On the Inverse Classification Problem and its Applications

Based on Aggarwal, Chen & Han (2006) and Lowd & Meek (2005)

Presented by Tianyu Cao
Outline

- Section 1
  - Application Background
  - Problem Definition
  - Gini Index and Support Threshold
  - Inverted List & Inverse Classification Algorithm
  - Experimental Results
  - Conclusions & Discussions

- Section 2
  - A Naive Idea for Inverse Classification

- Section 3
  - Implications for Security Issues on Machine learning
Section 1

Based on the paper “On the Inverse Classification and its Application”.
Application Background

- Action driven applications in which the features can be used to define certain actions. How to select feature values to drive the decision support system towards a desired end-result?

- Example.
  - Mass marketing
  - Mail Spammer
  - Possible attacks to a general classification system
Problem Definition

- Given a training data set $D$ that contains $N$ records, $X_1, X_2, X_3, \ldots X_n$. The test dataset contains $M$ records, $Y_1, Y_2, Y_3, \ldots Y_m$. Each record $Y_i$ is not completely defined. Each record $Y_i$ is associated with a desired class label $C_i$.

- The inverse classification is defined as how to fill the unknown feature $f_j$ for each record $Y_i$ such that the probability of $Y_i$ being classified into $C_i$ is maximized.
### Example

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>Decision variable 1</th>
<th>Decision variable 2</th>
<th>...</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>?</td>
<td>?</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>0002</td>
<td>?</td>
<td>?</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>...</td>
<td>?</td>
<td>?</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

How to fill the missing values such that the customer will most likely to respond to the mailer?
Ways to fill the unknown features

Assume the missing features are $i_1, i_2, i_3, \ldots i_q$. Suppose the number of possible values for these features are $v_{i_1}, v_{i_2}, v_{i_3}, \ldots v_{i_q}$. The number of ways to fill the missing value is as follows.

$$\prod_{j=1}^{q} v_{i_j}$$
How to Search Efficiently in this Space

- Some feature value combinations are not good and thus should be pruned out.
- Two prune strategies.
  - Gini index
  - Support threshold
- Use Gini index to rank the candidate combinations.
Gini Index

- Some notations.
- \( L(i,q) \) be the set of points in the training set such that the value of the \( i \)th feature is the \( q \)th possible value of the \( i \)th feature.
- \( F(i,q,s) \) be the fraction of points in the set \( L(i,q) \) such that the class label is \( s \).
Gini Index

- Gini index is defined as follows.

\[ G(L(i, q)) = \sum_{s=1}^{k} f(i, q, s)^2 \]

- The least discriminative case. Each \( f(i, q, s) = 1/k \).
- The most discriminative case. Only one \( f(i, q, s) \) is 1, all the other \( f(i, q, s) \) is 0.
### Example

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>Decision variable 1</th>
<th>Decision variable 2</th>
<th>Decision variable 3</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>0002</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>0003</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>0004</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>0005</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>0006</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>0007</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>0008</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Example

- $L(1,1)$ is $\{0001,0002,0003,0004\}$.
- Therefore the gini index of $L(1,1)$ is $(1/2)^2+(1/2)^2=1/2$, which is the least discriminative case.
- Consider the combination of feature 1 is 1 and feature 3 is 1. The set of points are $\{0002,0004\}$. The gini index is $1^2+0=1$, which is the most discriminative case.
Support Threshold

- The number of points in the set $L(i,q)$ must be greater than the support threshold.
- In the previous example, if the support threshold is 4. Even though the combination of feature 1 is 1 and feature 3 is 1 satisfies the gini index requirement, it does not satisfy the support threshold.
Inverted List

- An efficient data structure that can be used to calculate gini index and support for a given combination of feature values.
- Similar to inverted index in information retrieval.
## Inverted List

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

![Diagram of Inverted List](image_url)
Calculate Combination of L(i,q)

- To calculate the set of points that satisfy feature 1 is 1 and feature 2 is 0, we simply merge the inverted list L(1,1) and L(2,0).
- This corresponds to \( L(1,1) \cap L(2,0) \).
- For the convenience of calculating Gini index, the class label is also stored.
The Inverse Classification Algorithm

- Step 1. Compute all possible singleton sets $L_1$ for test example $T$ with support at least $s$ and gini-index at least $a$ and prune out those $L_1$ that dominant class is not the target class. Set $k=1$.

