\mathbf{x}_n - 2\mathbf{x}^\mathsf{T}\mathbf{x}_n \end{aligned}$$

each term of which is now a dot product that can be represented by a linear kernel.

$$\|\mathbf{x} - \mathbf{x}_n\|^2 = k(\mathbf{x}, \mathbf{x}) + k(\mathbf{x}_n, \mathbf{x}_n) - 2k(\mathbf{x}, \mathbf{x}_n) \tag{1}$$

Thus the predicted label, $y(\mathbf{x}) = t_n$, can be computed by finding $n \in \{1, \ldots, N\}$ that returns the smallest distance $\|\mathbf{x} - \mathbf{x}_n\|$, where N is the number of examples with known label.

Now the kernel function k in equation (1) can be replaced with any (nonlinear) valid kernel.

3. Distance-weighted Nearest Neighbor – Multi Class

Kernel-based distance-weighted nearest neighbor for K classes, where $K \geqslant 2$.

- Assume K classes for label t, that is, $t \in \{C_1, ..., C_K\}$.
- The predicted label for $\mathbf{x}$ is $y(\mathbf{x}) = \underset{t}{\operatorname{argmax}} \sum_{n=1}^{N} \kappa(\mathbf{x}, \mathbf{x}_n)\delta_t(t_n)$

  where $\delta_t(x)$ is Kronecker delta function $\delta_t(x) = \begin{cases} 1 & \text{if } x = t \\ 0 & \text{otherwise} \end{cases}$

The algorithm above works as follows. For each class C_k (where $k = 1, \ldots, K$), construct a vector of 0s and 1s with the length N (number of data points) by assigning 1 to each point $\mathbf{x}_n$ that belongs to class C_k (that is, $t_n = C_k$), and 0 otherwise. we will have K vectors of 1s and 0s, each of length N. Also for each point $\mathbf{x}_n$ compute a weight $\kappa(\mathbf{x}, \mathbf{x}_n)$ (or a similarity measure between $\mathbf{x}$ and $\mathbf{x}_n$) and multiply it by the corresponding element (which is 0 or 1) in all vectors. Out of the resulting K vectors, pick the one with the largest sum of elements. The class associated with the selected vector is the predicted class for $\mathbf{x}$.

4. Matrix Properties [Exercise 6.24, page 322]

A diagonal matrix $\mathbf{W}$ whose elements satisfy $0 < W_{ii} < 1$ is positive definite.

A symmetric $n \times n$ real matrix W is said to be positive definite if the quadratic form $z^T W z$ (which yields a scalar) is strictly positive for every non-zero column vector z of n real numbers.

$$z^T W z > 0 \quad \forall z \neq \mathbf{0}$$

We have

$$\begin{aligned}
& z^T W z = \\
& \begin{bmatrix} z_1 & \cdots & z_n \end{bmatrix} \begin{bmatrix} W_{11} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & W_{nn} \end{bmatrix} \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} = \\
& \begin{bmatrix} z_1 W_{11} & \cdots & z_n W_{nn} \end{bmatrix} \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} = \\
& z_1^2 W_{11} + \cdots + z_n^2 W_{nn} \qquad (1)
\end{aligned}$$

We show that the diagonal form $z_1^2 W_{11} + \cdots + z_n^2 W_{nn}$ is strictly positive. Note that $W_{ii} > 0$ and $z_i^2 \geqslant 0$, which implies $z_i^2 W_{ii} \geqslant 0$ for all $i \in \{1, \ldots, n\}$. However, there exists at least one $k \in \{1, \ldots, n\}$ for which $z_k > 0$ because z is a non-zero vector. Therefore, $z_k^2 W_{kk} > 0$. Thus, expression (1) is strictly greater than 0. □

The sum of two positive definite matrices is itself positive definite.

