Classification - II

Model combination and visualization

http://poloclub.gatech.edu/cse6242
<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>X</th>
<th>Y</th>
<th>Artist</th>
<th>Len.</th>
<th>...</th>
<th>F_d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some nights</td>
<td>🤝 Fun</td>
<td>4:23</td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Skyfall</td>
<td>😞 Adele</td>
<td>4:00</td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Comf. numb</td>
<td>😞 Pink Fl.</td>
<td>6:13</td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>We are young</td>
<td>🤝 Fun</td>
<td>3:50</td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Chopin's 5th</td>
<td>?? Chopin</td>
<td>5:32</td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Training a classifier

Q: How do you learn appropriate values for \(a, b, c, \ldots\) 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_{01} )</td>
</tr>
<tr>
<td>P1</td>
<td>( C_{01} )</td>
<td>0</td>
</tr>
</tbody>
</table>

\( T_0 \) (true class 0), \( T_1 \) (true class 1)

\( P_0 \) (predicted class 0), \( P_1 \) (predicted class 1)
<table>
<thead>
<tr>
<th>Method</th>
<th>Coding</th>
<th>Training time</th>
<th>Cross validation</th>
<th>Testing time</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>kNN classifier</td>
<td>None</td>
<td>Can be slow</td>
<td>None</td>
<td>Slow</td>
<td>??</td>
</tr>
<tr>
<td>Naive Bayes classifier</td>
<td>Fast</td>
<td>None</td>
<td>Fast</td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>Decision trees</td>
<td>Slow</td>
<td>Very slow</td>
<td>Very fast</td>
<td>??</td>
<td></td>
</tr>
</tbody>
</table>
What to pick?

Possible strategies:

● Go from simplest model to more complex model until you obtain desired accuracy
● Discover a new model if the existing ones do not work for you
● Combine all (simple) models
Strategy 1

Consider the data set $S = \{(x_i, y_i)\}_{i=1,...,n}$

- Pick a sample $S^*$ with replacement of size $n$ from $S$
- Do the training on this set $S^*$ to get a classifier $f^*$
- Repeat the above step $B$ times to get $f_1, f_2, \ldots, f_B$
- Final classifier $f(x) = \text{majority}\{f_b(x)\}_{j=1,\ldots,B}$

This is called **Bagging**
Bagging

Why would bagging work?
● Combining multiple classifiers reduces the variance of the final classifier

When would this be useful?
● We have a classifier with low bias and high variance (any examples)
Bagging decision trees

Consider the data set $S$

- Pick a sample $S^*$ with replacement of size $n$ from $S$
- Grow a decision tree $T_b$ greedily
- Repeat $B$ times to get $T_1, \ldots, T_B$
- The final classifier will be

$$f(x) = \text{majority}\{f_{T_b}(x)\}_{b=1,\ldots,B}$$
Random decision trees

Grow a decision tree greedily until there are at most $N_{min}$ points in any node using the following strategy:

- Randomly pick any $m$ of the $d$ attributes available
- Find the best split/attribute from only these $m$ attributes available

Bagged random decision trees

= Random forests
Points about random forests

Algorithm parameters

- Usual values for $m$: $\sqrt{d}, 1, 10$
- Usual values for $N_{min} \geq 1$
  (applicable in classification only)
- Usual value for $B$: keep increasing $B$
  until the training error stabilizes
Bagging/Random forests

Consider the data set $S = \{(x_i, y_i)\}_{i=1,\ldots,n}$

- Pick a sample $S^*$ with replacement of size $n$ from $S$
- Do the training on this set $S^*$ to get a classifier (e.g. random decision tree) $f^*$
- Repeat the above step $B$ times to get $f_1, f_2, \ldots, f_B$
- Final classifier

$$f(x) = \text{majority}\{f_b(x)\}_{j=1,\ldots,B}$$
Final words

Advantages

● Efficient and simple training
● Allows you to work with simple classifiers
● ** Random-forests generally useful and quite accurate in practice
● Embarrassingly parallelizable