- Step 2. Join $L_k$ and $L_1$ to form $C_{k+1}$; Prune $C_{k+1}$ according to gini index and support threshold. Assume the remaining is $C_{k+1}'$.

- Step 3. Let $L_{k+1}=C_{k+1}'$, if $L_{k+1}$ is not empty. Go to step 2.
The Inverse Classification Algorithm

- Step 4 Let LF be the union of all Lk.
- Step 5 While there are missing values in T, go to step 6 otherwise terminate.
- Step 6 Pick the feature value combination H with highest gini-index from LF. Add the missing value in T using the corresponding values in H. Remove any sets in LF which are inconsistent with the filled value in T. Go to step 5.
Experimental Considerations and Settings

- No benchmark datasets for this problem.
- Used UCI dataset by dividing the dataset into three parts. Two parts are used as training and the remaining part are used as testing.
- Set some feature in the testing set to be missing.
- Try to use the inverse classification algorithm to fill the missing value. In this case the desired class label is the test instance’s actual class label.
Experimental Results

Figure 2: Naive Bayesian: Accuracy boost on mushroom with increasing number of removed attributes
Experimental Results

Figure 3: REPTree: Accuracy boost on mushroom with increasing number of removed attributes
Experimental Results

Figure 4: Decision Table: Accuracy boost on mushroom with increasing number of removed attributes
Experimental Results

Figure 14: Sensitivity for mushroom: Accuracy with Support
Experimental Results

Figure 15: Sensitivity for mushroom: Accuracy with GiniThreshold
Experimental Results

Figure 22: Time complexity on mushroom w.r.t. data size
Experimental Results

Figure 23: Time complexity on mushroom w.r.t. Attr_Remove
Summary of Section 1

- The accuracy before substitution of the missing attributes reduces with increasing number of missing attributes.
- The accuracy of the classifier showed a considerable boost over original dataset.
- In general the default gini index in training set served an effective default value.
- The threshold value could be defined well by the average number of data points per nominal attributes.
- The algorithm is scalable to the dataset size and the number of missing attributes.
Discussions

- The inverse classification problem algorithm does not make use of any classification models. Why not?
- Relaxation to the problem itself.
- The inverse classification is defined as how to fill the unknown feature $f_j$ for each record $Y_i$ such that the probability of $Y_i$ being classified in to $C_i$ is maximized.
Discussions

- Sometimes it is enough to fill the missing value so that the data point $Y_i$ is likely to be classified into the target class $C_i$. It means that the probability of data point $Y_i$ being classified into $C_i$ is reasonably high.

- Combining these two ideas we can get a somewhat naive solution to the inverse classification problem.
Section 2

Based on the presenter’s thoughts.
A Naive Idea

- **Step 1** Build a classifier C from the training data.
- **Step 2** Consider the missing attributes of a test instance T as a subspace, uniformly random sample a point Pi from this subspace.
- **Step 3** Fill Pi into the missing attributes of the testing instance T and send the test instance to the classifier C. If C classifies T into the desired class Ci, the process ends. Otherwise go to step 2.
A Naive Idea

- As long as the classifier C fits to the training data set reasonably well, we should expect that the instance T with missing value filled is likely to be classified to the target class by any classifier that fits the training data reasonably well.
A Naive Idea

- Second Problem.
- How many points do we need to sample in order to find one that will be classified into the target class by the classifer C?
- It turns out we only need to sample a few points.
A Naive Idea

Assume we sample $n$ points in this subspace, we want a blue point. The probability that we can get a blue point is as follows.

$$1 - (P_{red})^n$$

This probability approaches to 1 exponentially fast as long as the fraction of blue points is not 0. Consider an extreme case, the fraction of blue points is 1% and the fraction of red points is 99%. Even with such an extreme case, the probability that you can get a blue point of sampling 500 points is actually higher than 99%.
A Naive Idea

- There are problems with the naive solution even for the relaxed inverse classification problem.
- It is not always able to find a classifier to fit well to the training data.
- How to justify the point that you find is likely to be classified to the desired target class. (You cannot use the same classifier for testing this.)
Section 3

Based on the slides from “Good Word Attacks on Statistical Spam Filters”.
Implications for Security Issues