$$z^T A z > 0\ ,\ z^T B z > 0 \quad \forall z_{n \times 1} \neq \mathbf{0}$$

where A and B are $n \times n$ matrices (They have to be of the same size because they are to be added together). We can rewrite $z^T A z > 0$ as

$$\begin{bmatrix} z_1 & \cdots & z_n \end{bmatrix} \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} > 0$$

$$\begin{bmatrix} z_1 a_{11} + \cdots + z_n a_{n1} & \cdots & z_1 a_{1n} + \cdots + z_n a_{nn} \end{bmatrix} \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} > 0$$

$$(z_1^2 a_{11} + \cdots + z_1 z_n a_{n1}) + \cdots + (z_1 z_n a_{1n} + \cdots + z_n^2 a_{nn}) > 0$$

The last two lines can be written as

$$\begin{aligned}
\begin{bmatrix}\sum_{i=1}^{n} z_i a_{i1} & \cdots & \sum_{i=1}^{n} z_i a_{in}\end{bmatrix}\begin{bmatrix}z_1\\ \vdots \\ z_n\end{bmatrix} &> 0\\
z_1\sum_{i=1}^{n} z_i a_{i1} + \cdots + z_n \sum_{i=1}^{n} z_i a_{in} &> 0\\
\sum_{j=1}^{n}\sum_{i=1}^{n} z_j z_i a_{ij} &> 0\\
\sum_{i=1}^{n}\sum_{j=1}^{n} z_i z_j a_{ij} &> 0 \qquad (2)
\end{aligned}$$

Similarly, we can rewrite the quadratic form $z^{\mathsf{T}}Bz > 0$ as

$$\sum_{i=1}^{n}\sum_{j=1}^{n} z_i z_j b_{ij} > 0 \qquad (3)$$

We show that $z^{\mathsf{T}}(A+B)z$ is strictly positive for any non-zero vector z of size n. We have

$z^{\mathsf{T}}(A+B)z =$

$$\begin{bmatrix}z_1 & \cdots & z_n\end{bmatrix}\left(\begin{bmatrix}a_{11} & \cdots & a_{1n}\\ \vdots & \ddots & \vdots\\ a_{n1} & \cdots & a_{nn}\end{bmatrix} + \begin{bmatrix}b_{11} & \cdots & b_{1n}\\ \vdots & \ddots & \vdots\\ b_{n1} & \cdots & b_{nn}\end{bmatrix}\right)\begin{bmatrix}z_1\\ \vdots \\ z_n\end{bmatrix} =$$

$$\begin{bmatrix}z_1 & \cdots & z_n\end{bmatrix}\begin{bmatrix}a_{11}+b_{11} & \cdots & a_{1n}+b_{1n}\\ \vdots & \ddots & \vdots\\ a_{n1}+b_{n1} & \cdots & a_{nn}+b_{nn}\end{bmatrix}\begin{bmatrix}z_1\\ \vdots \\ z_n\end{bmatrix} =$$

$$\begin{bmatrix}z_1(a_{11}+b_{11}) + \cdots + z_n(a_{n1}+b_{n1}) & \cdots & z_1(a_{1n}+b_{1n}) + \cdots + z_n(a_{nn}+b_{nn})\end{bmatrix}\begin{bmatrix}z_1\\ \vdots \\ z_n\end{bmatrix} =$$

$$(z_1^2(a_{11}+b_{11}) + \cdots + z_1 z_n(a_{n1}+b_{n1})) + \cdots + (z_1 z_n(a_{1n}+b_{1n}) + \cdots + z_n^2(a_{nn}+b_{nn})) =$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} z_i z_j (a_{ij}+b_{ij}) =$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} z_i z_j a_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{n} z_i z_j b_{ij} > 0$$

which is true because of (2) and (3). □

5. Max Margin Hyperplanes [Exercise 7.2, page 357]

The decision boundary $\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n) + b = 0$ whose parameters $\boldsymbol{w}$ and b are the optimal solution of the following max margin optimization problem does not depend on the value of γ.

$$\begin{aligned}
&\operatorname*{argmax}_{\boldsymbol{w},b}\left(\min_n\left(\frac{t_n(\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n)+b)}{\|\boldsymbol{w}\|}\right)\right)\\
&\text{subject to}\\
&\qquad t_n(\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n)+b) \geqslant \gamma\\
&\qquad \gamma > 0.
\end{aligned}$$

This box can be skipped if you prefer a shorter answer

We assume that the training data is separable by a linear model $y(\boldsymbol{x}_n) = \boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n) + b$ in the feature space such that $y(\boldsymbol{x}_n) \geqslant 0$ if the label for $\boldsymbol{x}_n$ is $t_n = +1$, and $y(\boldsymbol{x}_n) < 0$ if the label is $t_n = -1$. This implies that $y(\boldsymbol{x}_n) = 0$ is the decision boundary.

