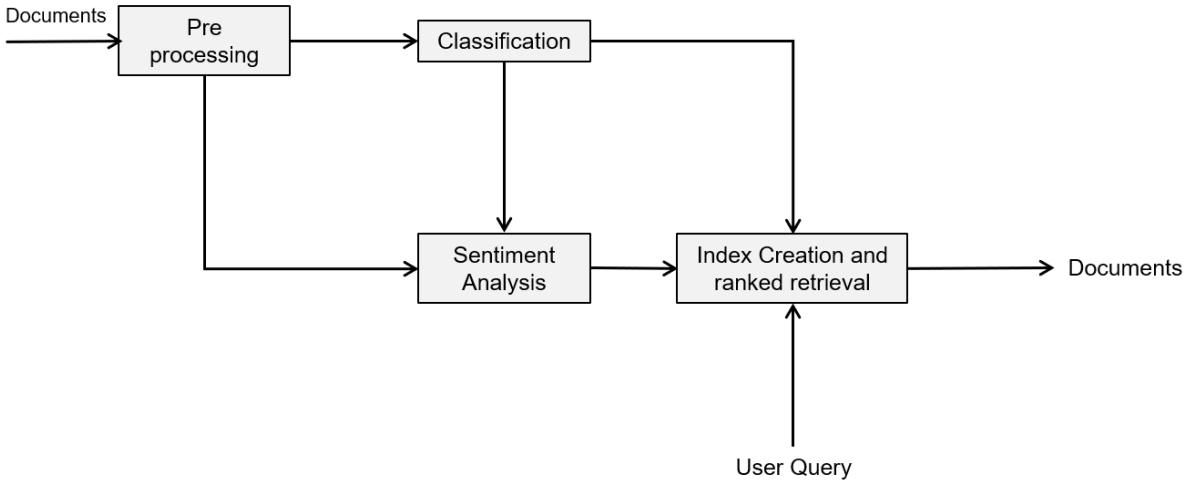
Introduction

There are many news domains, and online versions of newspaper offices and TV stations appearing on the Internet. The ability to access, organize and think critically about the information is becoming both more important and more difficult. Often a search engine is used, both the customized and the universal ones. Besides that, many methods have been applied to analyze the mass news, including news classification, topic detection and special event generation.



Block Diagram of the system

We have built a basic model to classify news articles according to their content using text classification algorithms. We have also performed sentiment analysis on the news articles and assigned a polarity (positive, negative or neutral). We have provided a GUI to input a query to be searched over classified articles and displayed the related news articles matching the query. For example, the input query US President related positive political news returns top-n (n=10) news articles that convey the required information.

Dataset

We have used two news datasets:

1. **AG's Corpus** : AG is a collection of more than 1 million news articles. News articles have been gathered from more than 2000 news sources by ComeToMyHead in more than 1 year of activity. ComeToMyHead is an academic news search engine which has been running since July, 2004. The dataset is provided by the academic community for research purposes in data mining (clustering, classification, etc.), information retrieval (ranking, search, etc), xml, data compression, data streaming, and any other non- commercial activity. The AG's news topic classification dataset is constructed by Xiang Zhang (xiang.zhang@nyu.edu) from the dataset above. It is used as a text classification benchmark in the following paper: Xiang Zhang, Junbo Zhao, Yann LeCun. Character-level Convolutional Networks for Text Classification. Advances in Neural Information Processing Systems 28 (NIPS 2015). The AG's news topic classification dataset is constructed by choosing 4 largest classes (World, Sports, Business, Sci/Tech) from the original corpus. Each class contains 30,000 training

samples and 1,900 testing samples. The total number of training samples is 120,000 and testing 7,600. The attributes in the dataset are:

FIELD	TYPE	NULL VALUE
source	varchar(32)	NO
url	varchar(255)	NO
title	text	YES
image	varchar(255)	YES
category	varchar(32)	NO
description	text	YES
pubdate	timestamp	YES
video	varchar(255)	YES

2. **BBC News** : The dataset consists of 2225 documents from the BBC news website corresponding to stories in five topical areas from 2004-2005. The news is classified into five classes as Business, Entertainment, Politics, Sports and Tech. The attributes in the dataset are as follows:

FIELD	TYPE
headline	varchar (255)
description	varchar (255)
category	varchar (32)

MODULE I: Pre-processing

In this step, news articles in the form of text files are fed into the system to undergo changes that make it usable for the other components in the system. The various steps performed as a part of pre-processing are as follows:

1. Tokenization: This step involves division of text into sets of discrete parts, called tokens. The words obtained from the tokenizer for the basis for the feature space of the training data.
2. Noise removal: This step involves removal of stop words, punctuation and converting the entire text to lowercase.
3. POS Tagging and Lemmatization: In this step, each word is reduced to its root form by lemmatizing it based on its Part-of-Speech (POS) tag.
4. TF-IDF vectorization:
   4.1. TF-IDF with Channel Distribution Information (CDI) based feature extraction : In this step, we convert the text documents into CDI TF-IDF vectors. The AG News dataset contains news articles from various sources (channels). The CDI TF-IDF scheme aims to exploit the variation in distribution of terms across different channels for better feature selection. CDI IDF is a refined IDF measurement for term based on the characters of channel-IDF values.

