Association rule mining Rule strength measure

$$\text{support}(X \to Y) = \frac{\text{count}(X \cup Y)}{n}$$

$$\text{confidence}(X \to Y) = \frac{\text{count}(X \cup Y)}{\text{count}(X)}$$

1 Apriori frequent itemset algorithm

init-pass obtains the items in lexicographic order.

```
Algorithm Apriori(T)
   C1 ← init-pass(T);
   F1 ← {f | f ∈ C1, f.count/n ≥ minsup};    // n: no. of transactions in T
   for (k = 2; Fk-1 ≠ ∅; k++) do
        Ck ← candidate-gen(Fk-1);
        for each transaction t ∈ T do
           for each candidate c ∈ Ck do
               if c is contained in t then
                  c.count++;
           end
        end
        Fk ← {c ∈ Ck | c.count/n ≥ minsup}
   end
return F ← ∪k Fk;
```

```
Function candidate-gen(Fk-1)
   Ck ← ∅;
   forall f1, f2 ∈ Fk-1
        with f1 = {i1, … , ik-2, ik-1}
        and f2 = {i1, … , ik-2, i'k-1}
        and ik-1 < i'k-1 do
      c ← {i1, …, ik-1, i'k-1};               // join f1 and f2
      Ck ← Ck ∪ {c};
      for each (k-1)-subset s of c do
        if (s ∉ Fk-1) then
           delete c from Ck;                  // prune
      end
   end
   return Ck;
```

2 MS Apriori

sort in ascending

```
Algorithm MSapriori(T, MS, φ)        // φ is for support difference constraint
   M ← sort(I, MS);
   L ← init-pass(M, T);
   F1 ← {{i} | i ∈ L, i.count/n ≥ MIS(i)};
   for (k = 2; Fk-1 ≠ ∅; k++) do
         if k=2 then
           Ck ← level2-candidate-gen(L, φ)
         else Ck ← MScandidate-gen(Fk-1, φ);
         end;
         for each transaction t ∈ T do
            for each candidate c ∈ Ck do
                if c is contained in t then
                     c.count++;
                if c – {c[1]} is contained in t then
                    c.tailCount++
            end
         end
        Fk ← {c ∈ Ck | c.count/n ≥ MIS(c[1])}
   end
   return F ← ∪kFk;
```

Figure 1: MsApriori

data set T and the sorted items M, to produce the seeds L for generating candidate itemsets of length 2, i.e., C_2. init-pass() has two steps:

1. It first scans the data once to record the support count of each item.
2. It then follows the sorted order to find the first item i in M that meets MIS(i). i is inserted into L. For each subsequent item j in M after i, if $j.count/n \geq$ MIS(i), then j is also inserted into L, where $j.count$ is the support count of j, and n is the total number of transactions in T.

Figure 2: Init Pass

```
Function level2-candidate-gen(L, φ)
  C2 ← ∅;                                   // initialize the set of candidates
  for each item l in L in the same order do
     if l.count/n ≥ MIS(l) then
        for each item h in L that is after l do
           if h.count/n ≥ MIS(l) and |sup(h) – sup(l)| ≤ φ then
              C2 ← C2 ∪ {{l, h}};  // insert the candidate {l, h} into C2
```

Fig. 2.7. The level2-candidate-gen function

```
Function MScandidate-gen(Fk–1, φ)
   Ck ← ∅;                                  // initialize the set of candidates
   forall f1, f2 ∈ Fk                       // find all pairs of frequent itemsets
     with f1 = {i1, … , ik–2, ik–1}         // that differ only in the last item
     and f2 = {i1, … , ik–2, i'k–1}
     and ik-1 < i'k–1 and |sup(ik-1) – sup(i'k–1)| ≤ φ do
       c ← {i1, …, ik–1, i'k–1};            // join the two itemsets f1 and f2
       Ck ← Ck ∪ {c};                       // insert the candidate itemset c into Ck
       for each (k–1)-subset s of c do
          if (c[1] ∈ s) or (MIS(c[2]) = MIS(c[1])) then
             if (s ∉ Fk–1) then
                delete c from Ck;           // delete c from the set of candidates
       endfor
   endfor
   return Ck;                               // return the generated candidates
```