The *margin* is defined to be the distance of the closest point $\boldsymbol{x}_n$ to the decision surface $\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n) + b = 0$. It can be proved that the distance of any point $\boldsymbol{x}_n$ from the surface is $\frac{|y(\boldsymbol{x}_n)|}{\|\boldsymbol{w}\|} = \frac{|\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n)+b|}{\|\boldsymbol{w}\|}$. Now since we assume that our data is linearly separable, there exist some $\boldsymbol{w}$ and b such that the model $y(\boldsymbol{x}_n) = \boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n) + b$ can correctly classify all data points. Therefore, $t_n(\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n)+b) > 0$ for all $\boldsymbol{x}_n$. Thus, the distance can be written as $\frac{t_n(\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n)+b)}{\|\boldsymbol{w}\|}$ (note that $|t_n| = 1$).

To find the margin, we need to find x_n that minimizes the distance. Therefore, the margin is $\min_n\left(\frac{t_n(\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n)+b)}{\|\boldsymbol{w}\|}\right)$ subject to $t_n(\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n) + b) > 0$.

In the max margin classifier, we search for $\boldsymbol{w}$ and b that maximize the margin (to minimize the generalization error), that is, $\operatorname*{argmax}_{\boldsymbol{w},b}\left(\min_n\left(\frac{t_n(\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n)+b)}{\|\boldsymbol{w}\|}\right)\right)$ subject to $t_n(\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n) + b) > 0$.

No matter what is the optimal value for $\boldsymbol{w}$ and b, the scaling $\boldsymbol{w} \to \kappa\boldsymbol{w}$ and $b \to \kappa b$ does not change the classifier because it doesn't change the distances from the separating hyperplane, $\frac{t_n(\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n)+b)}{\|\boldsymbol{w}\|}$ (because it multiplies the numerator and the denominator by the same factor κ), so it doesn't change the distance of the closest point to the hyperplane, $\min_n\left(\frac{t_n(\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n)+b)}{\|\boldsymbol{w}\|}\right)$, either.

Therefore, we can use this scaling to set $t_n(\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n) + b) = \gamma$ for the closest point to the hyperplane, where γ is an arbitrary positive constant (because there exists a κ for any given γ). That is, $\min_n\left(t_n(\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n) + b)\right) = \gamma$. This implies that (a) all points satisfy the constraint $t_n(\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_n) + b) \geqslant \gamma$, and (b) to maximize the margin we need to $\operatorname*{argmax}_{\boldsymbol{w},b}\frac{\gamma}{\|\boldsymbol{w}\|}$ (since $\|\boldsymbol{w}\|$ is independent of n) which is equivalent to $\operatorname*{argmax}_{\boldsymbol{w},b}\frac{1}{\|\boldsymbol{w}\|}$ (since γ is independent of $\boldsymbol{w}$ and b), or $\operatorname*{argmin}_{\boldsymbol{w},b}\|\boldsymbol{w}\|$. $\square$

6. SVM Regression [Exercise 7.7, page 357]

SVM for regression, primal problem:

$$\boxed{\begin{array}{ll} \text{minimize}_{\vec{w},b} & J(\vec{w},b)=\frac{1}{2}\|\vec{w}\|^2+C\sum_{n=1}^{N}(\xi_n+\hat{\xi}_n) \\ \text{subject to} & t_n \leq \vec{w}^T\vec{\varphi}(\vec{x}_n)+b+\varepsilon+\xi_n \\ & t_n \geq \vec{w}^T\vec{\varphi}(\vec{x}_n)+b-\varepsilon-\hat{\xi}_n \\ & \xi_n, \hat{\xi}_n \geq 0 \qquad \forall n\in\{1,\cdots,N\} \end{array}}$$