   For term T, the IDF value in multiple channels is a set of its IDF values across every channel.

$$IDF(T) = \{IDF_{C_i}(T) |\ C_i \in C\} = \left\{\log_2 \frac{|D_i|}{|1 + \{d_i : T \in d_i\}|}\right\}$$

where ***C*** is the whole collection and C_i is the collection of channel ***I***; D_i is the number of articles of C_i; d_i is the number of pages where term ***T*** is present.
Every term is likely to be frequently mentioned in one or more channels. But, if it is present frequently in all of the channels, then it is highly likely to be a noise word.
The importance of each term is described by a normalized standard deviation.

$$SD\_Rate(T) = \frac{\sigma(T)}{\mu(T)}$$

$$\sigma(T) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(IDF_{C_i}(T) - \mu)^2}$$

$$\mu(T) = \frac{1}{n}\sum_{i=1}^{n} IDF_{C_i}(T), C_i \in C$$

where $\boldsymbol{\mu(T)}$ is the average IDF value of the set, $\boldsymbol{\sigma(T)}$ is the standard deviation of channel-IDF values of term ***T***.
In order to locate the differences of IDF values between the particular channel and the others, the normalized standard deviation without the particular channel is calculated.

$$SD_{Rate_i}(\boldsymbol{T}) = \frac{\boldsymbol{\sigma_i(T)}}{\boldsymbol{\mu_i(T)}}$$

where $\sigma_i(T)$ and $\mu_i(T)$ are the standard deviation and the average IDF values without IDF value in channel i.
So, the adjusted channel-IDF of each term is given by

$$IDF'_i(T) = IDF_i(T) * \left[\frac{SD\_Rate(T)}{SD\_Rate_i(T)}\right]^2$$

This adjusted channel-IDF can reflect the importance of a term in a channel distribution. For a top word with a high standard deviation, all the channel-IDF values will be increased. Conversely, the meaningless terms which have an almost even channel distribution will be depressed.
Finally, the corrected IDF value of each term is given by

$$IDF_{corrected}(T) = \frac{1}{n}\sum_{i=1}^{n} IDF'_i(T)$$

In this way, each document is represented as a vector by calculating the $TF*IDF_{corrected}$ values for each term.

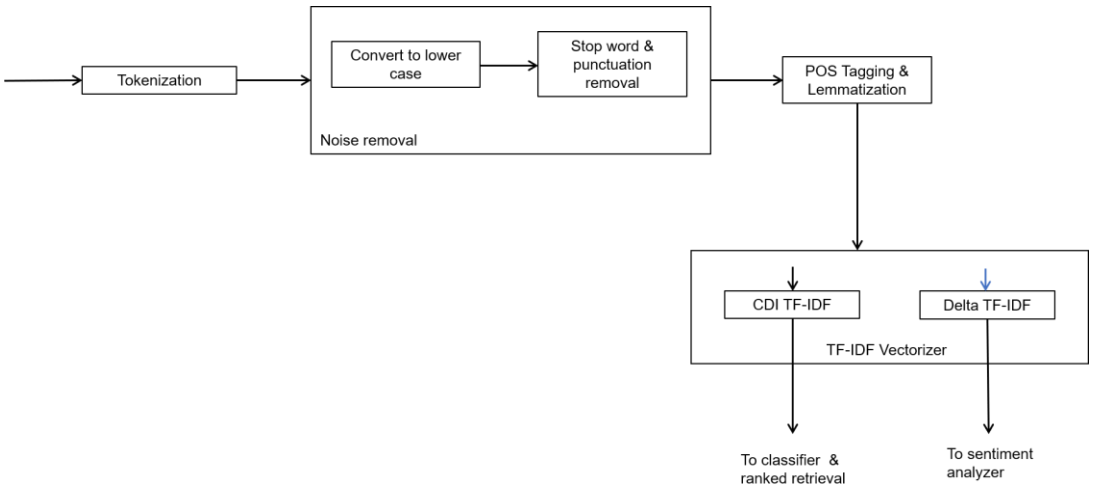
4.2. Delta TF-IDF based feature extraction : In sentiment analysis, it is important to know the distribution of words in the positive and negative corpora. Delta TF-IDF is a document representation technique that can be used to determine seed words for lexicon generation.
For each term T in document d, the delta TF-IDF is given by