Fig. 2.8. The MScandidate-gen function

Figure 3: Candidate Gen MS

3 GSP (sequential pattern mining)

GSP mining algorithm

- Very similar to the Apriori algorithm

```
Algorithm GSP(S)
   C1 ← init-pass(S);                            // the first pass over S
   F1 ← {⟨{f}⟩| f ∈ C1, f.count/n ≥ minsup};     // n is the number of sequences in S
   for (k = 2; Fk–1 ≠ ∅; k++) do                 // subsequent passes over S
     Ck ← candidate-gen-SPM(Fk–1);
     for each data sequence s ∈ S do             // scan the data once
        for each candidate c ∈ Ck do
           if c is contained in s then
             c.count++;                          // increment the support count
        end
     end
     Fk ← {c ∈ Ck | c.count/n ≥ minsup}
   end
   return ∪k Fk;
```

Fig. 12. The GSP Algorithm for generating sequential patterns

Function candidate-gen-SPM(F_{k-1})

1. **Join step**. Candidate sequences are generated by joining F_{k-1} with F_{k-1}. A sequence s_1 joins with s_2 if the subsequence obtained by dropping the first item of s_1 is the same as the subsequence obtained by dropping the last item of s_2. The candidate sequence generated by joining s_1 with s_2 is the sequence s_1 extended with the last item in s_2. There are two cases:
   - the added item forms a separate element if it was a separate element in s_2, and is appended at the end of s_1 in the merged sequence, and
   - the added item is part of the last element of s_1 in the merged sequence otherwise.

   When joining F_1 with F_1, we need to add the item in s_2 both as part of an itemset and as a separate element. That is, joining $\langle\{x\}\rangle$ with $\langle\{y\}\rangle$ gives us both $\langle\{x, y\}\rangle$ and $\langle\{x\}\{y\}\rangle$. Note that x and y in $\{x, y\}$ are ordered.
2. **Prune step**. A candidate sequence is pruned if any one of its (k–1)-subsequence is infrequent (without minimum support).

Fig. 13. The candidate-gen-SPM() function

When he says is the same as the subsequence... Means that you have to remove the parenthesis and see if the elements are in the same order.

4 Index for Classifiers

Metric	Description / Formula
Accuracy	$\frac{\text{Correctly Classified Examples}}{\text{Total Examples}}$
Precision	$\frac{TP}{TP+FP}$: Of predicted positives, how many are correct.
Recall/sensitivity (TPR)	$\frac{TP}{TP+FN}$: Of actual positives, how many are identified.
F_1 Score	$2 \times \frac{p \cdot r}{p+r}$: Combines precision and recall.
Specificity (TNR)	$\frac{TN}{TN+FP}$: True negative rate.
False Positive Rate (FPR)	$1 - \text{TNR}$
ROC Curve	Plot: FPR (x-axis) vs. TPR (y-axis).

Table 1: Summary of Classification Metrics

5 Decision trees

```
. Algorithm decisionTree(D, A, T)
    if D contains only training examples of the same class c_j ∈ C then
        make T a leaf node labeled with class c_j;
    elseif A = ∅ then
        make T a leaf node labeled with c_j, which is the most frequent class in D
    else  // D contains examples belonging to a mixture of classes. We select a single
          // attribute to partition D into subsets so that each subset is purer
        p_0 = impurityEval-1(D);
        for each attribute A_i ∈ {A_1, A_2, ..., A_k} do
            p_i = impurityEval-2(A_i, D)
        end
        Select A_g ∈ {A_1, A_2, ..., A_k} that gives the biggest impurity reduction,
            computed using p_0 − p_i;
        if p_0 − p_g < threshold then     // A_g does not significantly reduce impurity p_0
            make T a leaf node labeled with c_j, the most frequent class in D.
        else                              // A_g is able to reduce impurity p_0
            Make T a decision node on A_g;
            Let the possible values of A_g be v_1, v_2, ..., v_m. Partition D into m
                disjoint subsets D_1, D_2, ..., D_m based on the m values of A_g.
            for each D_j in {D_1, D_2, ..., D_m} do
                if D_j ≠ ∅ then
                    create a branch (edge) node T_j for v_j as a child node of T;
                    decisionTree(D_j, A-{A_g}, T_j)// A_g is removed
                end
            end
        end
    end
```

