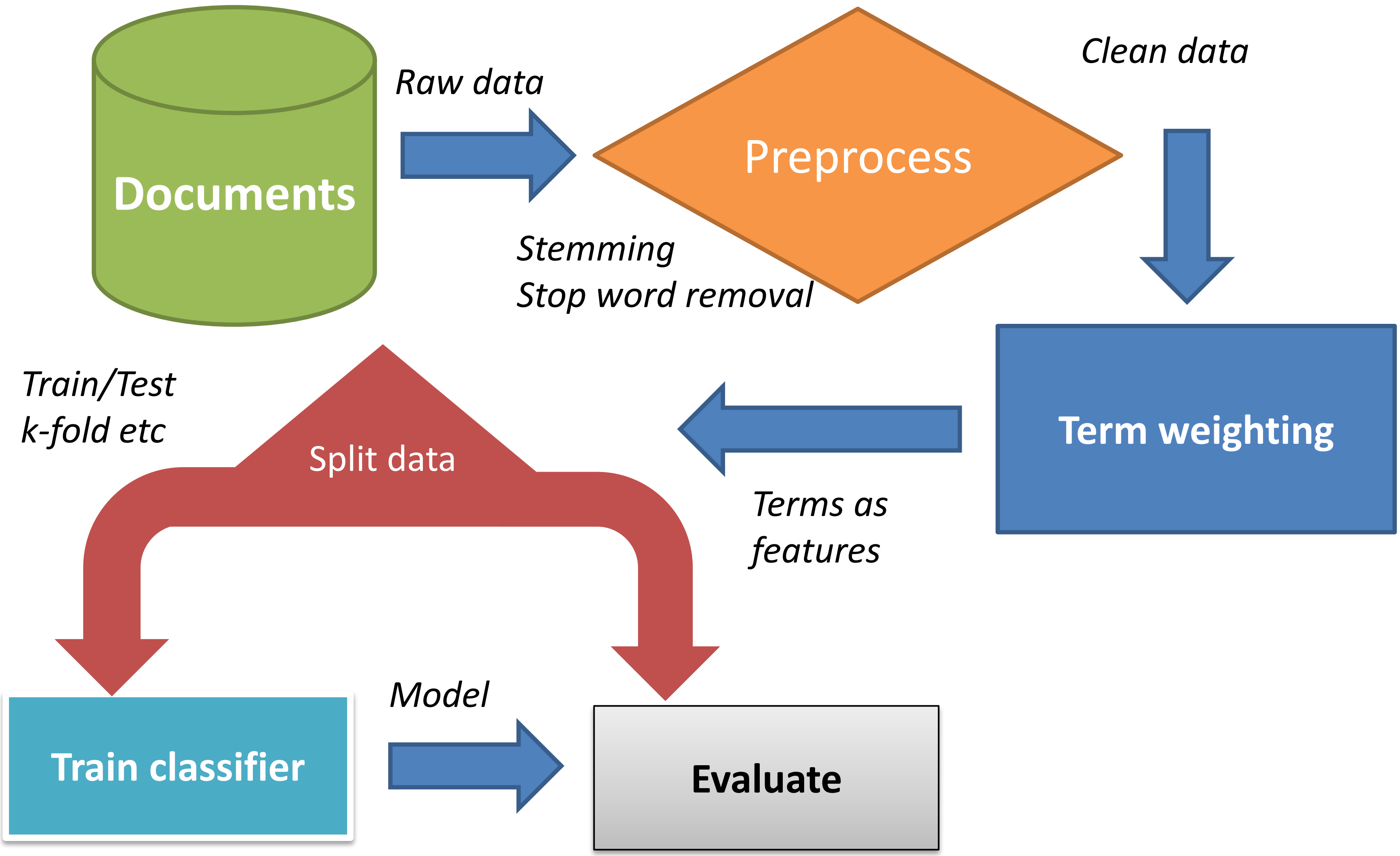
Graph of Words

Graph Based Features for Text Classification

The process


Documents
Raw data
Preprocess
Clean data
Stemming
Stop word removal
Term weighting
Terms as
features
Train/Test
k-fold etc
Split data
Model
Train classifier
Evaluate

