Decision Theory and Simple Classifiers

26 September 2017
Today’s lecture

- Decision theory

- Linear and quadratic classifiers
  - Gaussian models, the perceptron
Classification

• In detection we had one template
  • Decision: was it there?
    • Or rather, how much of it is there?

• In classification we have many templates
  • Decision: Which one is the most dominant?
Process overview

- We provide examples of classes
  - Training data

- We make models of each class
  - Training process

- We assign all new input data to a class
  - Classification
Making an assignment decision

• Face example
  • Dot products relate to a likelihood of match
  • This is linked to a probability

• Having a class probability for each face, how do we make a decision?
Motivating example

• Which do we pick?

\[
\begin{align*}
x &= \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\
y &= \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \\
x^\top y &= \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix} \\
\end{align*}
\]

Template face 1

Template face 2

Unknown face

0.94

0.87
Motivating example

- Template 1 is more “likely”

\[
\begin{align*}
\text{Likelihood} & \propto 0.94 \\
\text{Likelihood} & \propto 0.87
\end{align*}
\]
How the decision is made

- In simple cases the answer is intuitive

- To get a complete picture we need to probe a bit deeper

- Bayesian decision theory
Starting simple

- Two-class case, $\omega_1$ and $\omega_2$
- 1-dimensional data
Starting simple

• Given a sample $x$, is it $\omega_1$ or $\omega_2$?
• i.e. $P(\omega_i \mid x) =$ ?
Getting the answer

• The *class posterior probability* is:

\[
P(\omega_i | x) = \frac{P(x | \omega_i)P(\omega_i)}{P(x)}
\]

• To find the answer we need to fill in the terms in the right-hand-side
Filling the unknowns

- **Class priors**
  - How much of each class?
    
    \[
P(\omega_1) \approx \frac{N_1}{N} \]
    \[
P(\omega_2) \approx \frac{N_2}{N} \]

- **Class likelihood**: \( P(x | \omega_i) \)
  - Requires that we have a model of each \( \omega_i \)
  - E.g. \( \omega_i \) can be a Gaussian distributed so that:
    
    \[
P(x | \omega_i) = \mathcal{N} \left( x | \mu_{\omega_i}, \Sigma_{\omega_i} \right) \]
Filling the unknowns

- Evidence:

\[ P(x) = P(x \mid \omega_1)P(\omega_1) + P(x \mid \omega_2)P(\omega_2) \]

- We now can estimate the class posteriors:

\[ P(\omega_1 \mid x), P(\omega_2 \mid x) \]
Making the decision

• Bayes classification rule:
  If \( P(\omega_1 \mid x) > P(\omega_2 \mid x) \) then \( x \) belongs to class \( \omega_1 \)
  If \( P(\omega_1 \mid x) < P(\omega_2 \mid x) \) then \( x \) belongs to class \( \omega_2 \)

• Easier version:
  \[ P(x \mid \omega_1)P(\omega_1) \geq P(x \mid \omega_2)P(\omega_2) \]

• Equiprobable class version:
  \[ P(x \mid \omega_1) \geq P(x \mid \omega_2) \]
Visualizing the decision

- Assume a Gaussian model for the classes
  - Likelihood: $P(x \mid \omega_i) = \mathcal{N}(x \mid \mu_i, \sigma_i)$
Errors in classification

- We can’t win all the time though
  - Some inputs will be misclassified

- What are the errors?

\[
\varepsilon_2 = \int_{-\infty}^{x_0} P(x | \omega_2)P(\omega_2)dx, \quad \varepsilon_1 = \int_{x_0}^{\infty} P(x | \omega_1)P(\omega_1)dx
\]
Minimizing misclassifications

• The Bayes classification rule minimizes these potential misclassifications

• Can you do any better by moving the line?
Minimizing risk

• Not all errors are equal!
  • e.g. medical diagnoses

• Misclassification can be tolerable, or not, depending on the assumed risks
Adding a cost term

