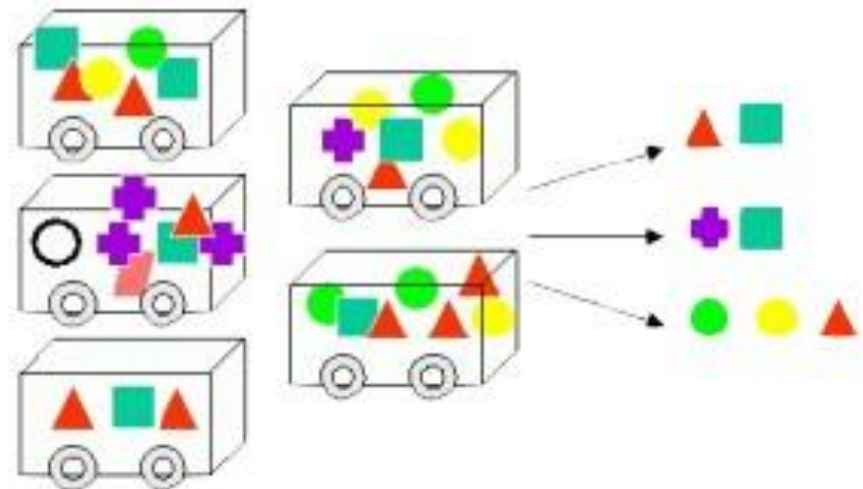


# Association Rules & Unbalanced Classification

Associate Professor  
**HUANG, Ke-Wei**



# Association Rule Mining

- This type of algorithms is covered in almost all data mining textbooks.
- It is the backend algorithm of many recommendation systems.
- The goal is to find out which set of items predict the occurrence of other items  $\Leftrightarrow$  what are the items bought together more often.
- Also known as “affinity analysis” or “market basket” analysis.
- This problems become an interesting problem for CS professors because in reality, the number of transactions is huge! How to find products frequently bought together is not only a statistical problem, but also a computer science problem (scalability).

# Market-Basket Transactions

Basket = Transaction	Items
1	Bread, Milk
2	Bread, Diapers, Beer, Eggs
3	Milk, Diapers, Beer, Coke
4	Bread, Milk, Diapers, Beer
5	Bread, Milk, Diapers, Coke

**Association  
Rules from  
these  
transactions**

**$X \rightarrow Y$   
(antecedent  $\rightarrow$  consequent)**

$\{\text{Diapers}\} \rightarrow \{\text{Beer}\},$   
 $\{\text{Milk, Bread}\} \rightarrow \{\text{Diapers}\}$   
 $\{\text{Beer, Bread}\} \rightarrow \{\text{Milk}\},$   
 $\{\text{Bread}\} \rightarrow \{\text{Milk, Diapers}\}$

# Examples

- Market basket analysis/affinity analysis
  - What products are bought together?
  - Where to place items on grocery store shelves?
  - What to place when customers queue for cashier?
  - Diaper and Beer story is the classical example.
- Amazon's recommendation engine
  - People who bought this product also bought...
  - Customers who viewed this item also viewed
- Telephone calling patterns for telecoms
  - The set of people tend to call most often?
- Social network analysis
  - Determine who you "may know"

# Examples

- Youtube and Netflix videos recommended to you.
- News websites typically have a section of related news articles and also news recommended to you (not by popularity only).
- From old personalized catalogues to the personalized email campaigns.
- Similar applications (but algorithms are slightly different):
  - Pandora => by your “like” and “dislike”, recommend pop music for you.
  - Stumbleupon => by your “thumb up” and “thumb down”, recommend interesting and new web pages to users.

# Examples

---

- Searching for interesting and frequently occurring patterns of DNA and protein sequences in cancer data
- Finding patterns of purchases or medical claims that occur in combination with fraudulent credit card or insurance use
- Identifying combinations of behavior that precede customers dropping their cellular phone service or upgrading their cable television package

# Retailers are indeed getting Smarter

---

“How Target Figured Out A Teen Girl Was Pregnant Before Her Father Did” Forbes...

- By data mining, Target found that women on the baby registry were buying larger quantities of unscented lotion around the beginning of their second trimester.
- Another analyst noted that sometime in the first 20 weeks, pregnant women loaded up on supplements like calcium, magnesium and zinc.
- Many shoppers purchase soap and cotton balls, but when someone suddenly starts buying lots of scent-free soap and extra-big bags of cotton balls, in addition to hand sanitizers and washcloths, it signals they could be getting close to their delivery date.

# Retailers are indeed getting Smarter

---

- Target was able to identify about 25 products that, when analyzed together, allowed him to assign each shopper a “pregnancy prediction” score. More important, Target could also estimate her due date to within a small window, so Target could send coupons timed to very specific stages of her pregnancy.
- A high-school girl’s father complained to Target that “She’s still in high school, and you’re sending her coupons for baby clothes and cribs? Are you trying to encourage her to get pregnant?”
- And you can guess what happened? This father soon found out his daughter was indeed pregnant.



