“...many problems are naturally classification problems”---Prof. Forsyth

Credit: wikipedia

Hongye Liu, Teaching Assistant Prof, CS361, UIUC, 4.8.2021
Last time

- Decision tree (II)
- Random forest
- Support Vector Machine (I)
Objectives

- Support Vector Machine (II)
  - Hinge loss + Regularization
  - Convex function, Gradient Descent
  - Stochastic Gradient Descent
  - Training & Validation

- Naïve Bayesian Classifier
When solving a classification problem, it is good to try several techniques.

Criteria to consider in choosing the classifier include:

- Accuracy
- Training speed
- Classification speed
- Performance with small training set
- Interpretability

Stochastic Gradient Descent
SVM problem formulation

- At first we assume a binary classification problem

- The training set consists of N items
  - Feature vectors \( x_i \) of dimension \( d \)
  - Corresponding class labels \( y_i \in \{-1, 1\} \)

- We can picture the training data as a \( d \)-dimensional scatter plot with colored labels
SVM uses a hyperplane as its decision boundary.

The decision boundary is:

\[ a_1 x^{(1)} + a_2 x^{(2)} + \ldots + a_d x^{(d)} + b = 0 \]

In vector notation, the hyperplane can be written as:

\[ \mathbf{a}^T \mathbf{x} + b = 0 \]
SVM assigns a class label to a feature vector according to the following rule:

+1 if \( \mathbf{a}^T \mathbf{x}_i + b \geq 0 \)

-1 if \( \mathbf{a}^T \mathbf{x}_i + b < 0 \)

In other words, the classification function is: \( \text{sign}(\mathbf{a}^T \mathbf{x}_i + b) \)

Note that

* If \( |\mathbf{a}^T \mathbf{x}_i + b| \) is small, then \( \mathbf{x}_i \) was close to the decision boundary

* If \( |\mathbf{a}^T \mathbf{x}_i + b| \) is large, then \( \mathbf{x}_i \) was far from the decision boundary
What if there is no clean cut boundary?

- Some boundaries are better than others for the training data
- Some boundaries are likely more robust for run-time data
- We need to a quantitative measure to decide about the boundary
- The **loss function** can help decide if one boundary is better than others

$$a^T x + b = 0$$
Loss function 1

- For any given feature vector $\mathbf{x}_i$ with class label $y_i \in \{\pm 1\}$, we want
  - Zero loss if $\mathbf{x}_i$ is classified correctly $\text{sign}(\mathbf{a}^T \mathbf{x}_i + b) = y_i$
  - Positive loss if $\mathbf{x}_i$ is misclassified $\text{sign}(\mathbf{a}^T \mathbf{x}_i + b) \neq y_i$
  - If $\mathbf{x}_i$ is misclassified, more loss is assigned if it’s further away from the boundary

- This loss function 1 meets the criteria above:
  \[
  \max(0, -y_i (\mathbf{a}^T \mathbf{x}_i + b))
  \]

- Training error cost
  \[
  S(\mathbf{a}, b) = \frac{1}{N} \sum_{i=1}^{N} \max(0, -y_i (\mathbf{a}^T \mathbf{x}_i + b))
  \]
Q. What’s the value of this function?

\[ \max(0, -y_i(a^T x_i + b)) \quad \text{if} \quad \text{sign}(a^T x_i + b) = y_i \]

A. 0.
B. others.
Q. What’s the value of this function?

$$\max(0, -y_i(a^T x_i + b)) \quad \text{if} \quad \text{sign}(a^T x_i + b) \neq y_i$$

A. 0.

B. A value greater than or equal to 0.
Loss function 1

For any given feature vector $\mathbf{x}_i$ with class label $y_i \in \{\pm 1\}$, we want

- Zero loss if $\mathbf{x}_i$ is classified correctly $\text{sign}(a^T \mathbf{x}_i + b) = y_i$
- Positive loss if $\mathbf{x}_i$ is misclassified $\text{sign}(a^T \mathbf{x}_i + b) \neq y_i$
- If $\mathbf{x}_i$ is misclassified, more loss is assigned if it’s further away from the boundary

This loss function 1 meets the criteria above:

$$\max(0, -y_i (a^T \mathbf{x}_i + b))$$

Training error cost

$$S(a, b) = \frac{1}{N} \sum_{i=1}^{N} \max(0, -y_i (a^T \mathbf{x}_i + b))$$
The problem with loss function 1

- Loss function 1 does not distinguish between the following decision boundaries if they both classify $\mathbf{x}_i$ correctly.
  - One passes the two classes closely
  - One that passes with a wider margin

- But leaving a larger margin gives robustness for run-time data - the large margin principle

Credit: Kelvin Murphy
Loss function 2: the hinge loss

- We want to impose a small positive loss if \( \mathbf{x}_i \) is correctly classified but close to the boundary.

