An Introduction to Machine Learning

L2: Instance Based Estimation

Alexander J. Smola

Statistical Machine Learning Program
Canberra, ACT 0200 Australia
Alex.Smola@nicta.com.au

Tata Institute, Pune, January 2007
L1: Machine learning and probability theory
Introduction to pattern recognition, classification, regression, novelty detection, probability theory, Bayes rule, inference

L2: Density estimation and Parzen windows
Nearest Neighbor, Kernels density estimation, Silverman’s rule, Watson Nadaraya estimator, crossvalidation

L3: Perceptron and Kernels
Hebb’s rule, perceptron algorithm, convergence, feature maps, kernels

L4: Support Vector estimation
Geometrical view, dual problem, convex optimization, kernels

L5: Support Vector estimation
Regression, Quantile regression, Novelty detection, $\nu$-trick

L6: Structured Estimation
Sequence annotation, web page ranking, path planning, implementation and optimization
L2 Instance Based Methods

Nearest Neighbor Rules

Density estimation
- empirical frequency, bin counting
- priors and Laplace rule

Parzen windows
- Smoothing out the estimates
- Examples

Adjusting parameters
- Cross validation
- Silverman’s rule

Classification and regression with Parzen windows
- Watson-Nadaraya estimator
Nearest Neighbor Rule

Goal
Given some data $x_i$, want to classify using class label $y_i$.

Solution
Use the label of the nearest neighbor.

Modified Solution (classification)
Use the label of the majority of the $k$ nearest neighbors.

Modified Solution (regression)
Use the value of the average of the $k$ nearest neighbors.

Key Benefits
- Basic algorithm is very simple.
- Can use arbitrary similarity measures
- Will eventually converge to the best possible result.

Problems
- Slow and inefficient when we have lots of data.
- Not very smooth estimates.
Nearest Neighbor Classifier

```python
from pylab import *
from numpy import *

... load data ...

xnorm = sum(x**2)
xtestnorm = sum(xtest**2)

dists = (-2.0*dot(x.transpose(), xtest) + xtestnorm).transpose() + xnorm

labelindex = dists.argmin(axis=1)
```

\(k\)-Nearest Neighbor Classifier

```python
sortargs = dists.argsort(axis=1)
k = 7
ytest = sign(mean(y[sortargs[:,0:k]], axis=1))
```

Nearest Neighbor Regression

just drop \texttt{sign(...)}
7 Nearest Neighbors
7 Nearest Neighbors
Regression Problem
Nearest Neighbor Regression
7 Nearest Neighbors Regression
Mini Summary

Nearest Neighbor Rule
Predict same label as nearest neighbor

$k$-Nearest Neighbor Rule
Average estimates over $k$ neighbors

Details
- Easy to implement
- No training required
- Slow if lots of training data
- Not so great performance
Tossing a dice (again)
Big Problem
Only sampling *many times* gets the parameters right.

Rule of Thumb
We need at least 10-20 times as many observations.

Conjugate Priors
Often we know what we should expect. Using a conjugate prior helps. We *insert fake additional data* which we assume that it comes from the prior.

Conjugate Prior for Discrete Distributions
- Assume we see $u_i$ additional observations of class $i$.

$$
\pi_i = \frac{\text{#occurrences of } i + u_i}{\text{#trials} + \sum_j u_j}.
$$

- Assuming that the dice is even, set $u_i = m_0$ for all $1 \leq i \leq 6$. For $u_i = 1$ this is the Laplace Rule.
Example: Dice

20 tosses of a dice

<table>
<thead>
<tr>
<th>Outcome</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counts</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>MLE</td>
<td>0.15</td>
<td>0.30</td>
<td>0.10</td>
<td>0.05</td>
<td>0.20</td>
<td>0.20</td>
</tr>
<tr>
<td>MAP ((m_0 = 6))</td>
<td>0.25</td>
<td>0.27</td>
<td>0.12</td>
<td>0.08</td>
<td>0.19</td>
<td>0.19</td>
</tr>
<tr>
<td>MAP ((m_0 = 100))</td>
<td>0.16</td>
<td>0.19</td>
<td>0.16</td>
<td>0.15</td>
<td>0.17</td>
<td>0.17</td>
</tr>
</tbody>
</table>

Consequences

- Stronger prior brings the estimate closer to uniform distribution.
- More robust against outliers
- **But:** Need more data to detect deviations from prior
Correct dice
Mini Summary

Maximum Likelihood Solution
- Count number of observations per event
- Set probability to empirical frequency of occurrence.

Maximum a Posteriori Solution
- We have a good guess about solution
- Use conjugate prior
- Corresponds to inventing extra data
- Set probability to take additional observations into account

Extension
- Works also for other estimates, such as means and covariance matrices.
Density Estimation

Data
Continuous valued random variables.