Entropy D is the dataset. C are the different classes in the dataset.

$$\text{entropy}(D) = -\sum_{j=1}^{|C|} \Pr(C_j) \log_2 \Pr(C_j)$$

Is a positive value. The sum of the probabilities of the classes is 1. The more the data is purer, the more the entropy value is close to zero. Worst entropy is 1 (all the classes are equally distributed).

$$\text{entropy}_{A_i}(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \text{entropy}(D_j)$$

This value is the entropy if we choose to partition the data over attribute A. There are v possible values for the attribute A. Entropy(D_j) is the entropy in the subset of data that has the value v_j for the attribute A.

$$\text{gain}(D, A_i) = \text{entropy}(D) - \text{entropy}_{A_i}(D)$$

The higher the gain the better

6 Naive based classification

Product rule

$$\Pr(a1, a2) = \Pr(a1)\Pr(a2|a1)$$

Product rule for conditional probability

$$\Pr(a1, a2|c) = \Pr(a2|c)\Pr(a2|a1, c)$$

General case

$$\Pr(a_1 \cdots a_n \mid c) = \Pr(a_1 \mid c)\Pr(a_2 \mid c, a_1) \cdots \Pr(a_n \mid c, a_1 \cdots a_{n-1})$$

Conditional independence assumption

$$\Pr(a_i \mid c, a_1 \cdots a_{i-1}) = \Pr(a_i \mid c)$$

Goal:

$$\Pr(c \mid (a_1 \cdots a_n)) = \frac{\Pr(c)\Pr((a_1 \cdots a_n) \mid c)}{\Pr(a_1 \cdots a_n)}$$

Using the law of total probability

$$\Pr(a_1 \cdots a_n) = \sum_{r=1}^{|C|} \Pr(c)\Pr((a_1 \cdots a_n) \mid c)$$

Using the product rule and the conditional independence assumption

$$\Pr(c \mid (a_1 \cdots a_n)) = \frac{\Pr(c) \prod_{i=1}^{n} \Pr(a_i \mid c)}{\sum_{r=1}^{|C|} \Pr(c) \prod_{i=1}^{n} \Pr(a_i \mid c)}$$

Adjusting the probability to account for attribute values that don't occur with that class.

$$\Pr(A_i = a_i \mid C = c_j) = \frac{n_{ij} + \lambda}{n_j + \lambda m_i}$$

Instead if there are missing values we just ignore them and use what we have

- n_j: # examples with $C=c_j$ in training data
- n_{ij}: # examples with both $A_i=a_i$ and $C=c_j$
- m_i: # possible values of attribute A_i.
- Normally, we use λ= 1

7 Naive based text classification

$$\Pr(c|d) = \frac{\Pr(c)\Pr(d|c)}{\Pr(d)}$$

$$= \frac{\Pr(c) \prod_{k=1}^{\text{all word in d}} \Pr(w_k|c)}{\sum_{r=1}^{|C|} \Pr(c_r) \prod_{k=1}^{\text{all word in d}} \Pr(w_k|c)}$$

$$\Pr(c) = \frac{\sum_{i=1}^{|D|} \Pr(c|d_i)}{|D|}$$

$$\Pr(w_k|c) = \frac{\sum_{i=1}^{|D|} N_{ik}\Pr(c|d_i)}{\sum_{s=1}^{|V|} \sum_{i=1}^{|D|} N_{si}\Pr(c|d_i)}$$

N_{ij} is the number of times that the word j appears in a document i.