- As such problem arises, this poses a potential threat to machine learning system. Adversarial learning deals with such issues.
- Consider a Content Based Spam Filtering System. The success of inverse classification implies that the spammer can manipulate the spam mail so that it can pass the Spam Filter.
Content-based Spam Filtering

1. From: spammer@example.com
   Cheap mortgage now!!!

2. cheap = 1.0
   mortgage = 1.5

3. Total score = 2.5 > 1.0 (threshold)

Spam
Content-based Spam Filtering

1. From: spammer@example.com
   Cheap mortgage now!!! Corvallis
   Corvallis  OSU

2. cheap = 1.0
   mortgage = 1.5
   Corvallis = -1.0
   OSU = -1.0

3. Total score = 0.5 < 1.0 (threshold)

OK
Implications for Security Issues

- It is generally difficult because the spammer does not know the feature the spam filter uses. The spammer does not have the necessary training data.
- But assume the spammer does somehow both of these, how well can he do?
Good Word Attacks

- Practice: good word attacks
  - Passive attacks
  - Active attacks
  - Experimental results
Attacking Spam Filters

- Can we efficiently find a list of “good words”?
- Types of attacks
  - Passive attacks -- no filter access
  - Active attacks -- test emails allowed
- Metrics
  - Expected number of words required to get median (blocked) spam past the filter
  - Number of query messages sent
Filter Configuration

- **Models used**
  - Naïve Bayes: generative
  - Maximum Entropy (Maxent): discriminative

- **Training**
  - 500,000 messages from Hotmail feedback loop
  - 276,000 features
  - Maxent let 30% less spam through
Passive Attacks

- **Heuristics**
  - Select random dictionary words (Dictionary)
  - Select most frequent English words (Freq. Word)
  - Select highest ratio: English freq./spam freq. (Freq. Ratio)

- **Spam corpus: spamarchive.org**

- **English corpora:**
  - Reuters news articles
  - Written English
  - Spoken English
  - 1992 USENET
## Passive Attack Results

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Corpus</th>
<th>NB Words</th>
<th>ME Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionary</td>
<td>N/A</td>
<td>-132</td>
<td>-555</td>
</tr>
<tr>
<td>Freq. Word</td>
<td>Reuters</td>
<td>1,350</td>
<td>1,420</td>
</tr>
<tr>
<td></td>
<td>Written</td>
<td>1,920</td>
<td>2,110</td>
</tr>
<tr>
<td></td>
<td>Spoken</td>
<td>1,470</td>
<td>1,040</td>
</tr>
<tr>
<td></td>
<td>USENET</td>
<td>898</td>
<td>845</td>
</tr>
<tr>
<td>Freq. Ratio</td>
<td>Reuters</td>
<td>112</td>
<td>735</td>
</tr>
<tr>
<td></td>
<td>Written</td>
<td>133</td>
<td>305</td>
</tr>
<tr>
<td></td>
<td>Spoken</td>
<td>159</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>USENET</td>
<td>257</td>
<td>149</td>
</tr>
</tbody>
</table>
Active Attacks

- Learn which words are best by sending test messages (queries) through the filter
- First-N: Find $n$ good words using as few queries as possible
- Best-N: Find the best $n$ words
First-N Attack

Step 1: Find a “Barely spam” message

Original legit.

Hi, mom!

“Barely legit.”

now!!!

“Barely spam”

mortgage now!!!

Original spam

Cheap mortgage now!!!

Spam

Threshold
First-N Attack
Step 2: Test each word

Good words

“Barely spam” message

Legitimate

Spam

Less good words

Threshold
Best-N Attack

Key idea: use spammy words to sort the good words.
Active Attack Results
(n = 100)

<table>
<thead>
<tr>
<th>Attack type</th>
<th>Naive Bayes words (queries)</th>
<th>Maxent words (queries)</th>
</tr>
</thead>
<tbody>
<tr>
<td>First-N</td>
<td>59 (3,100)</td>
<td>20 (4,300)</td>
</tr>
<tr>
<td>Best-N</td>
<td>29 (62,000)</td>
<td>9 (69,000)</td>
</tr>
<tr>
<td>Passive</td>
<td>112 (0)</td>
<td>149 (0)</td>
</tr>
</tbody>
</table>

- Active attacks much more effective than passive attacks
References

1. Charu C. Aggarwal, Chen Chen, Jiawei Han. On the Inverse Classification Problem and its Applications. ICDE 2006.

Thanks