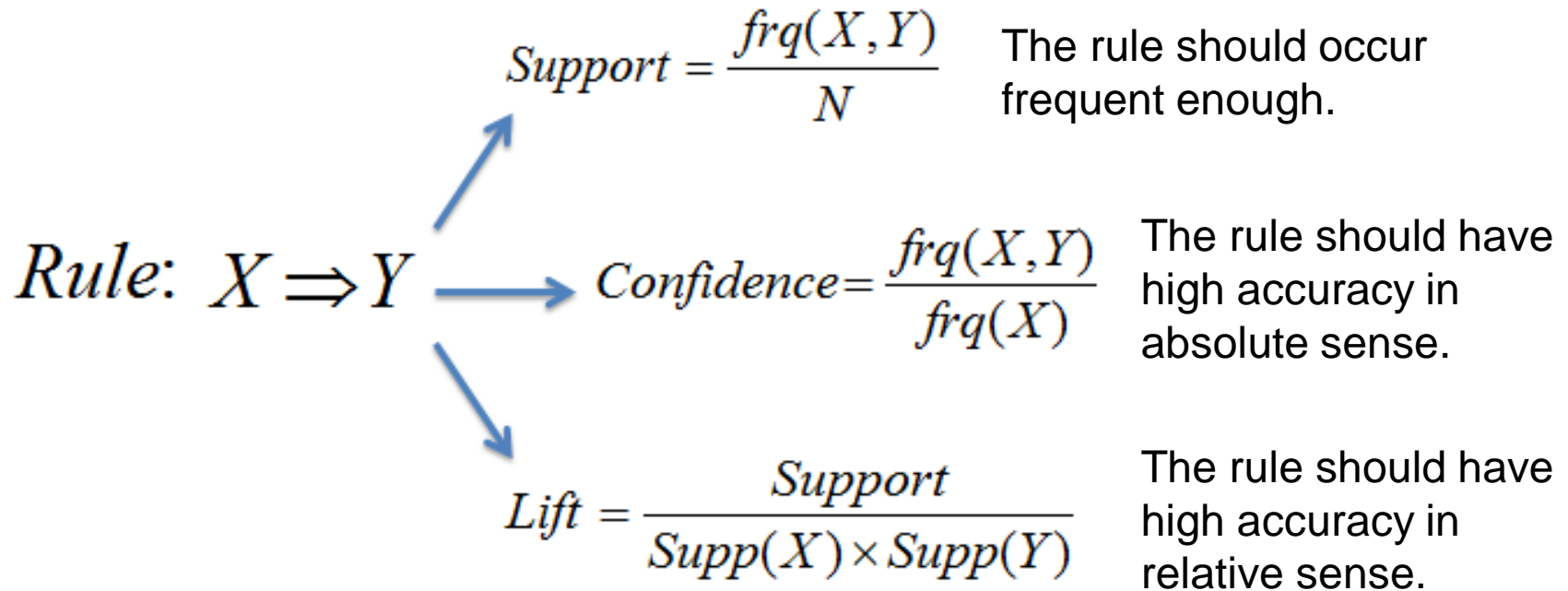
# Apriori Algorithm

## Definition: Frequent Itemset

- **Itemset**
  - A set of one or more items
    - E.g.: {Milk, Bread, Diaper}
  - $k$ -itemset
    - An itemset that contains  $k$  items
- **Support count ( $\sigma$ )**
  - Frequency of occurrence of an itemset (number of transactions it appears)
  - E.g.  $\sigma(\{\text{Milk, Bread, Diaper}\}) = 2$
- **Support**
  - Fraction of the transactions in which an itemset appears
  - E.g.  $s(\{\text{Milk, Bread, Diaper}\}) = 2/5$
- **Frequent Itemset**
  - An itemset whose support is greater than or equal to a threshold

<i>TID</i>	<i>Items</i>
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

# 3 Important Metrics in Association Rules



- In data mining and association rule learning, lift is a performance measure as having an enhanced response, measured against a random choice targeting model.
- In simple words, if without any modeling, we know something happens with probability  $x\%$ . With our model, the probability is  $y\%$ , then the lift is  $y/x$ .
- For example, suppose a population has an average response rate of 5%, but a certain model (or rule) has identified a segment with a response rate of 20%. Then that segment would have a lift of 4.0 ( $20\%/5\%$ ).

# Association rules

- Given a set of transactions **D**, find rules that will predict the occurrence of an item (or a set of items) based on the occurrences of other items in the transaction

## Market-Basket transactions

<i>TID</i>	<i>Items</i>
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

## Examples of association rules

$\{\text{Diaper}\} \rightarrow \{\text{Beer}\},$   
 $\{\text{Milk, Bread}\} \rightarrow \{\text{Diaper, Coke}\},$   
 $\{\text{Beer, Bread}\} \rightarrow \{\text{Milk}\},$

# An even simpler concept: frequent itemsets

- Given a set of transactions **D**, find combination of items that occur frequently