8 K-means

```
Algorithm k-means(k, D)
Choose k data points as the initial centroids (cluster centers)
repeat
    for each data point x ∈ D do
        compute the distance from x to each centroid;
        assign x to the closest centroid        // a centroid represents a cluster
    endfor
    re-compute the centroids using the current cluster memberships
until the stopping criterion is met
```

9 Agglomerative clustering

```
Algorithm Agglomerative(D)
Make each data point in the data set D a cluster,
Compute all pair-wise distances of x1, x2, …, xn ∈ D;
repeat
    find two clusters that are nearest to each other;
    merge the two clusters form a new cluster c;
    compute the distance from c to all other clusters;
until there is only one cluster left
```

Distance metrics:

- Single link: distance between two clusters is the distance between two closest data points in the two clusters.
- Complete link: distance between two clusters is the distance of two furthest data points in the two clusters.
- Average link: average distance between all points in the two clusters. (most precise metrics but also most computationally expensive)
- Centroid link: distance between the centroids of the two clusters. (doesn't consider the shape of the cluster)

10 LU learning

11 Search Engines

11.1 Transition Probability Matrix

Transition probability matrix:

$$A_{i,j} = \begin{cases} \frac{1}{\text{outdegree}(i)} & \text{if there is a link from } i \text{ to } j, \\ 0 & \text{otherwise.} \end{cases}$$

stochastic matrix: the sum of the elements in each row is 1.

we make stochastic by assigning an equal probability to each link for rows that are all zeros. or by removing the node without outgoing edges.

Irreducible An adjacency matrix is irreducible if the graph it represents is *strongly connected*, meaning there is a path between every pair of nodes in the graph.

Aperiodic A state i in a Markov chain is periodic with period $k > 1$ if k is the smallest number such that all paths leading from state i back to state i have a length that is a multiple of k.

A graph is *aperiodic* if there are no cycles that you cannot escape from.

Making the Matrix Aperiodic and Irreducible To ensure the matrix is both aperiodic and irreducible We assign a small probability that the user will jump to a random page. (we add a damping factor)

Final formula for page rank

$$P = (1 - d)e + dA^T P$$

12 Finding holes

Use tree. Add uniformly sampled points with class Hole. Perform the regular decision tree. We add a different number of N points at each node.

- The number of N points for the current node E is determined by the following rule (note that at the root node, the number of inherited N points is 0):

1. If the number of N points inherited from the parent node of E is less than the number of Y points in E, then:
   - The number of N points for E is increased to the number of Y points in E.
2. Else:
   - The number of inherited N points is used for E.

13 EM (expectation maximization)

Train a classifier with only the labeled documents. Use it to probabilistically classify the unlabeled documents. Use ALL the documents to train a new classifier. Iterate steps 2 and 3 to convergence.

14 Recommender systems

Matrix factorization Users and movies are categorized in a latent space that is composed by a fixed number of features (could be the genre, star actor). We obtain two matrixes, one for users and one for movies. The probability that an user likes the movie is the sum of product of the user and movie features.

$$P_{i,j} = U_i^T \cdot M_y$$

Where U and I are column vectors.

The learning rule is

$$u_{ki}^{t+1} = u_{ki}^t + 2\gamma(r_{ij} - p_{ij})m_{kj}^t$$

$$m_{kj}^{t+1} = m_{kj}^t + 2\gamma(r_{ij} - p_{ij})u_{ki}^t$$

15 Sentiment quintuple

Holder, time, entity, aspect, sentiment.

16 General definitions

Continual learning: Continual learning is the ability of a model to learn new tasks incrementally without forgetting previously learned tasks. **Class-incremental learning**: Class-incremental learning involves learning new classes incrementally while retaining the ability to classify previously learned classes. **Task-incremental learning**: Task-incremental learning involves learning new tasks sequentially, where task identity is provided during inference. **Inter-task class separation**: Inter-task class separation refers to maintaining distinct boundaries between classes from different tasks to prevent interference. **Objectives of continual learning**: The two main objectives are avoiding catastrophic forgetting and promoting knowledge transfer between tasks. **Closed world learning**: Closed world learning assumes the model only encounters data belonging to predefined classes. **Open-world learning, on-the-job learning, and continual learning after deployment**: These involve learning from data incrementally after deployment, adapting to new classes, and handling open-set scenarios. **Out-of-distribution detection**: Out-of-distribution detection identifies inputs that differ significantly from the training data's distribution. **CML (Continual Meta-Learning)**: CML enables a model to quickly adapt to new tasks using prior knowledge while mitigating forgetting. **Self-initiated open-world continual learning and adaptation**: This framework involves a model autonomously identifying and learning from novel classes in an open-world scenario