$$\delta_d(T) = TF_d(T) * \log_2\left(\frac{|P|}{P_T}\right) - TF_d(T) * \log_2\left(\frac{|N|}{N_T}\right)$$

$$= TF_d(T) * \log_2\left(\frac{|P| * N_T}{P_T * |N|}\right)$$

$$= TF_d(T) * \log_2\left(\frac{N_T}{P_T}\right)$$

where $TF_d(T)$ is the frequency of the word T in document d, P_T is the number of documents in the positive corpus containing term T, P is the total number of documents in the positive corpus, N_T is the number of documents in the negative corpus containing term T, N is the total number of documents in the negative corpus.
This delta TF-IDF boosts the importance of words that are unevenly distributed between the positive and negative classes and depresses the importance of evenly distributed words. The value of an evenly distributed word is zero. The more uneven the distribution, the more important the word should be. Words that are more prominent in the negative corpus than the positive corpus will have a positive score, and the words more prominent in the positive corpus than the negative corpus will have a negative score.



Block Diagram of Pre-Processing

MODULE II: CLASSIFCATION

OBJECTIVE: Transductive Transfer Learning in Text Classification using Deep Convolutional Neural Networks

The common machine learning methods assume that the training and test data are taken from same feature space and distribution and when this distribution is changed, these methods have to be rebuilt from scratch. The concept of "Transfer Learning" tackles this problem by allowing the distribution of training and test data to be different.
Many examples can be found where this technique is needed to reduce the need and effort to recollect the training data. One example can be of news articles classification in which there is need to classify unlabeled articles into categories or classes such as sports, business etc.

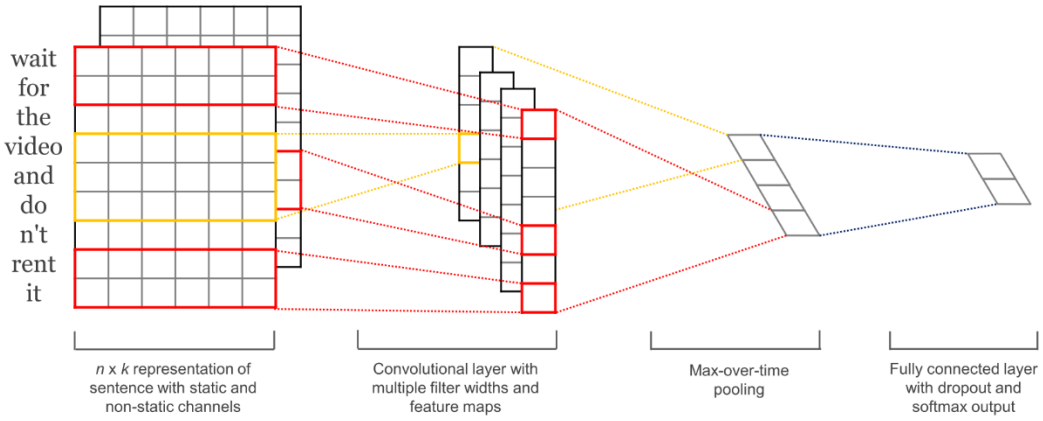
In this module, we have tried transfer learning using deep convolutional neural network by taking two different news datasets (i.e. AG's corpus and BBC corpus). The training is done on AG's corpus and then applying the trained model on BBC's dataset.
The evaluation is done by using accuracy as a measure against the labelled BBC data as the ground truth.

WHAT IS A CONVOLUTIONAL NEURAL NETWORK (CNN)?