## Market-Basket transactions

<i>TID</i>	<i>Items</i>
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

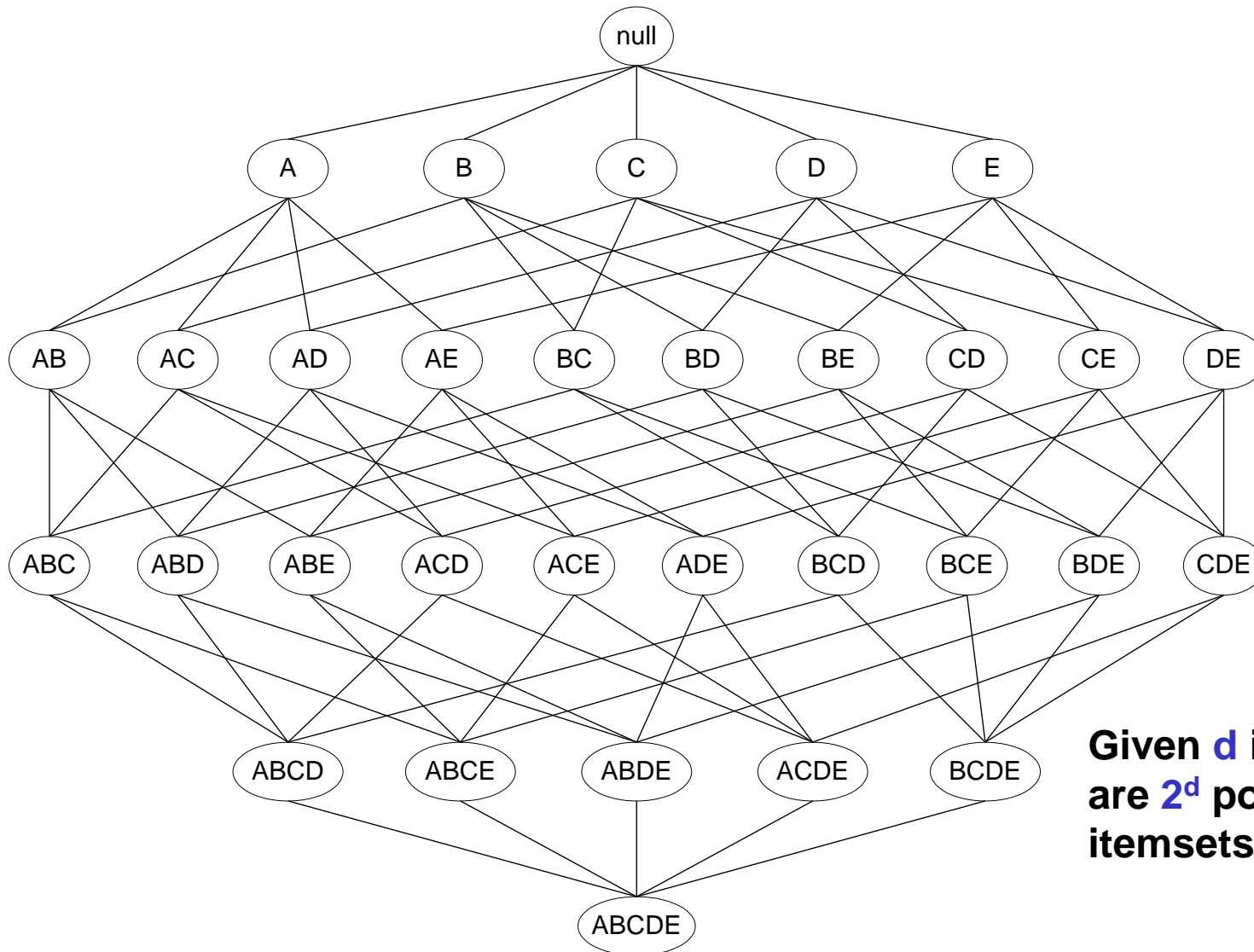
## Examples of frequent itemsets

{Diaper, Beer},  
{Milk, Bread}  
{Beer, Bread, Milk},

# Finding frequent sets

- **Task:** Given a transaction database **D** and a **min(support)** threshold find all frequent itemsets and the frequency of each set in this collection
- **Stated differently:** Count the number of times combinations of attributes occur in the data. If the count of a combination is above **min(support)** report it.
- **Recall:** The input is a transaction database **D** where every transaction consists of a subset of items from some universe
- This problem becomes difficult because in the real-world the number of item is huge => how many products does NTUC sell?

# How many itemsets are there?



Given **d** items, there are  **$2^d$**  possible itemsets

# Brute-force algorithm?

- Generate all possible itemsets (lattice of itemsets)
  - Start with 1-itemsets, 2-itemsets,...,d-itemsets
- Compute the frequency of each itemset from the data
  - Count in how many transactions each itemset occurs
- If the support of an itemset is above **min(support)**, report it as a frequent itemset.
- Complexity?
  - Match every candidate against each transaction
  - For **M** itemsets and **N** transactions, the complexity is~  **$O(NM)$**  => **Expensive since  $M = 2^d$  !!!**
  - NTUC clearly has more than one thousand products and how many transactions? Billions? How about Walmart?

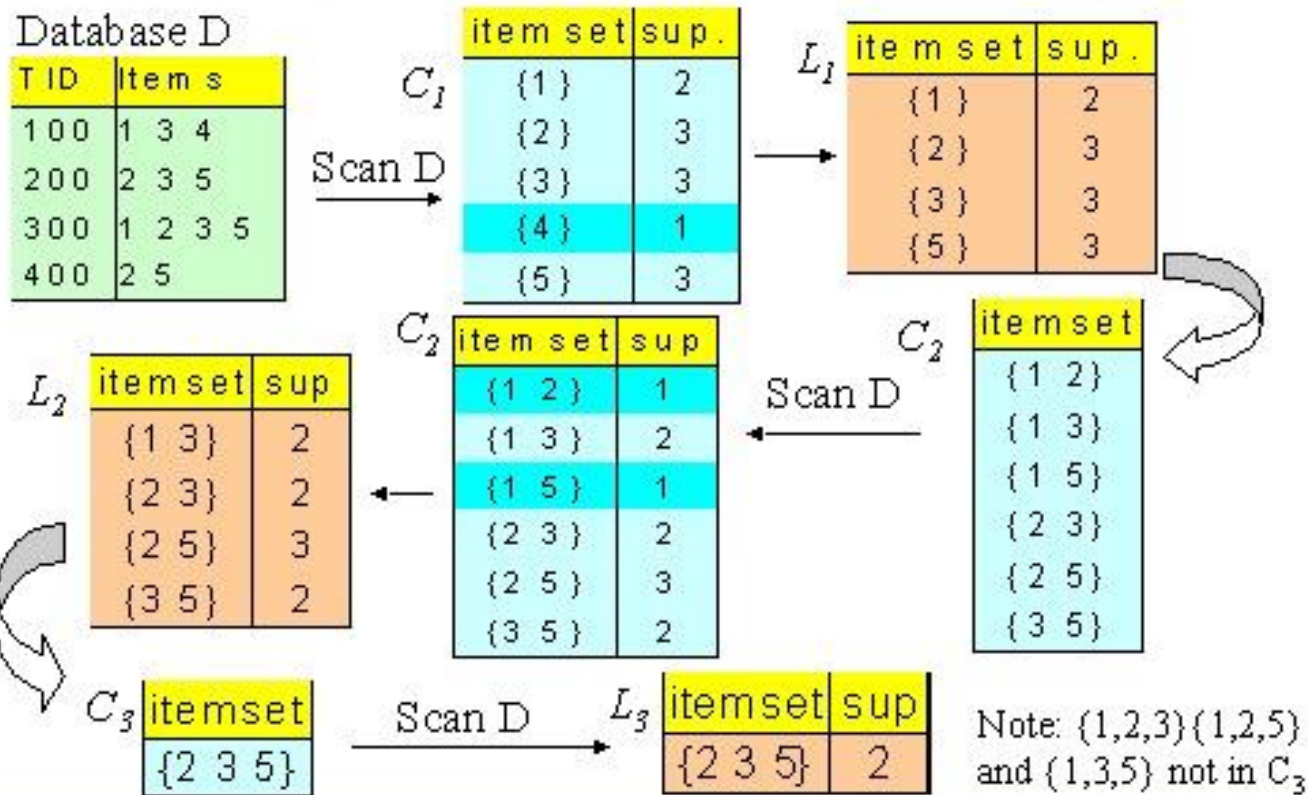


# Speeding-up the brute-force algorithm

- Reduce the **number of itemsets** (M)
  - Complete search:  $M=2^d$
  - Use pruning techniques to reduce M is the focus of Apriori algorithm next.
- Reduce the **number of transactions** (N)
  - Reduce size of N as the size of itemset increases
  - Some advanced algorithms will handle this aspect.
- Reduce the **number of comparisons** (NM)
  - Use efficient data structures to store the candidates or transactions
  - No need to match every candidate against every transaction