Naive Solution
Apply the bin-counting strategy to the continuum. That is, we discretize the domain into bins.

Problems
- We need lots of data to fill the bins
- In more than one dimension the number of bins grows exponentially:
  - Assume 10 bins per dimension, so we have 10 in $\mathbb{R}^1$
  - 100 bins in $\mathbb{R}^2$
  - $10^{10}$ bins (10 billion bins) in $\mathbb{R}^{10}$ . . .
Mixture Density
Sampling from $\rho(x)$
Bin counting
Parzen Windows

Naive approach

Use the empirical density

\[ p_{\text{emp}}(x) = \frac{1}{m} \sum_{i=1}^{m} \delta(x, x_i). \]

which has a delta peak for every observation.

Problem

What happens when we see slightly different data?

Idea

Smear out \( p_{\text{emp}} \) by convolving it with a kernel \( k(x, x') \). Here \( k(x, x') \) satisfies

\[ \int_{\mathcal{X}} k(x, x') dx' = 1 \text{ for all } x \in \mathcal{X}. \]
Parzen Windows

Estimation Formula
Smooth out $p_{\text{emp}}$ by convolving it with a kernel $k(x, x')$.

$$p(x) = \frac{1}{m} \sum_{i=1}^{m} k(x_i, x)$$

Adjusting the kernel width
- Range of data should be adjustable
- Use kernel function $k(x, x')$ which is a proper kernel.
- Scale kernel by radius $r$. This yields

$$k_r(x, x') := r^n k(rx, rx')$$

Here $n$ is the dimensionality of $x$. 
Discrete Density Estimate
Smoothing Function
Density Estimate

The graph shows two curves:
- The red curve represents the true density.
- The blue curve is an estimate of the true density.

The x-axis represents the range from -10 to 10, and the y-axis represents the density values from 0 to 0.35.
Examples of Kernels

Gaussian Kernel

\[ k(x, x') = (2\pi\sigma^2)^{\frac{n}{2}} \exp \left( -\frac{1}{2\sigma^2} \| x - x' \|^2 \right) \]

Laplacian Kernel

\[ k(x, x') = \lambda^n 2^{-n} \exp (-\lambda \| x - x' \|_1) \]

Indicator Kernel

\[ k(x, x') = 1_{[-0.5,0.5]}(x - x') \]

Important Issue

**Width** of the kernel is usually much more important than type.
Laplacian Kernel
Indicator Kernel
Gaussian Kernel with width $\sigma = 1$
Laplacian Kernel

Laplacian Kernel with width $\lambda = 1$
Laplacian Kernel with width $\lambda = 10$
Selecting the Kernel Width

Goal
We need a method for adjusting the kernel width.

Problem
The likelihood keeps on increasing as we narrow the kernels.

Reason
The likelihood estimate we see is distorted (we are being overly optimistic through optimizing the parameters).

Possible Solution
Check the performance of the density estimate on an unseen part of the data. This can be done e.g. by
- Leave-one-out crossvalidation
- Ten-fold crossvalidation
What we really want

- A parameter such that in expectation the likelihood of the data is maximized

\[
p_r(X) = \prod_{i=1}^{m} p_r(x_i)
\]

or equivalently

\[
\frac{1}{m} \log p_r(X) = \frac{1}{m} \sum_{i=1}^{m} \log p_r(x_i).
\]

- However, if we optimize \(r\) for the seen data, we will always overestimate the likelihood.

Solution: Crossvalidation

- Test on unseen data
- Remove a fraction of data from \(X\), say \(X'\), estimate using \(X \setminus X'\) and test on \(X'\).
Crossvalidation Details

Basic Idea
Compute \( p(X'|\theta(X\setminus X')) \) for various subsets of \( X \) and average over the corresponding log-likelihoods.

Practical Implementation
Generate subsets \( X_i \subset X \) and compute the log-likelihood estimate

\[
\frac{1}{n} \sum_{i}^{n} \frac{1}{|X_i|} \log p(X_i|\theta(X|\setminus X_i))
\]

Pick the parameter which maximizes the above estimate.

Special Case: Leave-one-out Crossvalidation

\[
p_{X\setminus x_i}(x_i) = \frac{m}{m-1} p_x(x_i) - \frac{1}{m-1} k(x_i, x_i)
\]
Cross Validation

![Graph showing the relationship between leave-one-out score and kernel width.]
Best Fit ($\lambda = 1.9$)
Mini Summary

Discrete Density
- Bin counting
- Problems for continuous variables
- Really big problems for variables in high dimensions (curse of dimensionality)

Parzen Windows
- Smooth out discrete density estimate.
- Smoothing kernel integrates to 1 (allows for similar observations to have some weight).
- Density estimate is average over kernel functions
- Scale kernel to accommodate spacing of data

Tuning it
- Cross validation
- Expected log-likelihood
**Goal**
Find the least likely observations \( x_i \) from a dataset \( X \). Alternatively, identify low-density regions, given \( X \).