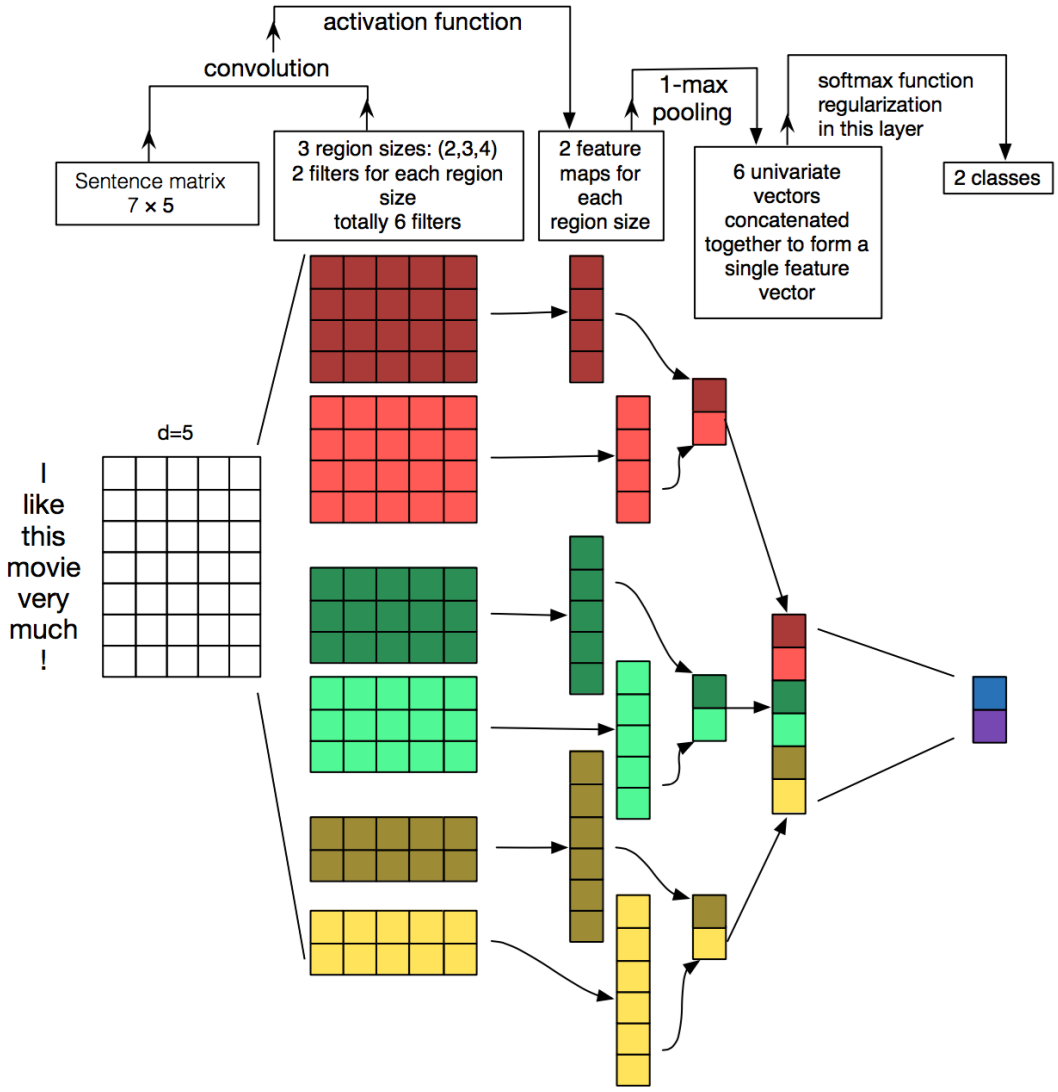
CNN is a feed-forward neural network with convolutional layers interleaved with pooling layers, where the top layer performs classification using the features generated by the layers below [ZhangConv].
The convolution layers have non-linear activation functions such as ReLU or tanh applied to the results. We use convolutions over the input to compute the output which results in local

connections, where each region of input is connected to a neuron in the output. Each layer applies different filters and aggregates their results. A CNN automatically learns the values of its filters. There are two aspects of this computation worth paying attention to: Location Invariance and Compositionality.

CNN in NLP (more specifically text classification)



The input to most NLP tasks are sentences or documents represented as a matrix. Each row of the matrix corresponds to one token, typically a word, but it could be a character. That is, each row is vector that represents a word.
We typically use filters that slide over full rows of the matrix (words). Thus, the “width” of our filters is usually the same as the width of the input matrix. The height, or region size, may vary, but sliding windows over 2-5 words at a time. Putting all the above together, a Convolutional Neural Network for NLP looks like the image shown below.