# Apriori Algorithm Example

## The Apriori Algorithm -- Example



# Apriori Algorithm Example

- C1 is trivial, we check all 1-item sets and count the support counts.
- L1 is generated from C1 by filtering out cases with low support. In this example, threshold  $> 1$
- C2 is the first **join step**. It is not as trivial as it looks like.
  - We consider all combinations two items out of 1, 2, 3, 5. At the same time, **the first item must be smaller than the second item in ID** => this implies we don't care the ordering and can reduce the size of C<sub>i</sub> later.
  - For any two-item sets, **we also need to check each one item is in C1**, which is automatically fulfilled in this round, but need to be checked in the following rounds.
- Then again we count the support of C2.
- Next, we create L2 by filtering out C2 cases with low support.

# Apriori Algorithm Example

- To generate C3 from L2, now is more complicated and without knowing Apriori, you cannot figure out it yourself unless you are very smart in math.
- It works like this. We compare any two rows in L2. The first item must be the same and then we will create a new itemset in C3 by sorted ordering.
- So, we only have one row start with "1" => no item in C3 starts with "1". We also have only one row start with "3" => no item in C3 starts with "1".
- We have two items in C3 that starts with "2" so we can create a new itemset {2,3,5}.
- NEXT STEP: it is not obvious in the current example but is an important **pruning step** in Apriori. We check {2,3} {2,5} {3, 5} (all k-1 itemsets) must be in C3 => YES! So we have one item in L3 and no more loop.

# Apriori Algorithm Example

- The meaningful frequent itemsets include L1, L2, and L3, NOT JUST L3.
- There are also several following steps not explained in detailed in this example; we need to create rules from these frequent itemsets.
- We can use confidence or lift as the performance metrics.
- For example, by  $\{2,3,5\}$ , we can check when  $\{2,3\}$  are bought, how many percentage of transactions also has 5? In our example, 2 out of 2 are correct (100%!) so it will mean any threshold of confidence. The program will output this rule:  $\{2,3\} \Rightarrow \{5\}$ .
- But if you check  $\{2,5\} \Rightarrow \{3\}$ ? Then 2 out of 3 are correct. Then it depends on your confidence threshold to decide whether this rule will be output to the user<sup>21</sup>

# Apriori Algorithm Example (Lift)

- We can calculate the Lift of rules from  $\{2,3,5\}$ .
- For  $\{2,3\} \Rightarrow \{5\}$ ,
  - the proportion of  $\{2,3\}$  is 50% and  $\{5\}$  is 75%. Under the independence assumption  $\{2,3\} \Rightarrow 5$  is  $3/8$ .
  - Our prediction is 100%
  - and thus the lift is  $8/3 \Rightarrow$  our analysis shows this rule increases the prediction probability by 2-3 ( $8/3$ ) times.
- But if you check  $\{2,5\} \Rightarrow \{3\}$ ?
  - the proportion of  $\{2,5\}$  is 75% and  $\{3\}$  is also 75%. Under the independence assumption  $\{2,5\} \Rightarrow \{3\}$  is  $9/16$ .
  - Our prediction is  $2/3$  accuracy.
  - The lift is  $(2/3) / (9/16) = 32/27$ , which is slightly higher than 1 and typically is not good enough.
- **You can also use lift to decide to keep  $\{2,5\} \Rightarrow \{3\}$  or not.**


# Summary

- Widely used in **recommender systems**
- The classical method is **Apriori algorithm**
- To reduce computation, we consider only “frequent” item sets (=support)
- Performance is measured by *confidence* and *lift*

Strengths	Weaknesses
<ul style="list-style-type: none"><li>• Is capable of working with large amounts of transactional data</li><li>• Results in rules that are easy to understand</li><li>• Useful for "data mining" and discovering unexpected knowledge in databases</li></ul>	<ul style="list-style-type: none"><li>• Not very helpful for small datasets</li><li>• Requires effort to separate the true insight from common sense</li><li>• Easy to draw spurious conclusions from random patterns</li></ul>

# Appendix: Apriori Algorithm

k is the number of items.