**Idea**
Perform density estimate \( p_X(x) \) and declare all \( x_i \) with \( p_X(x_i) < p_0 \) as novel.

**Algorithm**
Simply compute \( f(x_i) = \sum_j k(x_i, x_j) \) for all \( i \) and sort according to their magnitude.
Applications

Network Intrusion Detection
Detect whether someone is trying to hack the network, downloading tons of MP3s, or doing anything else unusual on the network.

Jet Engine Failure Detection
You can’t destroy jet engines just to see how they fail.

Database Cleaning
We want to find out whether someone stored bogus information in a database (typos, etc.), mislabelled digits, ugly digits, bad photographs in an electronic album.

Fraud Detection
Credit Cards, Telephone Bills, Medical Records

Self calibrating alarm devices
Car alarms (adjusts itself to where the car is parked), home alarm (furniture, temperature, windows, etc.)
Typical Data

3 4 8 6 1 1 3 6
0 0 4 7 1 4 4 2
6 0 4 3 3 7 4 1
3 5 0 0 2 1 0 0
1 7 8 0 6 0 0
Silverman’s Automatic Adjustment

Problem
One ’width fits all’ does not work well whenever we have regions of high and of low density.

Idea
Adjust width such that neighbors of a point are included in the kernel at a point. More specifically, adjust range $h_i$ to yield

$$h_i = \frac{r}{k} \sum_{x_j \in \text{NN}(x_i, k)} \| x_j - x_i \|$$

where $\text{NN}(x_i, k)$ is the set of $k$ nearest neighbors of $x_i$ and $r$ is typically chosen to be 0.5.

Result
State of the art density estimator, regression estimator and classifier.
Sampling from $\rho(x)$
Uneven Scales
Neighborhood Scales
Adjusted Width
Watson-Nadaraya Estimator

**Goal**
Given pairs of observations \((x_i, y_i)\) with \(y_i \in \{\pm 1\}\) find estimator for conditional probability \(\Pr(y|x)\).

**Idea**
Use definition \(p(x, y) = p(y|x)p(x)\) and estimate both \(p(x)\) and \(p(x, y)\) using Parzen windows. Using Bayes rule this yields

\[
\Pr(y = 1|x) = \frac{P(y = 1, x)}{P(x)} = \frac{m^{-1} \sum_{y_i=1} k(x_i, x)}{m^{-1} \sum_i k(x_i, x)}
\]

**Bayes optimal decision**
We want to classify \(y = 1\) for \(\Pr(y = 1|x) > 0.5\). This is equivalent to checking the sign of

\[
\Pr(y = 1|x) - \Pr(y = -1|x) \propto \sum_i y_i k(x_i, x)
\]
# Kernel function
import elefant.kernels.vector
k = elefant.kernels.vector.CGaussKernel(1)

# Compute difference between densities
ytest = k.Expand(xtest, x, y)

# Compute density estimate (up to scalar)
density = k.Expand(xtest, x, ones(x.shape[0]))
Parzen Windows Classifier
Parzen Windows Density Estimate
Decision Boundary

Picking \( y = 1 \) or \( y = -1 \) depends on the sign of

\[
Pr(y = 1| x) - Pr(y = -1| x) = \frac{\sum_i y_i k(x_i, x)}{\sum_i k(x_i, x)}
\]

Extension to Regression

- Use the same equation for regression. This means that

\[
f(x) = \frac{\sum_i y_i k(x_i, x)}{\sum_i k(x_i, x)}
\]

where now \( y_i \in \mathbb{R} \).

- We get a locally weighted version of the data
Regression Problem
Watson Nadaraya Regression
Novelty Detection
- Observations in low-density regions are special (outliers).
- Applications to database cleaning, network security, etc.

Adaptive Kernel Width (Silverman’s Trick)
- Kernels wide wherever we have low density

Watson Nadaraya Estimator
- Conditional density estimate
- Difference between class means (in feature space)
- Same expression works for regression, too
Density estimation
- empirical frequency, bin counting
- priors and Laplace rule

Parzen windows
- Smoothing out the estimates
- Examples

Adjusting parameters
- Cross validation
- Silverman’s rule

Classification and regression with Parzen windows
- Watson-Nadaraya estimator
- Nearest neighbor classifier