Text classification

- Document collection $Loc = \{D_1, D_2, \dots, D_n\}$
  - **N** docs Labeled with a class
- Dictionary (of the collection):
  - All *useful* terms in the collection
  - $T = \{t_1, t_2, \dots, t_m\}$
- Vector Space Model
  - Each document is represented by a vector of **m** weights
    - One for each term
- **How to define the weights?**
  1. Bag of words (TF-IDF)
  2. Graph of Words (TW-IDF)
- The weights are features of each document
  - Use any classifier to learn the classes (e.g. SVM)

Scenario

- Documents from Reuters
  - Labeled by hand
- Files:
  - r8-train-stemmed.txt
  - r8-test-stemmed.txt
  - tab separated
    - **Label** \t **text**
- All words are already stemmed

Class Label	# of Train docs	# of Test docs Total	# of docs
acq	1,596	696	2,292
crude	253	121	374
earn	2,840	1,083	3,923
grain	41	10	51
interest	190	81	271
money-fx	206	87	293
ship	108	36	144
trade	251	75	326
Total	**5,485**	**2,189**	**7,674**

- Train a classifier on the two types of features and compare results

Bag of Words

- *tf(f,d):* frequency of t in d
  - **Term importance within document**
- $idf(t, D) = log \frac{N}{|\{d \epsilon D, t \epsilon d\}|}$
  - **Term importance within collection**
- $tf - idf(t, d, D) = tf(f,$d) * $idf(t,D)$
  - Discriminative potential of a term
  - High values :
    - Frequent within documents
    - Infrequent in the entire collection
- Easy in python TF-IDF vectorizer

Evaluation

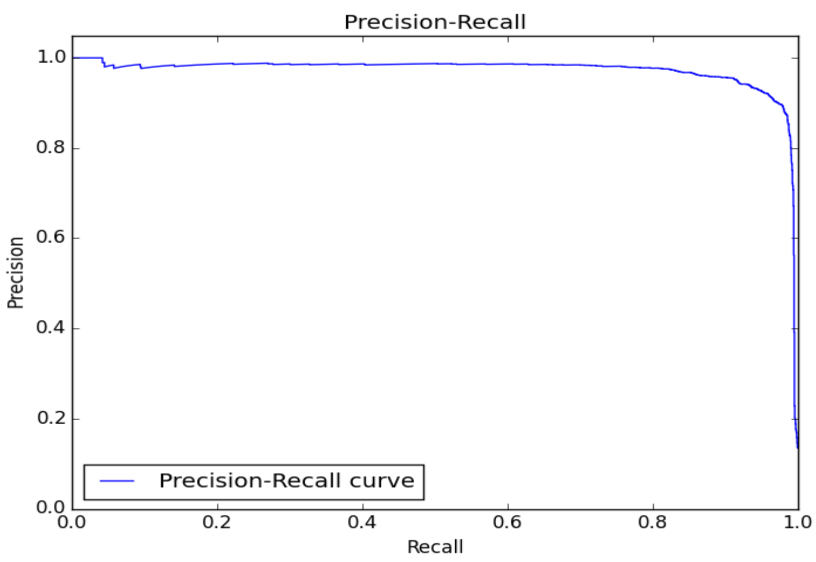
- A classifier can return weights/probabilities per class

Choose (the best) threshold

- Per class
  - $Precision = \frac{TP}{TP+FP}$
  - $Recall = \frac{TP}{TP+FN}$
  - $F1 = 2 * \frac{precision*recall}{precision+recall}$
- Macro-average over all classes