- The **hinge loss** function meets the criteria above:

\[
    \max(0, 1 - y_i(\mathbf{a}^T \mathbf{x}_i + b))
\]

- Training error cost

\[
    S(\mathbf{a}, b) = \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{a}^T \mathbf{x}_i + b))
\]
The problem with loss function 2

- Loss function 2 favors decision boundaries that have large $\|a\|$ because increasing $\|a\|$ can zero out the loss for a correctly classified $\mathbf{x}_i$ near the boundary.

- But large $\|a\|$ makes the classification function $\text{sign}(\mathbf{a}^T \mathbf{x}_i + b)$ extremely sensitive to small changes in $\mathbf{x}_i$ and make it less robust to run-time data.

- So small $\|a\|$ is better.
Hinge loss with regularization penalty

- We add a penalty on the square magnitude $\|a\|^2 = a^T a$

- Training error cost

$$S(a, b) = \left[ \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i(a^T x_i + b)) \right] + \lambda \left( \frac{a^T a}{2} \right)$$

- The **regularization parameter** $\lambda$ trade off between these two objectives
Q. What does the penalty discourage?

\[ S(a, b) = \left[ \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i(a^T x_i + b)) \right] + \lambda \left( \frac{a^T a}{2} \right) \]

A. Too big a magnitude of the vector \( a \)

B. Too many data points in the training set
How to compute the decision boundary?

\[
\minimize \text{ Loss function } S(\vec{a}, b)
\]

\[
(\vec{a}^*, b^*) = \arg \min_{(\vec{a}, b)} S(\vec{a}, b)
\]

\[
\begin{bmatrix}
\vec{a}_{1}^* \\
\vec{a}_{2}^* \\
\vdots \\
\vec{a}_{d}^*
\end{bmatrix}
\]

\[
\begin{bmatrix}
b^*
\end{bmatrix}
\]
If a set is convex, any line connecting two points in the set is completely included in the set.

A convex function: the area above the curve is convex.

\[ f(\lambda x + (1 - \lambda) y) < \lambda f(x) + (1 - \lambda) f(y) \]
Q. Is this curve a convex curve?

A. YES  
B. NO
Q. Is this curve a convex curve?

A. YES

B. NO
Q. Is this surface convex?

A. **YES**

B. **NO**

Source: wikipedia
Iterative minimization by gradient descent

- For a function such as \( f(a) = a^2 \)
- A convex surface

\[ \hat{a}_i = a_0 + \eta \hat{p} \]
\[ \hat{p} = -\nabla f \]
\[ f \text{ is scalar} \]

Source: wikipedia
Gradient Descent

\( \hat{a} = [a_1, a_2, \ldots, a_d]^T \) let's omit \( b \) for now

Loss function: \( f = f(\hat{a}) \)

\[ \nabla f = \begin{pmatrix}
\frac{\partial f}{\partial a_1} \\
\frac{\partial f}{\partial a_2} \\
\vdots \\
\frac{\partial f}{\partial a_d}
\end{pmatrix} \]

\( a_{n+1} = a_n + \eta \nabla f_n \)

Taylor expansion

\[ f(\hat{a}_{n+1}) = f(\hat{a}_n + \eta \nabla f_n) = f(\hat{a}_n) + \eta (\nabla f)^T \nabla f_n + O(\eta^2) \]

if \( \nabla f_n = -\nabla f(\hat{a}_n) \)

\[ f(\hat{a}_{n+1}) = f(\hat{a}_n) - \eta_n (\nabla f)^T \nabla f = f(\hat{a}_n) - \eta_n \| \nabla f \|^2 \]

\( \eta > 0 \)

\( \eta_n < 1 \)

learning rate or step size
Stochastic gradient descent

\[ f(\tilde{a}) = \frac{1}{k} \sum_{j=1}^{k} Q(\tilde{a}, j) \]