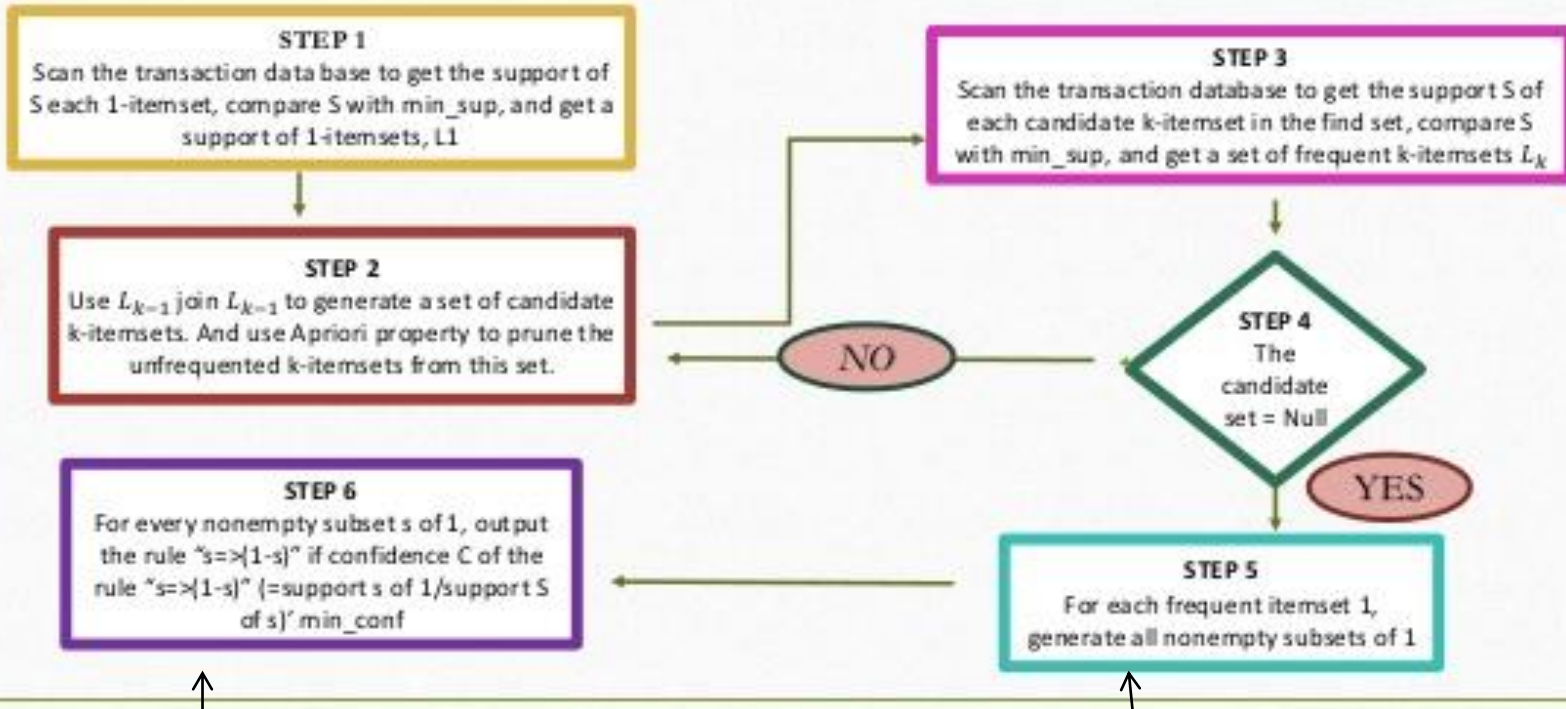


```
1)  $L_1 = \{\text{large 1-itemsets}\};$ 
2) for (  $k = 2; L_{k-1} \neq \emptyset; k++$  ) do begin
3)    $C_k = \text{apriori-gen}(L_{k-1});$  // New candidates – see Section 2.1.1
4)   forall transactions  $t \in \mathcal{D}$  do begin
5)      $C_t = \text{subset}(C_k, t);$  // Candidates contained in  $t$  – see Section 2.1.2
6)     forall candidates  $c \in C_t$  do
7)        $c.\text{count}++;$ 
8)   end
9)    $L_k = \{c \in C_k \mid c.\text{count} \geq \text{minsup}\}$ 
10) end
11)  $\text{Answer} = \bigcup_k L_k;$ 
```

Figure 1: Algorithm Apriori



# STEPS TO PERFORM APRIORI ALGORITHM



Check the rules meet the min. confidence requirement

We start generating rules.

# Apriori Algorithm

## 2.1.1 Apriori Candidate Generation

The `apriori-gen` function takes as argument  $L_{k-1}$ , the set of all large  $(k-1)$ -itemsets. It returns a superset of the set of all large  $k$ -itemsets. The function works as follows.<sup>1</sup> First, in the *join* step, we join  $L_{k-1}$  with  $L_{k-1}$ :

```
insert into  $C_k$ 
select  $p.item_1, p.item_2, \dots, p.item_{k-1}, q.item_{k-1}$ 
from  $L_{k-1} p, L_{k-1} q$ 
where  $p.item_1 = q.item_1, \dots, p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1};$ 
```

First important step

Next, in the *prune* step, we delete all itemsets  $c \in C_k$  such that some  $(k-1)$ -subset of  $c$  is not in  $L_{k-1}$ :

```
forall itemsets  $c \in C_k$  do
  forall  $(k-1)$ -subsets  $s$  of  $c$  do
    if  $(s \notin L_{k-1})$  then
      delete  $c$  from  $C_k$ ;
```

Second important step

# Handling Unbalanced Classification

- In many real world applications about (binary) classification, the proportion of  $Y=1$  is very small, such as less than 10%.
- If you just apply ordinary classification algorithms, precision could be high but recall is small.

For example

- Any kind of fraud detection, the proportion of fraudulent cases are small.
- In medicine related applications, the proportion of patients of a specific disease is small among all patients.

# Overview of Solutions

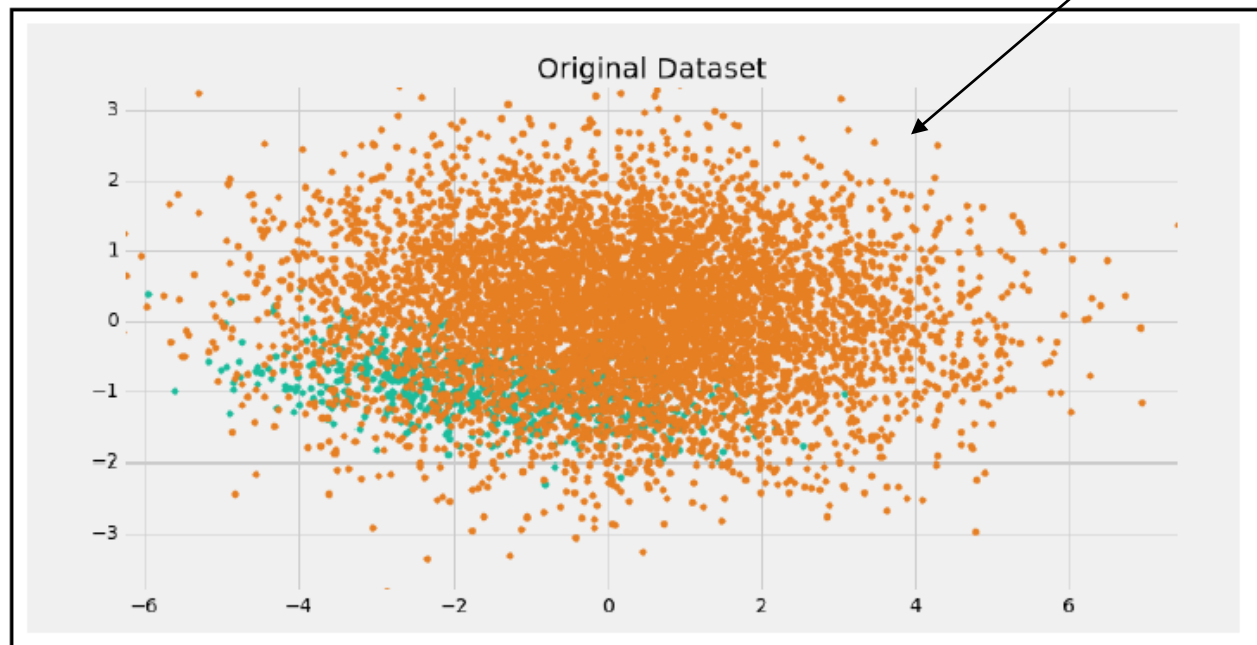
- Weighting
- Sampling
  - Under-sampling methods
  - Oversampling methods, including the famous method SMOTE
  - Using both under-sampling and oversampling together
- Other advanced methods
  - EasyEnsemble
  - BalanceCascade