$\Leftrightarrow t_n-\vec{w}^T\vec{\varphi}(\vec{x}_n)-b-\varepsilon-\xi_n\leq 0$

$\Leftrightarrow -t_n+\vec{w}^T\vec{\varphi}(\vec{x}_n)+b-\varepsilon-\hat{\xi}_n\leq 0$

$\Leftrightarrow -\xi_n, -\hat{\xi}_n \leq 0$

$$L_P(\vec{w},b,\vec{\xi},\hat{\vec{\xi}},\vec{\alpha},\hat{\vec{\alpha}},\vec{\mu},\hat{\vec{\mu}})=\frac{1}{2}\|\vec{w}\|^2+C\sum_{n=1}^{N}(\xi_n+\hat{\xi}_n)+\sum_{n=1}^{N}\alpha_n(t_n-\vec{w}^T\vec{\varphi}(\vec{x}_n)-b-\varepsilon-\xi_n)$$
$$+\sum_{n=1}^{N}\hat{\alpha}_n(-t_n+\vec{w}^T\vec{\varphi}(\vec{x}_n)+b-\varepsilon-\hat{\xi}_n)+\sum_{n=1}^{N}\mu_n(-\xi_n)+\sum_{n=1}^{N}\hat{\mu}_n(-\hat{\xi}_n)$$

$\alpha_n, \hat{\alpha}_n, \mu_n, \hat{\mu}_n \geq 0$ (Lagrange multipliers)

derivative of L_P with respect to primal variables

$$\frac{\partial L_P}{\partial \vec{w}}=0 \Rightarrow \vec{w}-\sum_{n=1}^{N}\alpha_n\vec{\varphi}(\vec{x}_n)+\sum_{n=1}^{N}\hat{\alpha}_n\vec{\varphi}(\vec{x}_n)=0$$
(since $\frac{1}{2}\|\vec{w}\|^2=\frac{1}{2}\vec{w}^T\vec{w}$, $\frac{d}{d\vec{x}}(\vec{x}^T\vec{x})=2\vec{x}$, $\frac{d}{dx}\vec{x}^T\vec{a}=\vec{a}$)
$$\Rightarrow \vec{w}=\sum_{n=1}^{N}(\alpha_n-\hat{\alpha}_n)\vec{\varphi}(\vec{x}_n) \quad (I)$$

$$\frac{\partial L_P}{\partial b}=0 \Rightarrow -\sum_{n=1}^{N}(-\alpha_n+\hat{\alpha}_n)=0 \Rightarrow \sum_{n=1}^{N}(\alpha_n-\hat{\alpha}_n)=0 \quad (II)$$

$$\frac{\partial L_P}{\partial \xi_n}=0 \Rightarrow CN-\alpha_n N-\mu_n N=0 \Rightarrow \alpha_n+\mu_n=C \quad (III)$$

$$\frac{\partial L_P}{\partial \hat{\xi}_n}=0 \Rightarrow CN-\hat{\alpha}_n N-\hat{\mu}_n N=0 \Rightarrow \hat{\alpha}_n+\hat{\mu}_n=C \quad (IV)$$

substitute in L_P to get L_D

$$L_D(\vec{\alpha},\hat{\vec{\alpha}},\vec{\mu},\hat{\vec{\mu}})=\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(\alpha_n-\hat{\alpha}_n)(\alpha_m-\hat{\alpha}_m)\vec{\varphi}(\vec{x}_n)^T\vec{\varphi}(\vec{x}_m)+C\sum_{n=1}^{N}(\xi_n+\hat{\xi}_n)$$
$$+\sum_{n=1}^{N}t_n(\alpha_n-\hat{\alpha}_n)-\sum_{n=1}^{N}(\alpha_n-\hat{\alpha}_n)\vec{w}^T\vec{\varphi}(\vec{x}_n)-b\underbrace{\cancel{\sum_{n=1}^{N}(\alpha_n-\hat{\alpha}_n)}}_{0\ (II)}$$
$$-\varepsilon\sum_{n=1}^{N}(\alpha_n+\hat{\alpha}_n)-\sum_{n=1}^{N}\underbrace{\cancel{(\alpha_n+\mu_n)}}_{C\ (III)}\xi_n-\sum_{n=1}^{N}\underbrace{\cancel{(\hat{\alpha}_n+\hat{\mu}_n)}}_{C\ (IV)}\hat{\xi}_n$$

