CS 474/574 Machine Learning
4. Support Vector Machines (SVMs)

Prof. Dr. Forrest Sheng Bao
Dept. of Computer Science
Iowa State University
Ames, IA, USA

September 18, 2020

SVM is a kind of linear classifiers. But a unique type of.

All samples are equal. But some samplers are equaler.

- Let's first see a demo of a linear classifier for linearly separable cases. Pay attention to the prediction outcome.
- Think about the error-based loss function for a classifier: $\sum_i (\hat{y} - y)^2$ where y is the ground truth label and $\hat{(y)}$ is the prediction.
- If $y = +1$ and $\hat{y} = +1.5$, should the error be 0.25 or 0 (because properly classified)?

The perceptron algorithm

- Recall earlier that a sample $(\mathbf{x}_i, y_i)$ is correctly classified if $\mathbf{w}^T\mathbf{x}_i y_i > 0$.
- Let's define a new cost function to be minimized: $J(\mathbf{w}) = \sum_{x_i \in \mathcal{M}} -\mathbf{w}^T\mathbf{x}_i y_i$ where $\mathcal{M}$ is the set of all samples misclassified ($\mathbf{W}^T\mathbf{X}_i y_i < 0$).
- Then, $\nabla J(\mathbf{w}) = \sum_{\mathbf{x}_i \in \mathcal{M}} -\mathbf{X}_i y_i$ (because $\mathbf{w}$ is the coefficients.)
- Only those misclassified matter!
- Batch perceptron algorithm: In each batch, computer $\nabla J(\mathbf{w})$ for all samples misclassified using the same current $\mathbf{w}$ and then update.

Single-sample perceptron algorithm

- Another common type of perceptron algorithm is called single-sample perceptron algorithm.
- Update $\mathbf{w}$ whenever a sample is misclassified.
  1. Initially, $\mathbf{w}$ has arbitrary values. $k = 1$.
  2. In the k-th iteration, use sample $\mathbf{x}_j$ such that $j = k \mod n$ to update the $\mathbf{w}$ by:

$$\mathbf{W}_{k+1} = \begin{cases} \mathbf{W}_k + \rho \mathbf{X}_j y_j & \text{, if } \mathbf{W}_j^T \mathbf{X}_\mathbf{j} y_j \leq 0, \text{ (wrong prediction)} \\ \mathbf{W}_k & \text{, if } \mathbf{W}_j^T \mathbf{X}_\mathbf{j} y_j > 0 \text{ (correct classification)} \end{cases}$$

     where ρ is a constant called **learning rate**.
  3. The algorithm terminates when all samples are classified correctly.
- Note that $\mathbf{x}_k$ is not necessarily the k-th training sample due to the loop.

An example of single-sample preceptron algorithm

- Feature vectors and labels:
  - $\mathbf{x}'_1 = (0,0)^T$, $y_1 = 1$
  - $\mathbf{x}'_2 = (0,1)^T$, $y_2 = 1$
  - $\mathbf{x}'_3 = (1,0)^T$, $y_3 = -1$
  - $\mathbf{x}'_4 = (1,1)^T$, $y_4 = -1$
- First, let's augment them and multiply with the labels:
  - $\mathbf{x}_1 y_1 = (0,0,1)^T$,
  - $\mathbf{x}_2 y_2 = (0,1,1)^T$,
  - $\mathbf{x}_3 y_3 = (-1,0,-1)^T$
  - $\mathbf{x}_4 y_4 = (-1,-1,-1)^T$

0. Begin our iteration. Let $\mathbf{w}_1 = (0,0,0)^T$ and $\rho = 1$.
1. $\mathbf{W}_1^T \cdot \mathbf{x}_1 y_1 = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = 0 \leq 0$.

   Need to update $\mathbf{W}$: $\mathbf{W}_2 =$
   $$\mathbf{W}_1 + \rho \cdot \mathbf{x}_1 y_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

2. $\mathbf{W}_2^T \cdot \mathbf{x}_2 y_2 = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = 1 > 0$. No updated need. But since $\mathbf{w}$ so far does not classify all samples correctly, we need to keep going. Just let $\mathbf{w}_3 = \mathbf{w}_2$.

An example of preceptron algorithm (cond.)

