Recap

• (Ch 11) Learning to classify
  • Evaluating a classifier
  • Nearest neighbors classifier
  • Naïve Bayes classifier

Today

• (Ch 11) Learning to classify
  • Support vector machine (SVM) classifier
Learning to classify

Given a set of feature vectors $x_i$, where each has a class label $y_i$, we want to train a classifier that can map unlabeled vectors to labels.
How do we know if our classifier is good?

• We want the classifier to avoid making classification mistakes on unlabeled data that we will only see at run-time

• Problem 1: some mistakes may be more costly than others
   
   We can tabulate different types of error and define a loss function

• Problem 2: we will never know the true labels of the run-time data
   
   We must separate the labeled data into a training set and a validation/test set
Support vector machine (SVM) overview

• Decision boundary and classification function

• Loss function

• Training

• Validation/testing

• Extension to multiclass classification
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 \( y_i \in \{\pm 1\} \)

- We can picture the training data as a \( d \)-dimensional scatter plot with colored labels
Decision boundary

- An SVM uses a hyperplane as its **decision boundary**

- Mathematically, the decision boundary is

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

- In vector notation, the decision boundary is

\[ \mathbf{a}^T \mathbf{x} + b = 0 \]
Classification function (or prediction function)

• The SVM assigns a label to a feature vector $\mathbf{x}_i$ according to this rule

$$+1 \quad \text{if} \quad \mathbf{a}^T \mathbf{x}_i + b \geq 0$$
$$-1 \quad \text{if} \quad \mathbf{a}^T \mathbf{x}_i + b < 0$$

• 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’s no clean decision boundary?

• Some boundaries are still better than others on the training data

• Some boundaries are more likely to be robust to as-yet-unseen run-time data

• We will develop loss functions that tell us if one decision boundary is better than another
Loss function 1

• For any given feature vector $x_i$ with class label $y_i \in \{\pm 1\}$, we want
  
  • Zero loss if $x_i$ is classified correctly: $\text{sign}(a^T x_i + b) = y_i$
  • Positive loss if $x_i$ is misclassified: $\text{sign}(a^T x_i + b) \neq y_i$
  • If $x_i$ is misclassified, more loss the further away it is from the boundary

• Loss function 1 meets the requirements: $\max(0, -y_i(a^T x_i + b))$

• Training error cost

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

- Loss function 1 does not distinguish between the following decision boundaries if they both classify $x_i$ correctly
  - One that passes close to $x_i$
  - One that leaves a wide margin

- But leaving a margin gives robustness to run-time data
Loss function 2: hinge loss

- We want to impose a small positive loss if $x_i$ is correctly classified but close to the boundary.

- The **hinge loss** meets the requirements: $\max(0, 1 - y_i(a^T x_i + b))$.

- Training error cost

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

• Loss function 2 favors decision boundaries with large ||a|| because increasing ||a|| can artificially zero-out the loss for a correctly classified $x_i$ near the boundary

• But large ||a|| makes the classification function $\text{sign}(a^Tx_i + b)$ extremely sensitive to small changes in $x_i$, which is bad for robustness 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 $\lambda$ trades off between the two objectives
Training procedure

The training error cost \( S(a, b) \) is a function of the decision boundary parameters \((a, b)\), so it can help us find the best decision boundary

- Fix \( \lambda \) and set some initial values of \((a, b)\)

- Search iteratively for \((a, b)\) that minimize \( S(a, b) \) on the training set

- Repeat the previous steps for several values of \( \lambda \) and choose the one that gives the decision boundary with best accuracy on a validation set
Iterative minimization by gradient descent

- For simplicity, we write

\[ S(a, b) = \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i(a^T x_i + b)) + \lambda \left( \frac{a^T a}{2} \right) = \frac{1}{N} \sum_{i=1}^{N} S_i(a, b) + S_0(a, b) \]

- The direction of steepest descent is

\[ -\nabla S(a, b) = -\frac{1}{N} \sum_{i=1}^{N} \nabla S_i(a, b) - \nabla S_0(a, b) \]

Source: Chris Fregly
Stochastic gradient descent

• The exact $-\nabla S(a, b)$ is too cumbersome to evaluate for large $N$, so we approximate in each step by randomly sampling a single vector $x_k \in \{x_i\}$ with replacement

  Direction of descent: $-\nabla S(a, b) \approx -\nabla S_k(a, b) - \nabla S_0(a, b)$

• Recall that

  $$\nabla S_k(a, b) = \begin{bmatrix}
  \frac{\partial S_k}{\partial a_1} \\
  \vdots \\
  \frac{\partial S_k}{\partial a_1} \\
  \frac{\partial S_k}{\partial b}
\end{bmatrix}$$

Source: Chris Fregly
Update equations for parameters \((a, b)\)

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 update equations:

<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_k x_k))</td>
</tr>
<tr>
<td>(b \leftarrow b)</td>
<td>(b \leftarrow b - \eta(y_k))</td>
</tr>
</tbody>
</table>

An **epoch** is a certain number of steps in the descent (usually set to be approximately the size \(N\) of the training set).

In the \(e\)th epoch, it is common to set the **steplength** \(\eta = \frac{m}{e+n}\)

where \(m\) and \(n\) are constants chosen by experiment.
Validation/testing

• 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 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 $(a, b)$ and $\lambda$
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 label is chosen most
  • Computational complexity scales quadratically with the number of classes

• One vs. all
  • Train a separate binary classifier for each class against all other classes
  • To classify, run all classifiers and see which class label gets the highest score
  • Computational complexity scales linearly with the number of classes

• Both approaches can work well in practice