(still has $\vec{w}$, b, $\vec{\xi}$, and $\hat{\vec{\xi}}$)

$$=\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(\alpha_n-\hat{\alpha}_n)(\alpha_m-\hat{\alpha}_m)\vec{\varphi}(\vec{x}_n)^T\vec{\varphi}(\vec{x}_m)+\sum_{n=1}^{N}(\alpha_n-\hat{\alpha}_n)t_n$$
$$-\sum_{n=1}^{N}\sum_{m=1}^{N}(\alpha_n-\hat{\alpha}_n)(\alpha_m-\hat{\alpha}_m)\vec{\varphi}(\vec{x}_n)^T\vec{\varphi}(\vec{x}_m)-\varepsilon\sum_{n=1}^{N}(\alpha_n+\hat{\alpha}_n)$$

Dual problem

$$\boxed{\text{maximize}\quad L_D(\vec{\alpha},\hat{\vec{\alpha}})=-\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(\alpha_n-\hat{\alpha}_n)(\alpha_m-\hat{\alpha}_m)k(\vec{x}_n,\vec{x}_m)-\varepsilon\sum_{n=1}^{N}(\alpha_n-\hat{\alpha}_n)+\sum_{n=1}^{N}(\alpha_n-\hat{\alpha}_n)t_n} \quad (**)$$

(**) $\boxed{k(\vec{x}_n,\vec{x}_m)=\vec{\varphi}(\vec{x}_n)^T\vec{\varphi}(\vec{x}_m)}$

$\boxed{\text{subject to}}$

III $\rightarrow$ $0\leq\alpha_n\leq C$

IV $\rightarrow$ $0\leq\hat{\alpha}_n\leq C$

II $\rightarrow$ $\sum_{n=1}^{N}(\alpha_n-\hat{\alpha}_n)=0$

Complementary slackness (*)

$\alpha_n(t_n-y_n-\varepsilon-\xi_n)=0$

$\hat{\alpha}_n(-t_n+y_n-\varepsilon-\hat{\xi}_n)=0$

$(C-\alpha_n)\xi_n=0 \leftarrow$ (III)

$(C-\hat{\alpha}_n)\hat{\xi}_n=0 \leftarrow$ (IV)

(*) $\leftarrow$ (I)

$y(\vec{x})=\sum_{n=1}^{N}(\alpha_n-\hat{\alpha}_n)k(\vec{x},\vec{x}_n)+b$

$y(\vec{x})=\vec{w}\vec{\varphi}(\vec{x})+b$

7. Large Margin Perceptron

Let $\mathbf{u}$ be a current vector of parameters and $\mathbf{x}$ and $\mathbf{y}$ two training examples such that $\mathbf{u}^\mathsf{T}(\mathbf{x}-\mathbf{y}) < 1$. Use the technique of Lagrange multipliers to find a new vector of parameters $\mathbf{w}$ as the solution to the convex optimization problem below: minimize: $J(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}-\mathbf{u}\|^2$ *subject to:* $\mathbf{w}^\mathsf{T}(\mathbf{x}-\mathbf{y}) \geqslant 1$

$\vec{u}$: current vector of parameters.

(*) $\vec{x}, \vec{y}$: two training examples $\qquad \vec{u}^T(\vec{x}-\vec{y}) < 1 \quad$ (**)

$\vec{w}$: new vector of parameters

primal problem

minimize $\quad J(\vec{w}) = \frac{1}{2}\|\vec{w}-\vec{u}\|^2$

subject to $\quad \vec{w}^T(\vec{x}-\vec{y}) \geqslant 1 \quad \Rightarrow \quad 1-\vec{w}(\vec{x}-\vec{y}) \leqslant 0$

$$L_P(\vec{w},\alpha) = \frac{1}{2}\|\vec{w}-\vec{u}\|^2 + \alpha\left(1-\vec{w}^T(\vec{x}-\vec{y})\right)$$

primal Lagrangian

$$= \frac{1}{2}(\vec{w}-\vec{u})^T(\vec{w}-\vec{u}) + \alpha - \alpha\,\vec{w}^T(\vec{x}-\vec{y})$$