Continue in perceptron.ipynb

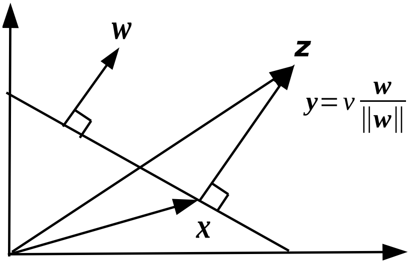
14. In the end, we have $\mathbf{W}_{14} = \begin{pmatrix} -3 \\ 0 \\ 2 \end{pmatrix}$, let's verify how well it works
    - $$\begin{cases} \mathbf{w}_{14} \cdot \mathbf{x}_1 y_1 & = 1 > 0 \\ \mathbf{w}_{14} \cdot \mathbf{x}_2 y_2 & = 1 > 0 \\ \mathbf{w}_{14} \cdot \mathbf{x}_3 y_3 & = 1 > 0 \\ \mathbf{w}_{14} \cdot \mathbf{x}_4 y_4 & = 1 > 0 \end{cases}$$

- Mission accomplished!
- Note that the perceptron algorithm will not *converge* unless the data is linearly separable.
- What is $\mathbf{w}$ exactly? A linear composition of all training samples!
- Do all samples contribute to $\mathbf{w}$? Not really!

Getting ready for SVMs

- Earlier our discussion used the augmented definition of linear binary classifier: the feature vector $\mathbf{x} = (x_1, \ldots, x_n, 1)^T$ and the weight vector $\mathbf{w} = (w_1, \ldots, w_n, w_b)^T$. The hyperplane is an equation $\mathbf{w}^T\mathbf{x} = 0$. If $\mathbf{w}^T\mathbf{x} > 0$, then the sample belongs to one class. If $\mathbf{w}^T\mathbf{x} < 0$, the other class.
- Let's go back to the un-augmented version. Let $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ and $\mathbf{w} = [w_1, w_2, \ldots, w_n]^T$. If $\mathbf{w}^T\mathbf{x} + w_b > 0$ then $\mathbf{x} \in C_1$. If $\mathbf{w}^T\mathbf{x} + w_b < 0$ then $\mathbf{x} \in C_2$. The equation $\mathbf{w}^T\mathbf{x} + w_b = 0$ is the hyperplane, where $\mathbf{w}$ only determines the direction of the hyperplane. To build a classifier is to search for the values for $w_1, \ldots, w_n$ and w_b, the bias/threshold.
- For convenience, we denote $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$.
- We have proved that $\mathbf{w}$, augmented or not, is perpendicular to the hyperlane.

What is the distance from a sample $\mathbf{z}$ to the hyperplane?



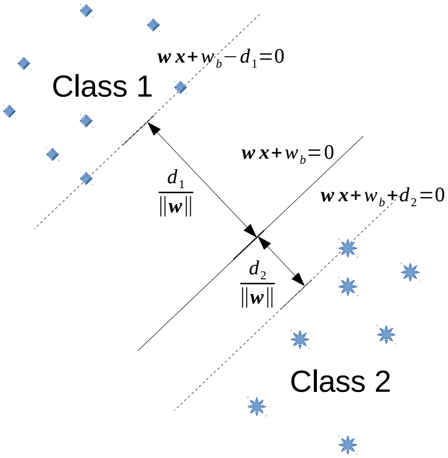
- Let the point on the hyperplane closest to $\mathbf{z}$ be $\mathbf{x}$. Define $\mathbf{y} = \mathbf{x} - \mathbf{z}$.
- Because both $\mathbf{y}$ and $\mathbf{w}$ are perpendicular to the hyperplane, we can rewrite $\mathbf{y} = v\frac{\mathbf{w}}{||\mathbf{w}||}$, where v is the Euclidean distance from $\mathbf{z}$ to $\mathbf{x}$ (what we are trying to get) and $\frac{\mathbf{w}}{||\mathbf{w}||}$ is the unit vector pointing at the direction of $\mathbf{w}$.
- Therefore, $\mathbf{z} = \mathbf{x} + v\frac{\mathbf{w}}{||\mathbf{w}||}$.