References: More, Ajinkya. "Survey of resampling techniques for improving classification performance in unbalanced datasets." *arXiv preprint arXiv:1608.06048* (2016).

# Experimental Setting

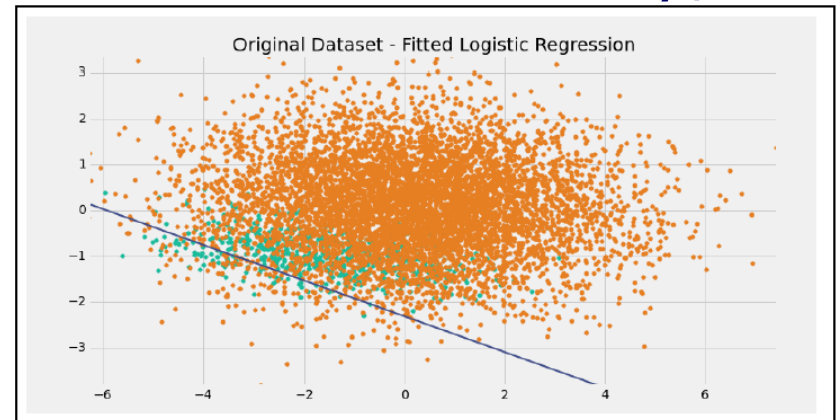
- Python library scikit-learn.
- n samples = 10000 (number of data points)
- n classes = 2 (number of classes)
- weights = [0:1; 0:9] (fraction of sizes of each class)
- Two numerical features

The patten looks like



# Baseline Model

- Logistic regression with l1 or l2 penalty tuned by 5-fold cross validation on the training dataset (70% training and 30% testing).
- This almost completely failed because if I classify all cases as L, then I will have 90% accuracy, 90% Precision on L and 0% recall on S.
- Recall is among all S, how many are identified.
- Low recall on S is the problem and also we care more about that.
- We want to catch criminals, identify diseases.



The performance on the test set is as follows.

precision on $L$	recall on $S$
0.90	0.12

# Solution 1: Weighting

- In many algorithms, you are allowed to input a weight vector for each row of your dataset. Let's say the weight is 2 for example, then the error rate on those rows with weight "2" will be counted twice.

$$-\sum_{j \in \mathcal{C}} \sum_{y_i = j} \ln(P(y_i = j | x_i; \theta))$$

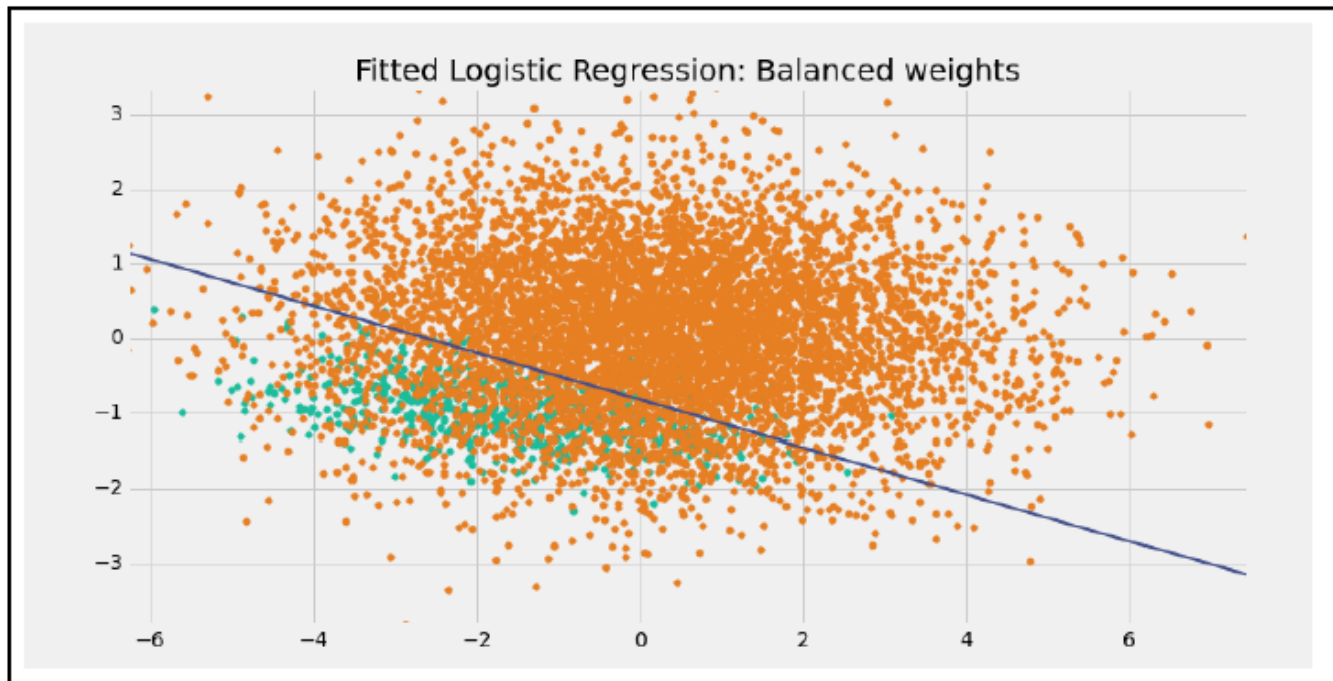


$$-\sum_{j \in \mathcal{C}} \sum_{y_i = j} w_j \ln(P(y_i = j | x_i; \theta))$$

- In scikit-learn, this can be done for supported classifiers using the 'class weight' parameter.