Stochastic gradient descent approximates

\[ f(\tilde{a}) \approx g(\tilde{a}) = \frac{1}{m} \sum_{i=1}^{m} Q(\tilde{a}, i) \]

\( x_k \in \{x_i\} \)

We often choose \( m = 1 \)
The difference btw GD and SGD

**GD**

Loss:

\[ f(\tilde{a}) = \frac{1}{k} \sum_{j=1}^{k} Q(\tilde{a}, j) + \text{penalty} \]

\[ Q = \max(0, 1 - y_j (\tilde{a}^T x_j + b)) \]

\[ \tilde{a}_{n+1} = \tilde{a}_n - \eta \nabla f(\tilde{a}_n) \]

\[ \lim_{n \to \infty} \tilde{a}_n = \text{argmin}_{\tilde{a}} (f(\tilde{a})) \quad \text{if } f \text{ is convex} \]

**SGD**

Loss:

\[ g(\tilde{a}) = \frac{1}{m} \sum_{i=1}^{m} Q(\tilde{a}, i) + \text{penalty} \]

\[ g(\tilde{a}) = f(\tilde{a}) + \varepsilon \quad \text{noise} \]

\[ \tilde{a}_{n+1} = \tilde{a}_n - \eta \nabla g(\tilde{a}_n) \]

\[ \lim_{n \to \infty} E[(\tilde{a}_n - \tilde{a}^*)^2] = 0 \quad \text{given convex loss and other conditions} \]
Update parameters of the hyperplane during the stochastic gradient descent

Since \( S_k(a, b) = \max(0, 1 - y_k(a^T x_k + b)) \) and \( S_0(a, b) = \lambda \left( \frac{a^T a}{2} \right) \)

We have the following updating equations:

\[
S(a, b) = S_k + S_0 \quad \text{if } m = 1
\]

<table>
<thead>
<tr>
<th>If ( y_k(a^T x_k + b) \geq 1 )</th>
<th>If ( y_k(a^T x_k + b) &lt; 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a \leftarrow a - \eta(\lambda a) )</td>
<td>( a \leftarrow a - \eta(\lambda a - y_kx_k) )</td>
</tr>
<tr>
<td>( b \leftarrow b )</td>
<td>( b \leftarrow b - \eta(-y_k) )</td>
</tr>
</tbody>
</table>

We often set \( \eta_n = \frac{1}{n} \)
The training error cost $S(a, b)$ is a function of decision boundary parameters $(a, b)$, so it can help us find the best decision boundary.

- Fix $\lambda$ and set some initial values for $(a, b)$
- Search iteratively for $(a, b)$
- Repeat the previous steps for several values of $\lambda$ and choose the one that gives the decision boundary with best accuracy on a validation data set.
Split the labeled data into **training**, **validation** and **test** sets.

For each choice of $\lambda$, run stochastic gradient descent to find the best decision boundary boundary parameters ($a$, $b$) using the training set.

Choose the best $\lambda$ based on accuracy on the validation set.

Finally evaluate the SVM’s accuracy on the **test** set.

This process avoids overfitting the data.
Extension to multiclass classification

- **All vs. all**
  - Train a separate binary classifier for each pair of classes.
  - To classify, run all classifiers and see which class it will be labeled most with.
  - Computational complexity is quadratic to the number of classes.

- **One vs. all**
  - Train a separate binary classifier for each class against all else.
  - To classify, run all classifiers and see which label gets the highest score.
  - Computational complexity scales linearly.
What if the data is inseparable linearly?

- There is a chance the data is inseparable
- Use the non-linear **SVM with kernels**!
- Decision boundary is curved
Naïve Bayes classifier

Training

- Use the training data \( \{(x_i, y_i)\} \) to estimate a probability model \( P(y|x) \)
- Assume that the features of \( \{x\} \) are conditionally independent given the class label \( y \)

\[
P(x|y) = \prod_{j=1}^{d} P(x^{(j)}|y)
\]

Classification

- Assign the label \( \arg\max_y P(y|x) \) to a feature vector \( x \)

\[
P(x_{1D} n \ 7 \ 1D n \ 71D) = P(x_{1D}) P(Y_{1D}) P(\geq 1D)
\]
Additional References

- Kelvin Murphy, “Machine learning, A Probabilistic perspective”
See you next time

See You!