Caveats:

● Needs low-bias classifiers
● Can make a not-good-enough classifier worse
Final words

Reading material

- Bagging: ESL Chapter 8.7
- Random forests: ESL Chapter 15

Strategy 2: Boosting

Consider the data set $S = \{(x_i, y_i)\}_{i=1,\ldots,n}$

- Assign a weight $w_{(i,0)} = (1/n)$ to each $i$
- Repeat for $t = 1,\ldots,T$:
  - Train a classifier $f_t$ on $S$ that minimizes the weighted loss: $\sum_{i=1}^n w_{(i,t)} L(y_i, f_t(x_i))$
  - Obtain a weight $a_t$ for the classifier $f_t$
  - Update the weight for every point $i$ to $w_{(i, t+1)}$ as following:
    - Increase the weights for $i$: $y_i \neq f_t(x_i)$
    - Decrease the weights for $i$: $y_i = f_t(x_i)$
- Final: $f(x) = \text{sign} \left( \sum_{t=1}^T a_t f_t(x) \right)$
Final words on boosting

Advantages

- Extremely useful in practice and has great theory as well
  - Better accuracy than random forests usually
- Can work with very simple classifiers

Caveats:

- Training is inherently sequential
  - Hard to parallelize

Reading material:

- ESL book, Chapter 10
Visualization in Classification

Usual tools

- ROC curve / cost curves
  - True-positive rate vs. false-positive rate

- Confusion matrix
Visualization in Classification

Newer tool

- Visualize the data and the class boundary with some 2D projection
Weights in combined models

Bagging / Random forests
- Majority voting

Boosting
- Systematic weighting based on its individual performance

Would it be useful to allow humans to play with these weights?
EnsembleMatrix

Understanding performance

- Identify problem areas
- Reorder rows/columns to put confused classes together
  - Can use a graph clustering algorithm
Improving performance

Improving performance

- Adjust the weights of the individual classifiers
- Data partition to separate out problem areas
  - Adjust weights just for these individual parts
- Claimed state-of-the-art performance!
  - *on one dataset

Figure 3. After partitioning the matrix, selecting a partition, outlined in orange, causes the thumbnails to display only the data instances in that partition. The component classifiers demonstrate very different behavior in this partition, including clustering and large differences in accuracy.

ReGroup - Naive Bayes at work

# ReGroup

<table>
<thead>
<tr>
<th>Features to represent each friend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gender, Age group</td>
</tr>
<tr>
<td>Family</td>
</tr>
<tr>
<td>Home city/state/country</td>
</tr>
<tr>
<td>Current city/state/country</td>
</tr>
<tr>
<td>High school/college/grad school</td>
</tr>
<tr>
<td>Workplace</td>
</tr>
<tr>
<td>Amount of correspondence</td>
</tr>
<tr>
<td>Recency of correspondence</td>
</tr>
<tr>
<td>Friendship duration</td>
</tr>
<tr>
<td># of mutual friends</td>
</tr>
<tr>
<td>Amount seen together</td>
</tr>
</tbody>
</table>

**Y - In group?**

**X - Features of a friend**

\[ P(Y = \text{true} | X) = ? \]

Compute \( P(X_d | Y = \text{true}) \) for each feature \( d \) using the current group members (how?)

ReGroup

Y - In group?
X - Features of a friend

\[ P(Y|X) = P(X|Y)P(Y)/P(X) \]
\[ P(X|Y) = P(X_1|Y) \cdots P(X_d|Y) \]

Compute \( P(X_i|Y = true) \) for every feature \( d \) using the current group members

- Use simple counting

Not exactly classification!

- Reorder remaining friends with respect to \( P(X|Y=true) \)
- "Train" every time a new member is added to the group

Some additional reading

- Interactive machine learning