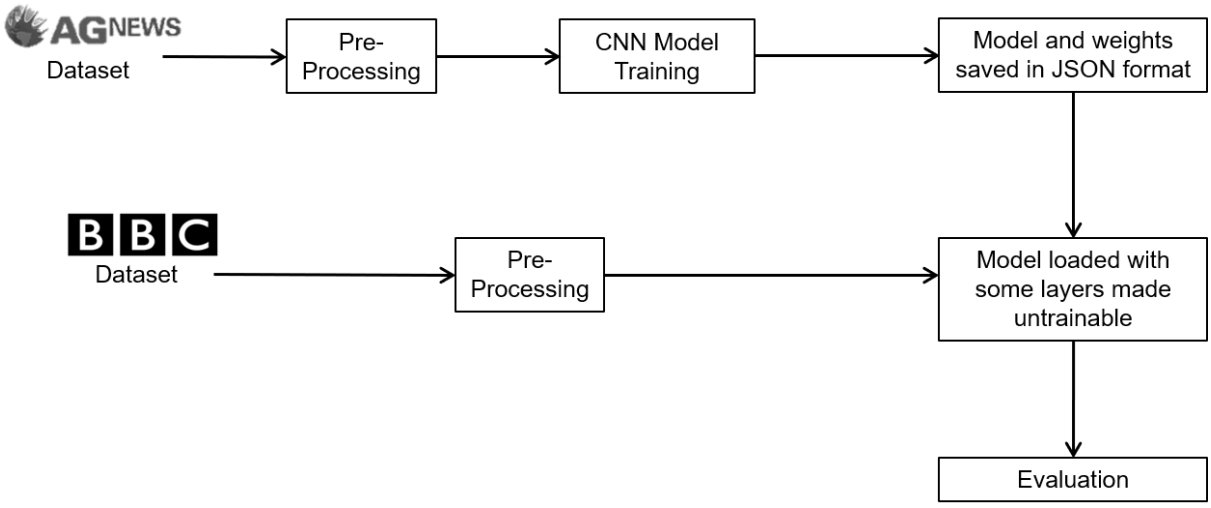


Here we depict three filter region sizes: 2, 3 and 4, each of which has 2 filters. Every filter performs convolution on the sentence matrix and generates (variable-length) feature maps. Then 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded. Thus a univariate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer. The final SoftMax layer then receives this feature vector as input and uses it to classify the sentence; here we assume binary classification and hence depict two possible output states. Also, we don't care where a word appears in a sentence. Words close to each other need not be semantically related.

Why CNNs?

CNNs are fast and we have applied the bag of words model, though might be incorrect but leads to good results.

System Description:



Block Diagram of Classification

Phase I: Model training on AG's corpus

Step 1: Pre-processing

- All the irrelevant fields were removed such as source, urls etc.
- Only title and description are taken as only text is needed for classification
- Both the fields are combined and rest is not required as

Step 2: CNN Model & its Hyperparameters

- Layers: Model has 1 input layer, 11 hidden layers (consisting of 6 convolution layers, 2 pooling layers and 2 dense layers) and 1 output layer
- Each convolution layer has 'ReLU' as its activation function strides = 1, padding =0
- Each pooling layer has pool_size = 3 and strides = 3, padding = 0

Hyperparameter	Value
nb_filter (Filters for convolution layers)	256
dense_outputs (Number of units in the dense layer)	1024
filter_kernels (Convolution layer kernel size)	[7, 7, 3, 3, 3, 3]
cat_output (Number of units in the final output layer i.e. Number of classes)	4
Maxlen (Maximum length. Longer gets chopped. Shorter gets padded.)	1600
batch_size	80
nb_epoch	10

- Model and weight HD5/H5 saved in json format to be used for transfer learning.

Phase II: Applying pre trained model on BBC dataset

Step 3: Preprocessing

- Same as applied on AG's corpus

Step 4: Model loading and CNN parameters

- Last output layer is dropped and a new output layer is added.
- First 5 layers are freezed to be made untrainable and rest are trainable according to the input.

Hyperparameter	Value
Maxlen (Maximum length. Longer gets chopped. Shorter gets padded.)	1600
batch_size	80
nb_epoch	10

**These parameters are specified for highest accuracy

Step 5: Evaluation

Epochs	Accuracy
8	88
10	90
20	89

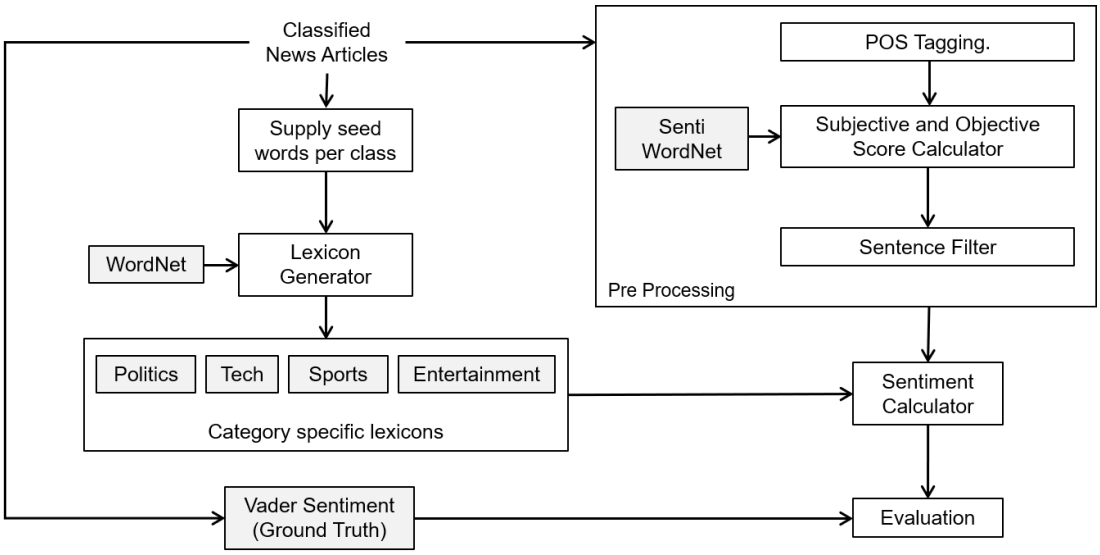
****

When normal classification was applied on AG's corpus:

Classifier	Accuracy	Accuracy
Naive Bayes	83.52 (with TF-IDF)	87.27 (with CDI TF-IDF)
SVM	88.36(with TF-IDF)	89.13 (with CDI TF-IDF)
CNN	93 (without any preprocessing)	95 (with some preprocessing)

MODULE III: Sentiment Analysis

In this module, we receive classified news articles from the previous phase and calculate the sentiment conveyed as positive, negative or neutral based on the opinion expressed by them. The following block diagram summarizes our approach.



Block Diagram of Sentiment Analysis

Seed Word Generation

To select the seed words, the delta TF-IDF vectorizer is used to represent category wise documents in the vector space for both the positive and negative corpus. For each corpus, the average values of delta TF-IDF are taken and the words with the top 3 values are selected as the seed words.

Lexicon Generator

The lexicon generator is used to generate a pool of words pertaining to a certain class from the generated list of seed words, along with their measure of positivity/negativity (impact). The classified news articles obtained from the previous phase have been tagged into five different categories: Business, Tech, Entertainment, Sport and Politics. The lexicon generator generates five separate lexicons for each of these categories.

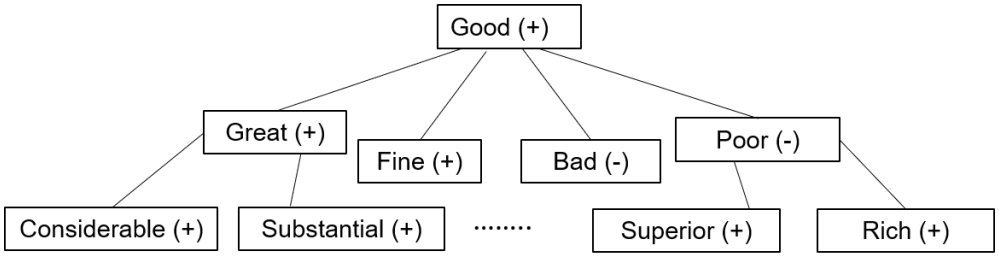
A small set of known words with known polarities is collected manually for all news categories. This set is then grown by repeatedly searching for synonyms in the computer dictionary WordNet. We start from the root word and generate a list of synonyms (synset) and antonyms for the word. We assign a polarity and score to the generated words using the following mechanism:

- Synonyms inherit the polarity(positive/negative) from their parents.
- Antonyms are assigned the polarity opposite to their parents.
- Seed Words (root) have a polarity of 1.
- If the polarity of a word *wrd* is positive, score of the word is given as:

$$S_{wrd} = wup\_similarity\ (wrd, root)$$

- If the polarity of a word *wrd* is negative, score of the word is given as:

$$S_{wrd} = wup\_similarity\ (wrd, root) - 1$$



S_{Great} = *wup_similarity*('Great','Good')
S_{Bad} = *wup_similarity*('Great','Good')-1
S_{Poor} = *wup_similarity*('Bad','Good')-1
$S_{Superior}$= *wup_similarity*('Superior','Good')

Lexicon Generation

The generated list is then appended to the lexicon, and in the next iteration, the same process is carried out on the newly added words, thereby propagating the process. The process stops when no new words are found. The significance of a word in the lexicon decreases as we move further away from the root word. Here, we take the Wu-Palmer metric as a measure to calculate the similarity of a word with respect to the seed. This way the score reduces, as we move down the lexicon.

Pre-Processing

SentiWordNet is a lexical resource for opinion mining and it assigns positive and negative scores to each synsets of WordNet.

The tagged articles are first tokenized and part of speech tagging is done on the tokenized words. We then use SentiWordNet on the POS tagged words, to calculate the subjective and objective score of each word.

The subjective and objective score of an **adjective** is calculated using:

$$Sub(w_i) = \alpha * \{|pos(w_i)| + |neg(w_i)|\}$$
$$Obj(w_i) = \alpha * \{ 1 - Sub(w_i) \}$$

The subjective and objective score of **verbs and adverbs** is calculated using:

$$Sub(w_i) = \beta * \{|pos(w_i)| + |neg(w_i)|\}$$
$$Obj(w_i) = \beta * \{ 1 - Sub(w_i) \}$$

- Here, $pos(w_i)$ and $neg(w_i)$ denote the positive and negative score of a word as obtained from the SentiWordNet.
- Since adjectives play a vital role in expressing opinion, it has more weight than other part of speech words. Hence the value of α is assumed to be greater than the value of β. For our case, we take $\alpha = 3$ and $\beta = 1$
- Other than adjectives, verbs and adverbs no other POS words are influential in determining the sentiment of a news article. Hence, they are not taken into account while calculating the score.