- The prediction for $\mathbf{z}$ is then (subsituting into linear classifier equation):

$$\begin{aligned}
\mathbf{w}^T\mathbf{z} + w_b &= \mathbf{w}^T(\mathbf{x} + v\frac{\mathbf{w}}{||\mathbf{w}||}) + w_b \\
&= \mathbf{w}^T\mathbf{x} + v\frac{\mathbf{w}^T\mathbf{w}}{||\mathbf{w}||} + w_b = \underbrace{\mathbf{w}^T\mathbf{x} + w_b}_{=0,\text{by definition}} + v\frac{\mathbf{w}^T\mathbf{w}}{||\mathbf{w}||} \\
&= v\frac{\mathbf{w}^T\mathbf{w}}{||\mathbf{w}||} = v\frac{||\mathbf{w}||^2}{||\mathbf{w}||} = v||\mathbf{w}||.
\end{aligned}$$

- Finally, $v = \frac{\mathbf{w}^T\mathbf{z} + w_b}{||\mathbf{w}||}$.

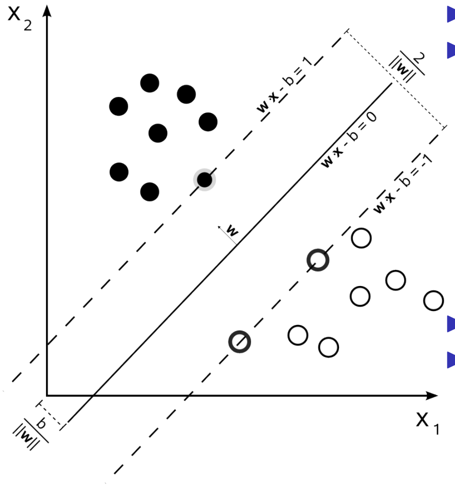
Hard margin linear SVM



- Assume that the minimum distance from any point in Class C_1 and C_2 to the hyperplane are $d_1/||\mathbf{w}||$ and $d_2/||\mathbf{w}||$, respectively, where $d_1, d_2 > 0$.
- Then we have $\mathbf{w}^T\mathbf{x} + w_b - d_1 \geq 0, \forall x \in C_1$, and $\mathbf{w}^T\mathbf{x} + w_b + d_2 \geq 0, \forall x \in C_2$.
- To make the classifier more discriminant, we want to maximize the distance between the two classes, known as the **margin**, i.e. $\max\left(\frac{d_1}{||\mathbf{w}||} + \frac{d_2}{||\mathbf{w}||}\right)$.
- An SVM classifier is also called a *Maximum Margin Classifier*.
- Assuming the two classes are linearly separable, our problem becomes:

$$\begin{cases} \max & \frac{d_1}{||\mathbf{w}||} + \frac{d_2}{||\mathbf{w}||} \\ s.t. & \mathbf{w}^T\mathbf{x} + w_b - d_1 \geq 0, \forall x \in C_1 \\ & \mathbf{w}^T\mathbf{x} + w_b + d_2 \geq 0, \forall x \in C_2 \end{cases}$$

Hard margin linear SVM (cond.)



- We prefer $d_1 = d_2$: both classes are equal.
- Since d_1 and d_2 are constants, we can let them be 1. Let the label $y_k \in \{+1, -1\}$ for sample $\mathbf{x}_k$, we can get a different form:

$$\begin{cases} \max & \frac{2}{||\mathbf{w}||} \\ s.t. & y_k(\mathbf{w}^T\mathbf{x}_k + w_b) \geq 1, \forall \mathbf{x}_k \in C_1 \cup C_2. \end{cases}$$

- Maximizing $\frac{2}{||\mathbf{w}||}$ is equivalent to minimizing $\frac{||\mathbf{w}||}{2}$.
- Finally, we transform it into a quadratic programming problem (**the primal form of SVMs**):

$$\begin{cases} \min & \frac{1}{2}||\mathbf{w}||^2 = \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ s.t. & y_k(\mathbf{w}^T\mathbf{x}_k + w_b) \geq 1, \forall \mathbf{x}_k. \end{cases}$$

Recap: the Karush-Kuhn-Tucker conditions

- Given a nonlinear optimization problem

$$\begin{cases} \min & f(\mathbf{x}) \\ s.t. & h_k(\mathbf{x}) \geq 0, \forall k \in [1..K], \end{cases}$$

where $\mathbf{x}$ is a vector, and $h_k(\cdot)$ is linear, its Lagrange multiplier (or Lagrangian) is:

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \sum_{k=1}^{K} \lambda_k h_k(\mathbf{x})$$

- The necessary condition that the problem above has a solution is KKT condition:

$$\begin{cases} \frac{\partial L}{\partial \mathbf{x}} = \mathbf{0}, & \\ \lambda_k \geq 0, & \forall k \in [1..K] \\ \lambda_k h_k(\mathbf{x}) = 0, & \forall k \in [1..K] \end{cases}$$

Properties of hard margin linear SVM

- The KKT condition to the SVM problem is

$$\begin{cases} A: \frac{\partial L}{\partial w} = \mathbf{0}, \\ B: \frac{\partial L}{\partial w_b} = 0, \\ C: \lambda_k \geq 0, & \forall k \in [1..K] \\ D: \lambda_k[y_k(\mathbf{w}^T\mathbf{x_k} + w_b) - 1] = 0, & \forall k \in [1..K] \end{cases}$$

- Let's solve it.

$$A: \frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{k=1}^{K} \lambda_k y_k \mathbf{x_k} \Rightarrow \mathbf{w} = \sum_{k=1}^{K} \lambda_k y_k \mathbf{x_k}$$

$$B: \frac{\partial L}{\partial w_b} = \sum_{k=1}^{K} \lambda_k y_k = 0$$

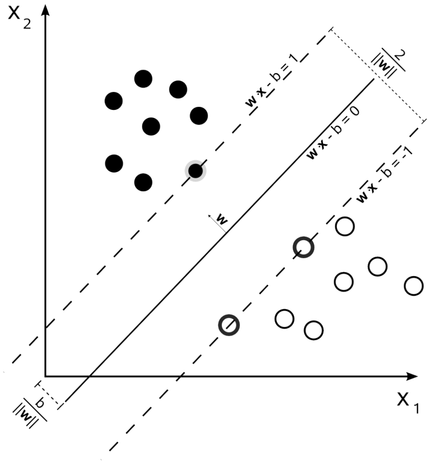
- Because λ_k is either positive or 0, the solution of the SVM problem is only associated with samples that $\lambda_k \neq 0$. Denote them as

Properties of hard margin linear SVM (cont.)

- Therefore, Eq. A can be rewritten into

$$\mathbf{w} = \sum_{\mathbf{x}_k \in N_s} \lambda_k y_k \mathbf{x_k}$$

  The samples $\mathbf{x}_k \in N_s$ collectively determine the $\mathbf{w}$, and thus called **support vectors**, supporting the solution.
- The support vectors also have an interesting "visual" properties. Solving Eqs. C and D for all $\mathbf{x}_k \in N_s$: $\lambda_k \neq 0$ and $\lambda_k[y_k(\mathbf{w}^T\mathbf{x_k} + w_b) - 1] = 0$, we have $y_k(\mathbf{w}^T\mathbf{x_k} + w_b) = 1$.
- Given that $y_k \in \{+1, -1\}$, we have $\mathbf{w}^T\mathbf{x_k} + w_b = \pm 1$. Bingo!



Solving hard margin linear SVM

- Remember that KKT condition is a necessary condition, not sufficient condition.
- The SVM problem is a quadratic programming problem. There are many documents on the Internet about solving hard margin linear SVM as a quadratic programming problem. Here is one in MATLAB http://www.robots.ox.ac.uk/~az/lectures/ml/matlab2.pdf. For Python, use the `cvxopt` toolbox. I have some hints here.

Soft margin linear SVM

.4 image

.6

- What if the samples are not linearly separable?
- Let $\xi_k = 0$ for all samples on or inside the correct margin boundary.
- Let $\xi_k = |y_k - (\mathbf{w}^T\mathbf{x}_k + w_b)|$, i.e., the prediction error, for all samples that are misclassified (red in the left figure), where the operator $|\cdot|$ stands for absolute value.
- In this case, we want to maximize the margin but minimize the number of misclassified samples.
- Therefore, we have a new optimization problem:

$$\begin{cases} \min & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{k=1}^{K}\xi_k \\ s.t. & y_k(\mathbf{w}^T\mathbf{x}_k + w_b) \geq 1 - \xi_k, \forall \mathbf{x}_k \\ & \xi_k \geq 0. \end{cases}$$