# Weighting

After we increase the weight by 9 times, the results are quite good.



precision on $L$	recall on $S$
0.98	0.89



## Solution 2: Under-sampling

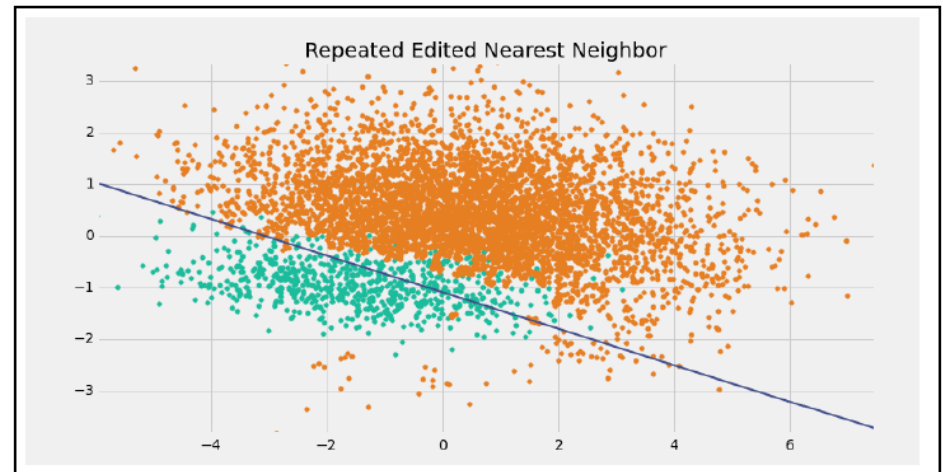
- There are many ways to do under-sampling the majority cases. We have 90% to 10%. You can sample to 20% (2 out of 9) vs 10% or even 10% vs 10% (1 out of 9).
- Random under-sampling=> to 2:1 =>

precision on $L$	recall on $S$
0.97	0.82

- The reference, tried 7 other under-sampling methods. Almost all did not outperform random under-sampling.

# Solution 3: Under-sampling

- Among the methods of under-sampling, most ideas is to remove the cases depending on how far it is from the  $S$  cases.
- The one works the best in this experiment is "repeated KNN". We remove majority cases if it is different from the majority voting of the KNN result (5 nearest neighbor for example).
- We repeat until no  $L$  case can be removed.



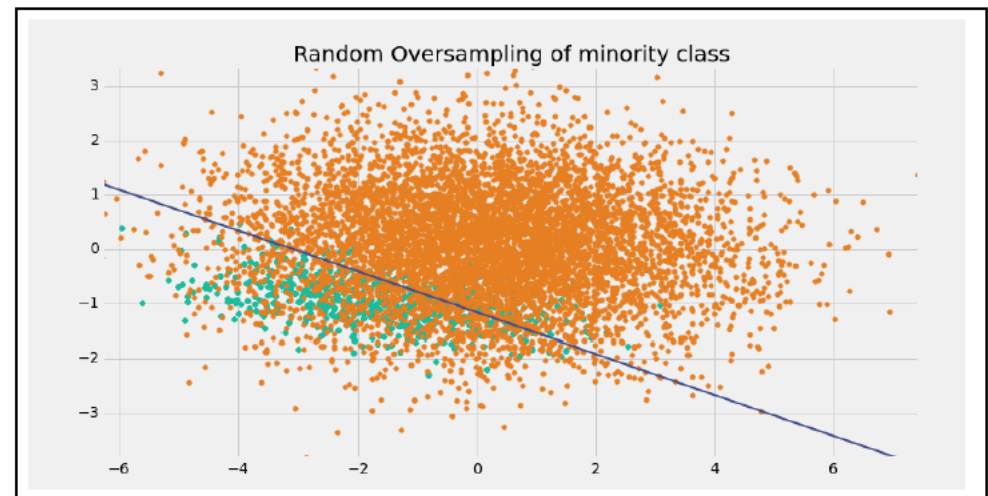
precision on $L$	recall on $S$
0.97	0.80

## Solution 4

- Similarly, we can over-sample the minority cases  $S$ . The problem is different from down-sampling in which we drop rows, in over-sampling, we need to create rows.
- The simplest idea is we sample with replacement randomly from  $S$  cases.

	$ L $	$ S $
Before resampling	6320	680
After resampling	6320	3160

precision on $L$	recall on $S$
0.97	0.76



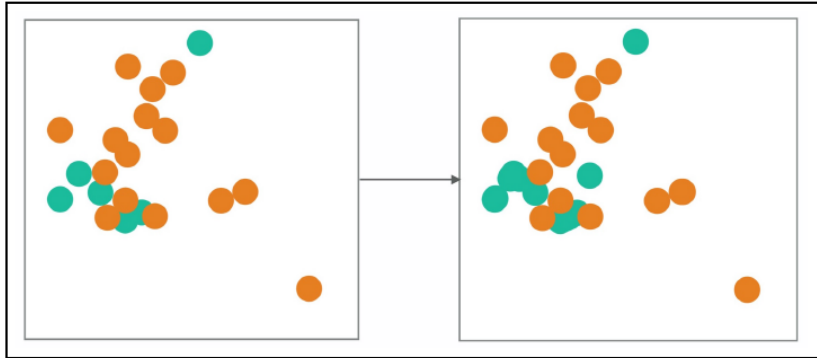
# Solution 5: SMOTE

- This is one of the most famous method for handling unbalanced dataset.
- However, from my experience and also in this experiment, SMOTE only delivers average performance.
- The SMOTE algorithm is