$\vec{u}, \vec{x}, \vec{y}$: constant vectors

$$\frac{\partial L_P}{\partial \vec{w}} = \overbrace{\left(\frac{1}{2}\right)\left(2(\vec{w}-\vec{u})\,1\right)}^{\text{chain rule}} - \alpha(\vec{x}-\vec{y}) = 0$$

$$\Rightarrow \quad \vec{w}-\vec{u} = \alpha(\vec{x}-\vec{y})$$

$$\Rightarrow \quad \vec{w} = \vec{u} + \alpha(\vec{x}-\vec{y}) \quad (***)$$

$\frac{d}{\vec{x}}\vec{x}^T\vec{x} = 2\vec{x}$

$\frac{d}{d\vec{x}}\vec{x}^T\vec{a} = \vec{a}$

set the derivative of Lagrangian with respect to the primal variable to zero

$$L_D(\alpha) = \frac{\alpha^2}{2}\|\vec{x}-\vec{y}\|^2 + \alpha - \alpha\left(\vec{u}+\alpha(\vec{x}-\vec{y})\right)^T(\vec{x}-\vec{y})$$

$$= \frac{\alpha^2}{2}\|\vec{x}-\vec{y}\|^2 + \alpha - \alpha\vec{u}^T(\vec{x}-\vec{y}) - \alpha^2\|\vec{x}-\vec{y}\|^2$$

$$= -\frac{\|\vec{x}-\vec{y}\|^2}{2}\alpha^2 + \left(1-\vec{u}^T(\vec{x}-\vec{y})\right)\alpha$$

substitute primal variable back in L_P to get L_D

dual problem

maximize $\quad L_D(\alpha) = -\frac{\|\vec{x}-\vec{y}\|^2}{2}\alpha^2 + \left(1-\vec{u}^T(\vec{x}-\vec{y})\right)\alpha$

subject to $\quad \alpha \geqslant 0$

a maximization problem in one variable.

$$\frac{dL_D}{d\alpha} = 0 \quad \Rightarrow \quad -\|\vec{x}-\vec{y}\|^2\alpha + 1 - \vec{u}^T(\vec{x}-\vec{y}) = 0$$

$$\Rightarrow \quad \alpha = \frac{1-\vec{u}^T(\vec{x}-\vec{y})}{\|\vec{x}-\vec{y}\|^2}$$

set the derivative to zero to compute optimal α

we can check that objective function is concave

$$\frac{d^2L_D}{d\alpha^2} = -\|\vec{x}-\vec{y}\|^2 < 0$$

$\alpha \geqslant 0$ because $\|\vec{x}-\vec{y}\|^2 > 0 \quad (*)$

and $1-\vec{u}^T(\vec{x}-\vec{y}) > 0 \quad (**)$

$$\boxed{\vec{w} = \vec{u} + \frac{1-\vec{u}^T(\vec{x}-\vec{y})}{\|\vec{x}-\vec{y}\|^2}(\vec{x}-\vec{y})}$$

substitute α back into (***)

8. Feature Selection (Document Classifier)

(a) The data is obtained from UC Irvine machine learning, Dexter dataset.

The data is derived from a text corpus, formatted in the bag-of-word representation. There are a total of $20,000$ features, where each feature represents the frequency of a word stem used in text.

Only the training and validation data are useful to us because the labels are provided for them. Each of these two sets includes 300 examples where half of them are labeled 1 (meaning the text is related to "corporate acquisition") and the other half -1.

There is a considerable number of features that are zero for all examples. Around half of the features are real text attributes, and the other half are just dummy features that were added to the data in a randomized manner to make for a feature selection exercise.

(b) According to the dataset description, the features represent "frequencies of occurrences of word stems in text". So the feature values are relevant as they indicate some sort of measurement, and thus they are not nominal features.