Evaluate all thresholds

- Precision Recall curve

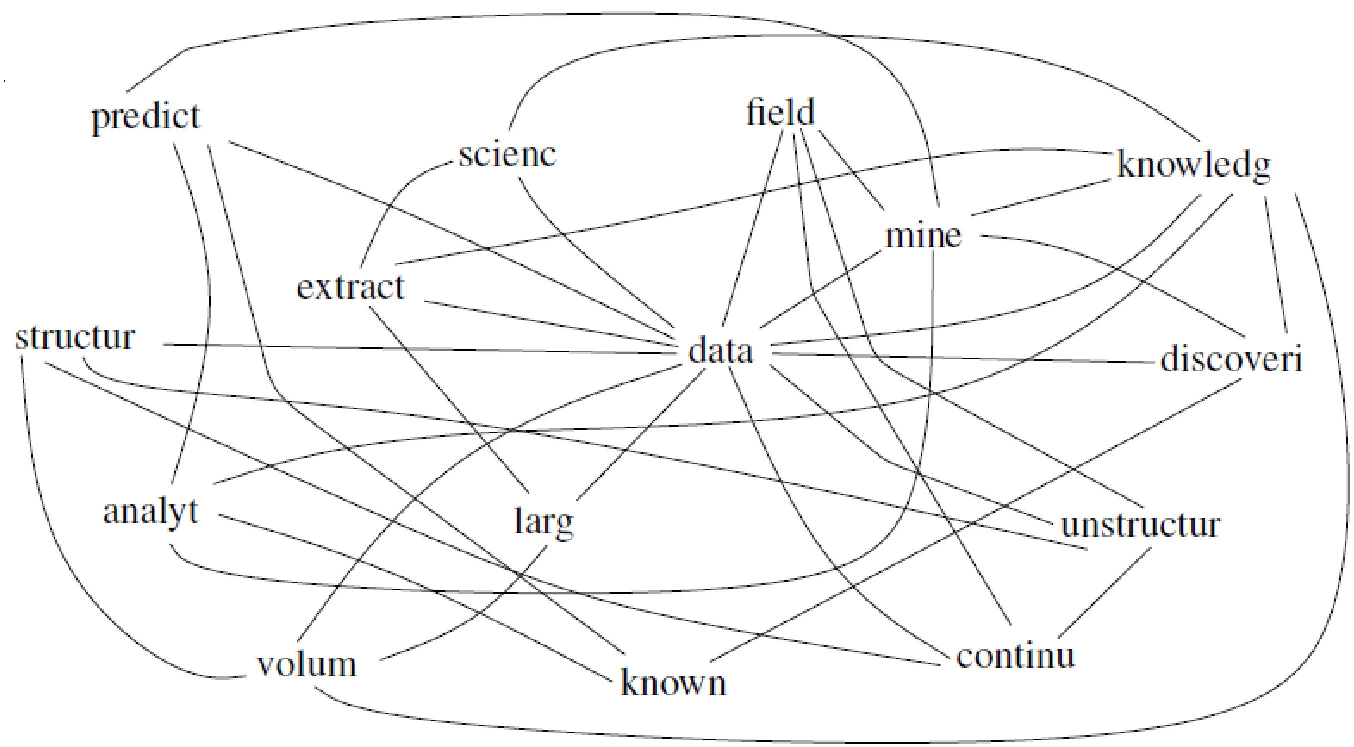


PYTHON TF-IDF

LETS SEE THE CODE

TW-IDF: From Document to Graph

Data Science is the extraction of knowledge from large volumes of data that are structured or unstructured which is a continuation of the field of data mining and predictive analytics, also known as knowledge discovery and data mining.



Sliding Window

For each document $\mathbf{d} \in \mathbf{D}$ *:*
    Initialize a graph $\mathbf{G}$
    For all words $\mathbf{w_i}$ *where* $\mathbf{w_i} \in \mathbf{d}$*:*
        Add $\mathbf{w_i}$ *to* $\mathbf{G}$
        For all words $\mathbf{w_{i+k}}$ *where* $\mathbf{i+k < window\_size}$
            Add $\mathbf{w_{i+k}}$ *to* $\mathbf{G}$
            Add an edge in $\mathbf{G}$ *between* $\mathbf{w_i}$ *and* $\mathbf{w_{i+k}}$

- In Python:
  - networkx.Graph() : initializes a graph object
  - graph_object.has_node(X) : to check if X node exists
    - graph_object.add_node(X)
  - graph_object.has_edge (X,Y): to check if edge between X and Y exists
    - graph_object.add_edge(X,Y)

TW-IDF

- Use graph properties to replace the TF factor
  - i.e. Node degree, PageRank etc.
- Node degree - Centrality :
  - $TW - IDF(t, d, D) = degree(node_t) * idf(t, D)$

François Rousseau and Michalis Vazirgiannis. 2013. Graph-of-word and TW-IDF: new approach to ad hoc IR. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management* (CIKM '13). ACM, New York, NY, USA, 59-68.

Python Code

1. Fill-in in the Code for the TW-IDF calculation
   - File : MyGraph.py
2. Compare the results between TF-IDF and TW-IDF
3. Retry with different window sizes and compare