- Implement a “loss” factor to each decision to compute “risk” for each class

\[ r_j = \sum_{i} \lambda_{ji} \int_{R_i} P(x | \omega_j) dx \]

- The \( \lambda \)'s specify how costly each decision is
  - \( \lambda_{ji} \) is cost for samples in region \( i \) while being of class \( j \)

- Choose regions to minimize overall risk

\[ r = \sum_{j} r_j P(\omega_j) = \sum_{i} \int_{R_i} \sum_{j} \lambda_{ji} P(x | \omega_j) P(\omega_j) dx \]
New decision process

• **Assign** \( x \) to \( \omega_1 \) if

\[
(\lambda_{21} - \lambda_{22})P(x \mid \omega_2)P(\omega_2) < (\lambda_{12} - \lambda_{11})P(x \mid \omega_1)P(\omega_1)
\]

- and vice-versa

• **Or using the likelihood ratio test:**

\[
x \in \omega_1 \quad \text{if} \quad \frac{P(x \mid \omega_1)}{P(x \mid \omega_2)} > \frac{P(\omega_2)(\lambda_{21} - \lambda_{22})}{P(\omega_1)(\lambda_{12} - \lambda_{11})}
\]

- and vice-versa
Example case

- A deadly disease detector
- Optimizing the cost of each decision; note that $\lambda_{21} \gg \lambda_{12}$

Minimum classification error

- $\lambda_{11} = 0$
- $\lambda_{21} = \text{big lawsuit!}$
- $\lambda_{12} = \text{unnecessary extra tests}$
- $\lambda_{22} = \text{treatment}$

Minimum risk

- $\lambda_{11} = 0$
- $\lambda_{21} = \text{big lawsuit!}$
- $\lambda_{12} = \text{unnecessary extra tests}$
- $\lambda_{22} = \text{treatment}$
True/False – Positives/Negatives

• Naming the outcomes

<table>
<thead>
<tr>
<th>Classifying for $\omega_1$</th>
<th>$x$ is $\omega_1$</th>
<th>$x$ is $\omega_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$ classified as $\omega_1$</td>
<td>True positive</td>
<td>False positive</td>
</tr>
<tr>
<td>$x$ classified as $\omega_2$</td>
<td>False negative</td>
<td>True negative</td>
</tr>
</tbody>
</table>

• False positive / False alarm / Type I error
• False negative / Miss / Type II error
Receiver Operating Characteristic

- Visualizing how well we can expect to do
Classifying Gaussian data

- Remember that we need the class likelihood to make a decision
- For now let’s assume that:
  \[ P(x | \omega_i) = \mathcal{N}(x | \mu_i, \sigma_i) \]
- i.e. that the input data is Gaussian distributed
Overall methodology

• Obtain training data

• Fit a Gaussian model to each class
  • Perform parameter estimation for mean, variance and class priors

• Define decision regions based on models and any given constraints
1D example

- The decision boundary will always be a point separating the two class regions.
2D example
2D example fitted Gaussians
Gaussian decision boundaries

- The decision boundary is defined as:

\[ P(x \mid \omega_1)P(\omega_1) = P(x \mid \omega_2)P(\omega_2) \]

- Replace likelihoods with Gaussians and solve to find what the boundary looks like
Discriminant functions

• Define a set of functions $g_i(x)$ so that:

  classify $x$ in $\omega_i$ if $g_i(x) > g_j(x)$, $\forall i \neq j$

• Decision boundaries are now defined as:

  $$g_{ij}(x) \equiv \left( g_i(x) = g_j(x) \right)$$
Discriminant functions for Gaussians

- We remove the exponentiation:

\[
g_i(x) = \log(P(x | \omega_i)P(\omega_i)) = \log P(x | \omega_i) + \log P(\omega_i)
\]

\[
= -\frac{1}{2}(x - \mu_i)\Sigma_i^{-1} \cdot (x - \mu_i) + \log P(\omega_i) + C_i
\]

\[
= \frac{1}{2} \left[ -x^\top \cdot \Sigma_i^{-1} \cdot x + x^\top \cdot \Sigma_i^{-1} \cdot \mu_i - \mu_i^\top \cdot \Sigma_i^{-1} \cdot \mu_i + \mu_i^\top \cdot \Sigma_i^{-1} \cdot x \right] 
\]

\[ + \log P(\omega_i) + C_i \]