Once the scores for the individual words in a sentence have been calculated, we now calculate the subjective and objective score for the entire sentence. This is done because, we need to strip the news article of the irrelevant sentences, and take the most impactful sentences for sentiment calculation. The subjective score of a sentence shows how well it conveys opinions, feelings or emotions. The more the subjective score of the sentence, the more relevant it is, and hence can be considered for sentiment calculation. Objective score, on the other hand

determines the lack of biasness or judgement. The higher the objective score, the more neutral the sentence is.
The scores for a sentence is calculated using:

$$Sub(s_i) = \frac{\sum_{i=1}^{n}(Sub(w_i))}{n}$$

$$Obj(s_i) = \frac{\sum_{i=1}^{n}(Obj(w_i))}{n}$$

Here n is the number of words per sentence. Once this is done, we discard those sentences where $Obj(s_i) > Sub(s_i)$. This way we only consider sentences with high impact for sentiment analysis. Next, these sentences undergo tokenization, stop word removal and lemmatization before proceeding to sentiment calculation.

Sentiment Calculation

Here, we use the lexicon generated in the first step to calculate the sentiment of the news article. The input to this phase is the trimmed news articles obtained after pre-processing. The sentences in these articles are now purely subjective and convey better sentiment than the entire article.
The lexicon apart from storing the polarities, also stores the polarity score (S_{wrd}) signifying how positive or negative a word is. For each word we lookup the polarity score in the lexicon and the sentiment score for the entire article (a_i) is given by:

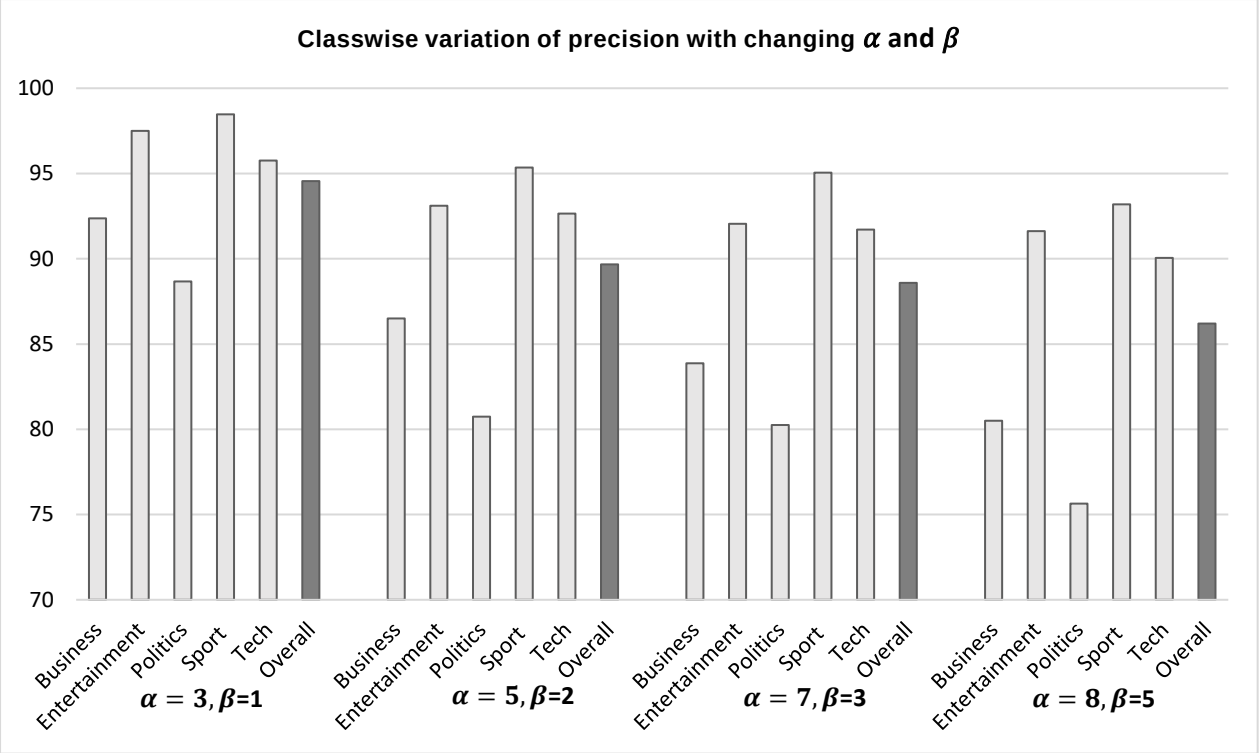
$$Sentiment(a_i) = \frac{\sum_{i=1}^{n}(S_{wrd})}{n}$$

We put a threshold on the sentiment score obtained from the above calculation to determine whether the article is positive, negative or neutral.