(Nominal numbers uniquely identify an object or entity. For instance, zip codes, or social security numbers identify geographic areas or individuals, respectively. Nominal numbers can be compared only for equality–to see if two geographic areas or two people are actually the same–but the value of the numbers do not mean anything. So comparison of their relative value, their difference, their sum, etc. is nonsensical.)

Therefore, we cannot use Mutual Information and Chi-square Statistic which can handle only nominal features, but we can use Pearson Correlation Coefficient, Signal-to-Noise Ratio, and T-test.

We use only the training data for ranking features. So we rank features using each of the three measures of correlation with the label. Then we select the top N features to be used for training linear SVM and the nearest neighbor algorithm.

The training data was formatted into the file `data/svm-data.train` using the Python script `code/preprocess-data-libsvm.py` to fit the format required by LIBSVM, then it was read into a sparse matrix using the Python script `code/rank-features.py`.

The feature matrix for each data set (that contains 300 examples of each $20,000$ features) has a total of $6,000,000$ elements of which only a small minority of $28,218$ elements are not zero. Therefore, storing the feature matrix as a sparse matrix seems a natural choice. This was done using the `scipy.sparse` library.

Pearson Correlation Coefficient

The mean value for labels, $\overline{y}$, of Dexter dataset is 0 as half of the examples are labeled 1 and the other half are labeled -1.

The absolute value of the correlation coefficient matters to us because we only care about the magnitude of correlation between a feature and the label. For the purpose of feature selection, we do not care if the correlation is direct or inverse.

If the correlation coefficient is undefined (if the denominator is 0), we replace the associated feature with 0 which implies the label is independent of that feature.

The division by zero in the correlation coefficient

$$\rho(X, Y) = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}}$$

happens if and only if $\sum_{i=1}^{n}(X_i - \overline{X})^2 = 0$ because $\sum_{i=1}^{n}(Y_i - \overline{Y})^2$ cannot be zero for all examples of a classification problem with at least two classes. Now, $\sum_{i=1}^{n}(X_i - \overline{X})^2 = 0$ happens if and only if all nonnegative $(X_i - \overline{X})^2$ terms are zero (for all $i = 1, \ldots, n$). In which case, the numerator has to be zero, too. So all division by zero cases are actually a $\frac{0}{0}$ case for the correlation coefficient. And this happens for a feature that has the same value across all examples. These features are useless for training purposes. So by manually setting their correlation coefficient to zero, they will be ranked last for feature selection.

The absolute value of the correlation coefficient is between 0 and 1.

Signal-to-Noise Ratio

In the `rank-features.py` script, we use the following formula to compute the standard deviation of a feature X.

$$\sigma = \sqrt{\overline{X^2} - \overline{X}^2}$$

Signal-to-Noise (S2N) ratio of feature X in relation to label Y is then computed as

$$\mu(X, Y) = \frac{|\mu_+ - \mu_-|}{\sigma_+ + \sigma_-}$$

where μ_+ and σ_+ are the sample mean and standard deviation of X for which $Y = 1$.

All S2N ratio values for our data is between 0 and 1.

T-test

$$T(X, Y) = \frac{|\mu_+ - \mu_-|}{\sqrt{\sigma_+^2/m_+ + \sigma_-^2/m_-}}$$

where m_+ is the number of examples with label 1.

The maximum value of t-test can be larger than 1.

Sort columns of feature matrix

We sort the columns of the feature matrix to have the most relevant feature on the first column of the matrix to the least relevant on the last column.

This sorting is done based on each of the three filters. The sort will be based on the value of the filter in descending order. That is, the feature with the highest filter value will be moved to the left-most column in the matrix.

Normalization

After feature selection, we divide each feature vector x_i by its Euclidean norm, so all of the resulting feature vectors $\frac{x_i}{|x_i|}$ will have a magnitude of 1.

Sort features first, then normalize.

We first sort the column indices of the feature matrix using a filter as sorting keys (a filter value will be computed for each column). We then normalize features vectors (matrix rows) in the feature matrix, and finally sort the columns of the normalized feature matrix by applying the sorted column indices.

This sequence of operations allows us to carry out the normalization only once regardless of the number of filters that we wish to use.