For each point  $p$  in  $S$ :

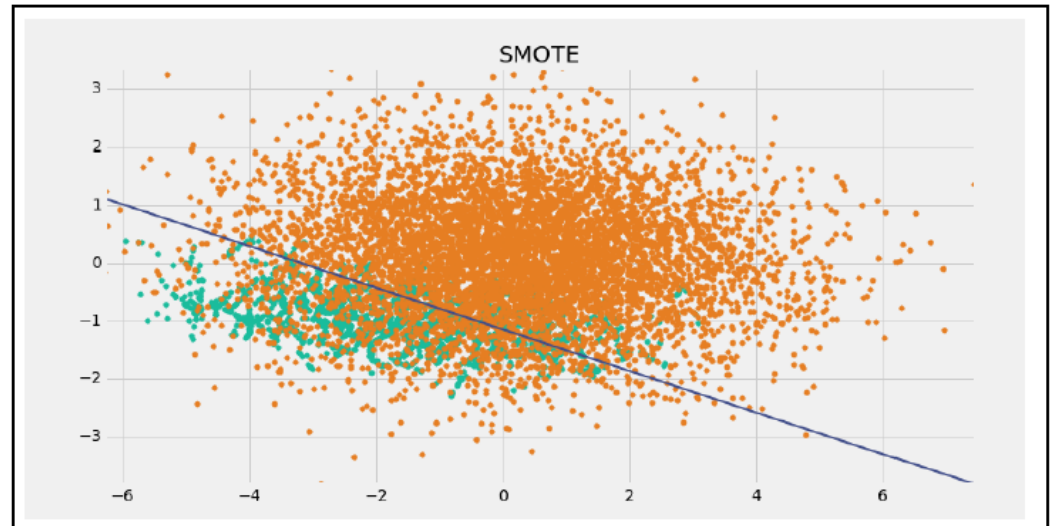
1. Compute its  $k$  nearest neighbors in  $S$ .
2. Randomly choose  $r \leq k$  of the neighbors (with replacement).
3. Choose a random point along the lines joining  $p$  and each of the  $r$  selected neighbors.
4. Add these synthetic points to the dataset with class  $S$ .

# Solution 5: SMOTE



	$ L $	$ S $
Before resampling	6320	680
After resampling	6320	3160

The reference provides two more variants of SMOTE and the performance are similar.



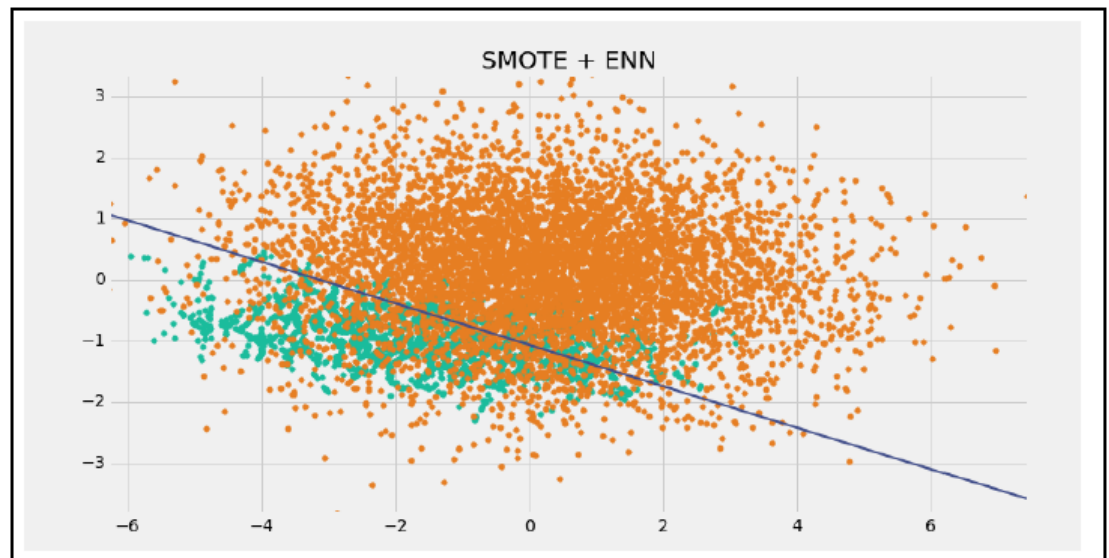
precision on $L$	recall on $S$
0.97	0.77

# Solution 6: Both Samplings

1. We can use under-sampling and over-sampling methods at the same time to make the label distribution more balanced.
2. The paper reported two combinations and one delivers very good performance.
3. SMOTE + ENN

	$ L $	$ S $
Before resampling	6320	680
After resampling	4894	3160

precision on $L$	recall on $S$
0.97	0.92



# Solution 7: BalanceCascade

- There are many other (sophisticated) methods proposed by professors.
- The best performing one in this survey article is “BalanceCascade”.

1. Set  $t = r^{\frac{1}{N-1}}$
2. For  $i = 1, \dots, N$ :
  - (a) Randomly sample a subset  $L_i$  of  $L$  such that  $|L_i| = |S|$ .
  - (b) Learn an AdaBoost ensemble using  $L_i$  and  $S$
3. Undersample  $L$  to remove points correctly classified by  $F_i$ .
4. Combine the above classifiers into a meta-ensemble

$$F_i(x) = \text{sgn}(\sum_{j=1}^{n_i} w_{ij} f_{ij}(x) - b_i)$$

$$F(x) = \text{sgn}(\sum_{i=1}^N (\sum_{j=1}^{n_i} w_{ij} f_{ij}(x) - b_i))$$

precision on $L$	recall on $S$
0.99	0.91

# Conclusion

- It seems using both over-sampling and under-sampling together can achieve good performance and those are simple solutions.
- Similar to other data mining problems, there are many other solutions and some methods work well whereas some methods did not.
- SMOTE is the most famous one and you can try. But better with under-sampling method used together.
- If this is important to your problem, you can try advanced methods (including BalanceCascade) listed in this paper.