- The decision boundaries \( g_i(x) = g_j(x) \) will be quadrics
Back to the data

- $\Sigma_i = \sigma^2 I$ produces line boundaries

- Discriminant:

\[
g_i(x) = w_i^T \cdot x + b
\]
\[
w_i = \mu_i / \sigma^2
\]
\[
b = -\frac{\mu_i^T \cdot \mu_i}{2\sigma^2} + \log P(\omega_i)
\]
Quadratic boundaries

- Arbitary covariance matrices can produce more elaborate boundaries

\[ g_i(x) = x^\top \cdot W_i \cdot x + w_i^\top \cdot x + w_i \]

\[ W_i = -\frac{1}{2} \Sigma_i^{-1} \]

\[ w_i = \Sigma_i^{-1} \cdot \mu_i \]

\[ w = -\frac{1}{2} \mu_i^\top \cdot \Sigma_i^{-1} \cdot \mu_i - \frac{1}{2} \log |\Sigma_i| \]

\[ + \log P(\omega_i) \]
Quadratic boundaries

- Arbitrary covariance matrices can produce more elaborate boundaries
Naïve Bayes classifier

• Dimensionality issues
  • For large dimensions the Gaussian estimate will require a lot of data! Order $N$ dimensions

• Naïve Bayes classifier assumes independence across dimensions
Naïve Bayes classifier

- Each dimension is sampled independently
  - Thus we don’t require many training samples
    \[ P(x | \omega_i) = \prod_j P(x_j | \omega_i) \]
- Overall classification is:
  \[ \omega = \arg \max_{\omega_i} \prod_j P(x_j | \omega_i) \]
  - Not elegant, but reasonably reliable
    - Looks familiar?
A different perspective

- Obtaining the discriminant function directly
  - Training data \( x_i \), labels \( y_i \in \{-1,1\} \)
  - Linear discriminant \( \mathbf{w} \) and bias \( b \)

\[ y_i = \mathbf{w}^\top \cdot \mathbf{x}_i + b \]

- Or we can skip the \( b \) and set \( \mathbf{x} = [\mathbf{x} ; 1] \) in which case \( \mathbf{w} = [\mathbf{w}, b] \)
- This is the same as the discriminant function for isotropic Gaussians
Linear classifiers

- Directly defines the class boundary line

$$w^T \cdot x = 0$$
Approach 1: The Perceptron

- Assume there is a solution
- Then find $w$ such that:
  
  \[
  w^\top \cdot x_i > 0 \quad \text{if} \quad x_i \in \omega_1
  \]
  \[
  w^\top \cdot x_i < 0 \quad \text{if} \quad x_i \in \omega_2
  \]

- How do we solve this?
A simple update algorithm

- Using corrections

- For all vectors $\mathbf{x}$:
  - If $\text{sgn}(\mathbf{w}^\top \cdot \mathbf{x}_i) = y_i$
    - Do nothing
  - If $\text{sgn}(\mathbf{w}^\top \cdot \mathbf{x}_i) = -y_i$
    - Then $\mathbf{w} = \mathbf{w} + \eta y_i \mathbf{x}_i$, $0 < \eta < 1$
  - Repeat until no error (or progress) is made
What does this mean?

