Classification

How to predict a discrete variable?

Based on Parishit Ram’s slides. Pari now at SkyTree. Graduated from PhD from GT. Also based on Alex Gray’s slides.
How will I rate "Chopin's 5th Symphony"?

<table>
<thead>
<tr>
<th>Songs</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some nights</td>
<td>😊</td>
</tr>
<tr>
<td>Skyfall</td>
<td>😞</td>
</tr>
<tr>
<td>Comfortably numb</td>
<td>😞</td>
</tr>
<tr>
<td>We are young</td>
<td>😊</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Chopin's 5th</td>
<td>???</td>
</tr>
</tbody>
</table>
Classification

What tools do you need for classification?

1. **Data** $S = \{(x_i, y_i)\}_{i=1,...,n}$
   - $x_i$ represents each example with $d$ attributes
   - $y_i$ represents the label of each example

2. **Classification model** $f_{(a,b,c,....)}$ with some parameters $a, b, c, ...$
   - A model/function maps examples to labels

3. **Loss function** $L(y, f(x))$
   - How to penalize mistakes
### Features

\[ x_i = (x_{i1}, \ldots, x_{id}) \]

<table>
<thead>
<tr>
<th>Song name</th>
<th>Label</th>
<th>Artist</th>
<th>Length</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some nights</td>
<td>![Happy Face]</td>
<td>Fun</td>
<td>4:23</td>
<td>...</td>
</tr>
<tr>
<td>Skyfall</td>
<td>![Sad Face]</td>
<td>Adele</td>
<td>4:00</td>
<td>...</td>
</tr>
<tr>
<td>Comf. numb</td>
<td>![Neutral Face]</td>
<td>Pink Fl.</td>
<td>6:13</td>
<td>...</td>
</tr>
<tr>
<td>We are young</td>
<td>![Happy Face]</td>
<td>Fun</td>
<td>3:50</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Chopin's 5th</td>
<td>??</td>
<td>Chopin</td>
<td>5:32</td>
<td>...</td>
</tr>
</tbody>
</table>
Training a classifier (building the “model”)

Q: How do you learn appropriate values for parameters \(a, b, c, \ldots\) such that

- (Part I) \(y_i = f_{(a,b,c,\ldots)}(x_i), i = 1, \ldots, n\)
  - Low/no error on the training set
- (Part II) \(y = f_{(a,b,c,\ldots)}(x)\), for any new \(x\)
  - Low/no error on future queries (songs)

Possible A: Minimize \(\sum_{i=1}^{n} L(y_i, f_{(a,b,c,\ldots)}(x_i))\) with respect to \(a, b, c, \ldots\)
Classification loss function

Most common loss: 0-1 loss function

\[ L_{0-1}(y, f(x)) = \mathbb{I}(y \neq f(x)) \]

More general loss functions are defined by a \( m \times m \) cost matrix \( C \) such that

\[ L(y, f(x)) = C_{ab} \]

where \( y = a \) and \( f(x) = b \)

<table>
<thead>
<tr>
<th>Class</th>
<th>T0</th>
<th>T1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0</td>
<td>( C_{10} )</td>
</tr>
<tr>
<td>P1</td>
<td>( C_{01} )</td>
<td>0</td>
</tr>
</tbody>
</table>

\( T0 \) (true class 0), \( T1 \) (true class 1)
\( P0 \) (predicted class 0), \( P1 \) (predicted class 1)

http://en.wikipedia.org/wiki/Loss_function
**k-Nearest-Neighbor Classifier**

The classifier:

\[ f(x) = \text{majority label of the } k \text{ nearest neighbors (NN) of } x \]

**Model parameters:**
- number of neighbors \( k \)
- distance function \( d(.,.) \)
k-Nearest-Neighbor Classifier

If $k$ and $d(.,.)$ are fixed

Things to learn: ?
How to learn them: ?

If $d(.,.)$ is fixed, but you can change $k$

Things to learn: ?
How to learn them: ?
$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$

**k-Nearest-Neighbor Classifier**

If $k$ and $d(.,.)$ are fixed

**Things to learn:** Nothing

**How to learn them:** N/A

If $d(.,.)$ is fixed, but you can change $k$

**Things to learn:** Nothing

**How to learn them:** N/A

**Selecting $k$:** Try different values of $k$ on some hold-out set
15-NN

1-NN

Pretty good!

Overfitted
Cross-validation

Find the best performing $k$

1. **Hold out a part** of the training data (this part called “test set” or “held out set”)

2. **Train your classifier** on the **rest of the data** (called training set)

3. Compute testing error on the **test set** (You can also compute training error on the training set)

4. Do this multiple times, once for each $k$, and pick the $k$ with best performance
   - with respect to the error (on hold-out set) averaged over all hold-out sets
5-fold CV
Cross-validation: Holdout sets

Leave-one-out cross-validation (LOO-CV)
• hold out sets of size 1

K-fold cross-validation
• hold sets of size \((n / K)\)
• \(K = 10\) is most common (i.e., 10 fold CV)
Points about cross-validation

Requires extra computation, but gives you information about expected test error

LOO-CV:

• Advantages
  o Unbiased estimate of test error (especially for small $n$)
  o Low variance

• Caveats
  o Extremely time consuming
Points about cross-validation

$K$-fold CV:

• Advantages
  o More efficient than LOO-CV

• Caveats
  o $K$ needs to be large for low variance
  o Too small $K$ leads to under-use of data, leading to higher bias

• Usually accepted value $K = 10$

Reading material:

• ESL book, Chapter 7.10

• Le Song's slides on CV
\[ x_i = (x_{i1}, \ldots, x_{id}); \quad y_i = \{1, \ldots, m\} \]

**k-Nearest-Neighbor Classifier**

If \( k \) is fixed, but you can change \( d(.,.) \)

**Things to learn:** ?

**How to learn them:** ?

**Cross-validation:** ?

Possible distance functions:

- **Euclidean distance:** \( \|x_i - x_j\|_2 = \sqrt{(x_i - x_j)^\top(x_i - x_j)} \)
- **Manhattan distance:** \( \|x_i - x_j\|_1 = \sum_{i=1}^{d} |x_{il} - x_{jl}| \)
- ...
$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$

**k-Nearest-Neighbor Classifier**

If $k$ is fixed, but you can change $d(.,.)$

**Things to learn:** distance function $d(.,.)$

**How to learn them:** optimization

**Cross-validation:** any regularizer you have on your distance function

Summary on k-NN classifier

• Advantages
  o No learning (unless you are learning the distance functions)
  o Quite powerful in practice (and has theoretical guarantees as well)

• Caveats
  o Computationally expensive at test time

Reading material:
• ESL book, Chapter 13.3
• Le Song's slides on kNN classifier
  http://www.cc.gatech.edu/~lsong/teaching/CSE6740/lecture2.pdf
Decision trees (DT)

Dependent variable: PLAY

OUTLOOK?
- sunny
  - HUMIDITY?
    - <= 70
      - Play 2
      - Don't Play 0
    - > 70
      - Play 2
      - Don't Play 3
- overcast
  - Play 4
  - Don't Play 0
- rain
  - WINDY?
    - TRUE
      - Play 0
      - Don't Play 2
    - FALSE
      - Play 3
      - Don't Play 0
Decision trees (DT)

$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$

The classifier:

$f_T(x)$ is the majority class in the leaf in the tree $T$ containing $x$

Model parameters: The tree structure and size
Decision trees

Things to learn: ?
How to learn them: ?
Cross-validation: ?

\[ x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\} \]
Decision trees

Things to learn: the tree structure
How to learn them: (greedily) minimize the overall classification loss
Cross-validation: finding the best sized tree with $K$-fold cross-validation

$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$
\[ x_i = (x_{i1}, \ldots, x_{id}); \quad y_i = \{1, \ldots, m\} \]

**Learning the tree structure**

**Pieces:**
1. best split on the chosen attribute
2. best attribute to split on
3. when to stop splitting
4. cross-validation
Choosing the split

Split types for a selected attribute $j$:

1. **Categorical attribute** (e.g. `genre')
   
   $x_{1j} = \text{Rock}, x_{2j} = \text{Classical}, x_{3j} = \text{Pop}$

2. **Ordinal attribute** (e.g. `achievement')
   
   $x_{1j} = \text{Gold}, x_{2j} = \text{Platinum}, x_{3j} = \text{Silver}$

3. **Continuous attribute** (e.g. song length)
   
   $x_{1j} = 235, x_{2j} = 543, x_{3j} = 378$

\[ x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\} \]
Choosing the split

At a node $T$ for a given attribute $d$, select a split $s$ as following:

$$\min_s \text{loss}(T_L) + \text{loss}(T_R)$$

where $\text{loss}(T)$ is the loss at node $T$

Node loss functions:

- Total loss: $\sum_{x_i \in T} L(y_i, f_T(x_i)) - \sum_{c \in T} p_{cT} \log p_{cT}$
- Cross-entropy: where $p_{cT}$ is the proportion of class $c$ in node $T$
\[ x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\} \]

**Choosing the attribute**

Choice of attribute:

1. Attribute providing the maximum improvement in training loss
2. Attribute with maximum information gain

\[
\min_{j \in \{1, \ldots, d\}} \min_{s(j)} \text{loss}(T_L) + \text{loss}(T_R)
\]
When to stop splitting?

1. Homogenous node (all points in the node belong to the same class OR all points in the node have the same attributes)

2. Node size less than some threshold

3. Further splits provide no improvement in training loss

\[ \text{loss}(T) \leq \text{loss}(T_L) + \text{loss}(T_R) \]
Controlling tree size

In most cases, you can drive training error to zero (how? is that good?)

What is wrong with really deep trees?

• **Really high "variance"**

What can be done to control this?

• *Regularize* the tree complexity
  
o  Penalize complex models and prefers simpler models (why?)

Look at Le Song's slides on the decomposition of error in bias and variance of the estimator [here](http://www.cc.gatech.edu/~lsong/teaching/CSE6740/lecture13-cv.pdf)
Summary on decision trees

• Advantages
  o Easy to implement
  o Interpretable
  o Very fast test time
  o Can work seamlessly with mixed attributes
  o ** Works quite well in practice

• Caveats
  o Can be too simplistic
  o Training can be very expensive
  o Cross-validation is hard (and plain annoying)
Final words on decision trees

Reading material:

• ESL book, Chapter 9.2

• Le Song's slides