**Recommendation System Architecture**

\*Designing the content-based recommendation engine text mining.

Content-based recommendation engine works by creating a similarity index. Think of this index as a matrix, with rows and columns as the articles and the cell value storing the similarity between the articles. The diagonal values of this matrix will be one. The similarity score is a value between zero and one.

**Architecture:**

* Preprocessing the News article:

Similarity Index

Similarity DB

News Articles

* Finding Sub Categerory:

Additional Features

Extracting

Sub category

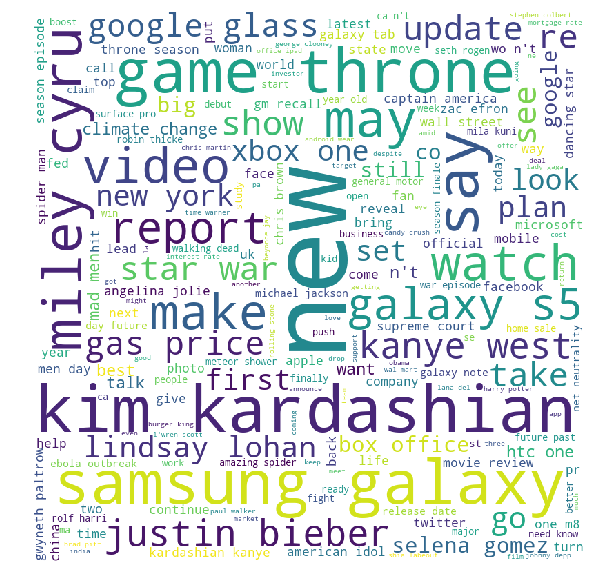
Similarity DB

**Additional Features (Recommended Features)**

For Memory extraction dependency, subcategory is explored for Business category for below methods:

**News extraction based on category type:**

Basic Text preprocessing has implemented the Title feature and extracted the words commonly used words from word cloud option.



1. Converting all letters to lower or upper case
2. Converting numbers into words or removing numbers
3. Removing punctuations, accent marks and other diacritics
4. Removing white spaces
5. Expanding abbreviations
6. Removing stop words, sparse terms, and particular words
7. Tokenize (WordPunctTokenizer)the title

* **Feature extract & Creating Sub category:**
  + **TDIDF** : TFIDF is a very popular weighting metric used in text mining (CountVectorizer, TfidfVectorizer)
  + **Cosine similarity:** Cosine similarity is non-invariant to changes in the magnitude of values. That is, if in one of the vectors we increase the value of its members, the cosine similarity will change. We need this behavior, as the vector contains the tfidf scores. A change in tfidf score means there is a change in the document. The document is no longer the same one we had before. Pearson coefficient is invariant to shifts in the vector. Hence for comparing similarities between documents, we use cosine as the metric.
  + **Document Similarity.**
  + **Topic Models: LatentDirichletAllocation:** Non-negative Matrix Factorization is applied with two different objective functions: The latter is equivalent to Probabilistic Latent Semantic Indexing.
  + Clustering documents using topic model features and further **creating sub category**, as **Stock, banking, industry and others** .
* **More Features for Recommendation System:**
  + **Polarity** Each word is looked up in the lexicon; positive and negative words are tagged with +1 and -1 respectively. Let's call the words which have received a score the polarized words. Not all words receive a score. Only those found in the lexicons receive a score. We can pass a customer lexicon through the polarity\_dt parameter to the sentiment function. For each of the polarized words, n words before them and n words after them are considered, and together they are called polarized context clusters.
  + **Jaccard's distance:** The Jaccard index measures the similarity between two sets, and is a ratio of the size of the intersection and the size of the union of the participating sets. Here we have only have two elements, one for publisher and one for category, so our union is 2. The numerator, by adding the two Boolean variable, we get the intersection.
* **Feature plan to implement to Apriori algorithm Rank base system**

Based on the cosine similarity, Jaccard and Polarity difference, compute rank by Low Normal high ranking. High ranking article recommended to users.

Top Ranked Feature

Apriori Algorithm

Additional Features

Support, confidence, lift

* **KNN Algorithm pseudo code:**

To find the clustering based on eucledian distance based.

import numpy as np

from sklearn.cross\_validation import train\_test\_spli

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X,Y,train\_size=0.9)

y\_pred\_knn = []

for val in x\_test:

euc\_dis = []

for point in x\_train:# eucledian distance

euc\_dis.append(((val[0]-point[0])\*\*2+(val[1]-point[1])\*\*2)\*\*0.5)

temp\_target = y\_train.tolist()

for i in range(len(euc\_dis)):

for j in range(0,len(euc\_dis)-i-1):

if(euc\_dis[j+1] < euc\_dis[j]):

euc\_dis[j], euc\_dis[j+1] = euc\_dis[j+1], euc\_dis[j]

temp\_target[j], temp\_target[j+1] = temp\_target[j+1], temp\_target[j]

vote = [0,0,0]

for i in range(3):

vote[temp\_target[i]] += 1

y\_pred\_knn.append(vote.index(max(vote)))

print('Accuracy:',accuracy\_score(y\_test,y\_pred\_knn))