### Kind of recommendation

+ newest recommendation

+ uniform recommendation

+ categorical recommendation

+ nearest (similarity) recommendation: compare item to item (Item >< item matrix)

+ most-relevant (preference) recommendation : compare items with user’s preference (item >< user matrix)

### SBML

~~+ similarity-based recommendation: based on similar score~~

+ kNN classification

+ kernel kmean clustering

### Crawling code review

+ VNU-movies-13k

+ VNU-songs-99k

# Recommendation

### uniform recommendation mechanism (Instagram)

+ you have 10k stocked items in database

+ you have 20 slots in screen to display

+ you have to pick 20 items (corresponding 20 slots) out of 10k stock items to display.

+ Instagram method: pick randomly based on user’s choice

+ pick randomly with uniform distribution to show new items every time user refresh

|  |
| --- |
| **public** List<TYPE> pickItems(**int** kpar) **throws** Exception  {  List<TYPE> res = **new** ArrayList<TYPE>();    File[] files = dataFolder.listFiles();  **if**(files == **null**) **return** res;      **while**(**true**)  {  **int** rk = coin.nextInt( files.length );    TYPE ik = readItem(files[rk]);  res.add(ik);    **if**(res.size() >= kpar) **break**;  }    **return** res;  } |
| + generate a random number from [0, n) where n is the number of item  + read and add the item into the recommendation list  + return the list |

### Categorical recommendation

+ pick items randomly with categorical distribution (not uniform distribution)

+ categorical distribution is based on user’s preferences

pC = { item1 -> p1, item2 -> p2, .., itemN -> pN }

pj = similar(itemj, userPref);   
itemj = { action, movie, drama, x-men, sci-fi }  
userPref = { action, comedy }

### Pick item from map randomly / uniformly

|  |
| --- |
| **public** **static** Random *coin* = **new** Random(197);    **private** **static** String nextKey(Map<String, Double> items)  {  **double** r = *coin*.nextDouble(), s = 0, ds = 1d/items.size();    **for**(String nk: items.keySet())  {  s += ds;  **if**(r < s) **return** nk;  }    **return** **null**;  } |

### Pick item from map with categorical distribution

https://en.wikipedia.org/wiki/Categorical\_distribution

|  |
| --- |
| **public** **static** Random *coin* = **new** Random(197);  **private** **static** String nextKeyCat(Map<String, Double> items)  {  **double** r = *coin*.nextDouble(), s = 0;    **for**(String nk: items.keySet())  {  s += items.get(nk);  **if**(r < s) **return** nk;  }    **return** **null**;  } |

### Writing a search engine in 5 minutes

+ Using lucene <https://lucene.apache.org/core/>

|  |
| --- |
| **public** **static** **void** main(String[] args)  {  List<String> stocked = RandUtils  .*buildList*("action", "drama", "comedy", "comic", "active");    ClaireSearchEngine<String> eng = **new** ClaireSearchEngine<String>()  {  JaccardTitleMatcher eng = **new** JaccardTitleMatcher();    @Override  **public** **double** jaccardIndex(String sk, String q)  {  **return** eng.jaccardSimilarScore(sk, q);  }    };    List<ClaireSearchResult<String>> results = eng.findSimilarItems("activ", stocked);    **for**(ClaireSearchResult<String> rk: results)  {  System.***out***.println(rk.score + ": " + rk.document);  }    } |
| **public** **class** JaccardTitleMatcher  {  **private** String \_\_left;  **private** Map<String, Integer> \_\_leftSet = **null**;    **private** String \_\_right;  **private** Map<String, Integer> \_\_rightSet = **null**;  **public** **double** jaccardSimilarScore(String l, String r)  {  //do not recompute grams set if the query is the same  **if**(\_\_left != l) { \_\_left = l; \_\_leftSet = gramSet(l); }  **if**(\_\_right != r) { \_\_right = r; \_\_rightSet = gramSet(r); }    //jaccard index for histogram  Map<String, Integer> s1 = \_\_leftSet;  Map<String, Integer> s2 = \_\_rightSet;    **int** c = 0, s = 0;      Set<String> cm = **new** TreeSet<String>();  cm.addAll(s1.keySet());  cm.addAll(s2.keySet());    **for**(String x: cm)  {  Integer x1 = s1.get(x);  **if**(x1 == **null**) x1 = 0;    Integer x2 = s2.get(x);  **if**(x2 == **null**) x2 = 0;    c += Math.*min*(x1, x2);  s += Math.*max*(x1, x2);  }    **return** c/(**double**)(s==0 ? 1 : s);  }  **private** Map<String, Integer> gramSet(String l)  {  Map<String, Integer> res = **new** TreeMap<String, Integer>();    **for**(**int** n=l.length(), k=0; k<n; k++)  {  String c1 = "" + (**char**)l.charAt(k), c2 = **null**, c3 = **null**;  add(c1, res);    **if**(k+1<n)  {  c2 = c1 + (**char**)l.charAt(k);  add(c2, res);  }    **if**(k+2<n)  {  c3 = c2 + (**char**)l.charAt(k);  add(c3, res);  }  }      **return** res;  }  **private** **void** add(String v, Map<String, Integer> res)  {  Integer c = res.get(v);  res.put(v, c==**null** ? 1 : c+1);  }  } |

### Matching titles

+ matching documents: s1 = wordset(d1) ; s2 = wordset(d2); j12 = jaccardScore(s1, s2)

+ matching titles: we need to use smaller units/entities (character uni-gram, bi-gram, tri-gram)  
“matrix the” >< “the matrix”

+ matching titles in equation: s1 = gramset(d1) ; s2 = gramset(d2); j12 = jaccardScore(s1, s2)

# Crawling

### Two datasets

### Review DucAnh’s code

### Starting the catalog crawling for Thao

<http://www.lyrics.vn/lyrics/1534-thien-thai.html>

<http://www.azlyrics.com/lyrics/aerosmith/idontwanttomissathing.html>