- **w** is normal to the boundary line
- **To produce** $w^\top \cdot x_i > 0$
  - Has to be within 90° of positive data
- **To produce** $w^\top \cdot x_i < 0$
  - Has to be outside of 90° for negative data

$$w^\top \cdot x_i > 0$$

$$w^\top \cdot x_i < 0$$
Looking at one class

1. Initial conditions

2. Correction with $x_1$

3. Correction with $x_3$

4. Correction with $x_1$
Approach 2: Using a cost function

- Minimize the cost function:

\[ J(w) = \sum_{\forall \text{sgn}(w^\top x_i) \neq y_i} \delta_i w^\top \cdot x_i \]

\[ \delta_i = -y_i = \begin{cases} 
-1, & \text{if } x_i \in \omega_1 \\
+1, & \text{if } x_i \in \omega_2 
\end{cases} \]

- Cost function will be zero if all classifications are correct

\[
J(w) = \sum_{\forall \text{sgn}(w^\top x_i) \neq y_i} \delta_i w^\top \cdot x_i
\]

\[
\begin{array}{c|c|c}
\text{Contribution to cost function} & x_i \in \omega_1 & x_i \in \omega_2 \\
\hline
\delta_i = -1 & 0 & \delta_i w^\top \cdot x_i > 0 \\
\hline
\delta_i = +1 & 0 & 0
\end{array}
\]

\[
\begin{array}{c|c|c|c}
\text{Contribution to cost function} & w^\top \cdot x_i > 0 & \delta_i w^\top \cdot x_i > 0 \\
\hline
w^\top \cdot x_i < 0 & \delta_i = -1 & 0
\end{array}
\]
Gradient descent approach

- Use a gradient descend algorithm:

\[ w = w - \eta \sum_{\forall \text{sgn}(w^T x_i) \neq y_i} \delta_i x_i \]

- Same as the perceptron!
- Proven convergence, fast and small!
  - Many variants exist
  - A core idea behind neural nets
Approach 3: Minimize Squared Error

• We can also directly solve the problem directly
  • Assign all samples in a matrix \( X \)
  • Assign all class labels in vector \( y \)

• Solve for \( w \) such that:

\[
y = w^\top \cdot X
\]
MSE classifier

• Solving for $w$ we get:

$$w = y \cdot X^+$$

• We only need the pseudoinverse of $X$
  • Robust closed form solution (if $X$ is full-rank)
• This is essentially a regression problem
  • Least-squares solution
Linear classifiers

• As long as we use a Euclidean distance metric, there is a Gaussian assumption

• Linear classifiers are easier to understand
  • But remember what they actually imply!
Looking back on detection

- We can now make detection more elegant
  - No need to look at correlation peaks

- Learn class models from examples
  - Doesn’t have to be a single template

- Translate the dot products to likelihoods
  - Apply decision theory to classify in either of the two classes
Example: Big smile detector

• Make a classifier that find smiling faces

Some non-smiling samples

Some smiling samples
Simple way

- Assume classes are Gaussian distributed (are they?)
- Get means and covariances for each class
Standard ways to classify

- Assume a linear classifier (isotropic Gaussian):
  \[
P(x \mid \omega_i) = \mathcal{N}(x \mid \mu_i, I) \propto e^{-(x-\mu_i)^\top(x-\mu_i)}
  \]
  - Assume equal priors and assign data to maximum likelihood class
  - Get’s us okay results most of the time

- Assume a quadratic classifier (full or diagonal covariance)
  \[
P(x \mid \omega_i) = \mathcal{N}(x \mid \mu_i, \Sigma_i) \propto e^{-(x-\mu_i)^\top \Sigma_i^{-1} (x-\mu_i)}
  \]
  - Careful, the covariance might be non-invertible
Or we can use features

- Perform PCA on the faces and drop to low dimensions
- Now the covariance estimate is better behaved
Redo the classification on the low dims

- Faster and easier, and just as good
  - Not crucial in this case, but the quadratic classifier is a bit better

- How can we improve this? Is it worth it?
Recap

• The Bayesian view
  • Risk, decision regions
  • Gaussian classifiers, Naïve Bayes

• Linear classifiers
  • The perceptron, separating hyperplanes
Next lecture

- Support Vector Machines
- Non-linear classifiers
  - Neural nets and Kernels
Reading

- Textbook chapters 2-2.4, 2.5.7 and 3-3.6
Problem set 2

• It’s out, in case you missed it ...