Sentiment Score	Sentiment
>0.2	Positive
-0.2 to 0.2	Neutral
<-0.2	Negative

Evaluation

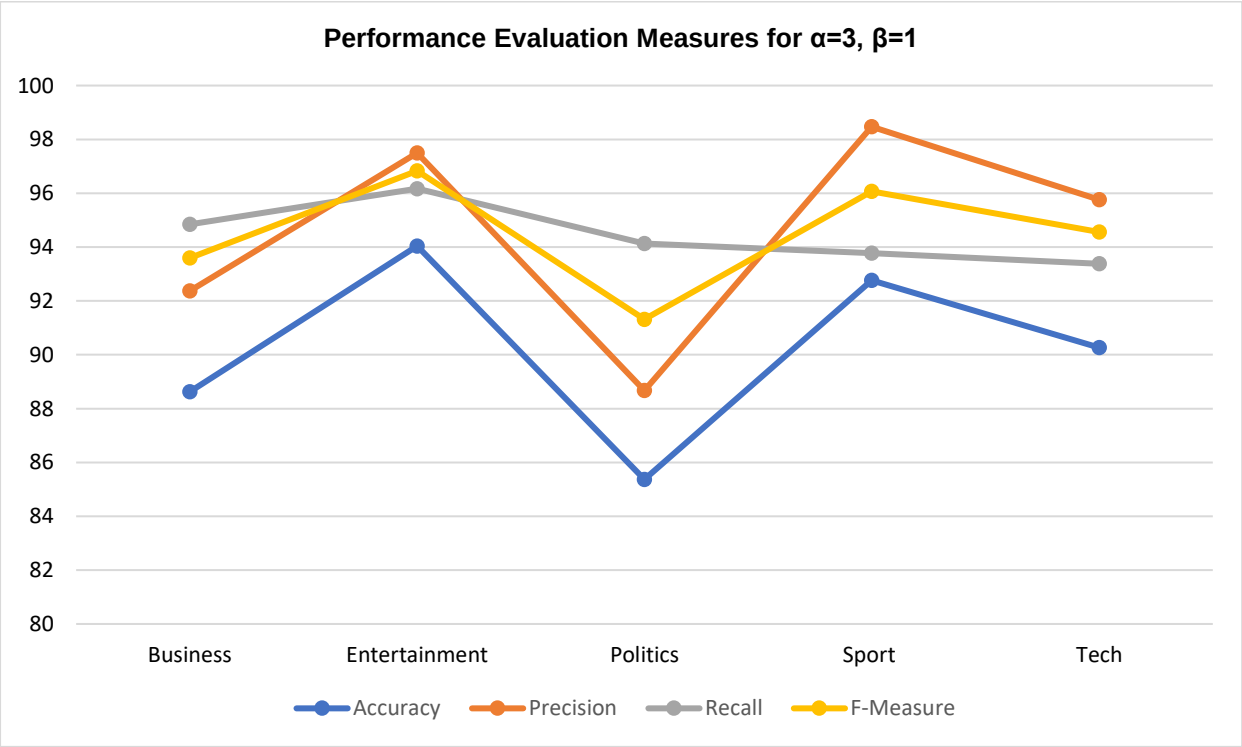
For evaluation, we subject the classified news articles to the VADER (Valence Aware Dictionary for Sentiment Reasoning) Sentiment Analyzer provided by the Natural Language Processing Toolkit (NLTK). It provides a rule based model for general sentiment analysis. Statistically it has been proven that VADER outperforms individual human raters (F1 Classification Accuracy = 0.96), and generalizes more favorably across contexts than any other benchmarks.
We assume the output provided by VADER as our ground truth, and compare our generated sentiment to it. The following were the results obtained.



Changing trends with changing α, β

As mentioned before, adjectives are given more weightage than verbs or adverbs, since they play the most important role in expressing opinion. However, it is seen, that increasing $\boldsymbol{\alpha}$ beyond 3 tends to have an adverse effect on the precision of the system. This is because, on assigning a very high value to an adjective, the polarity of that word overpowers other verbs or adverbs that might convey a different polarity.

After trying out multiple values of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ we finally select $\boldsymbol{\alpha = 3}$, $\boldsymbol{\beta}$**=1** as the optimum values.

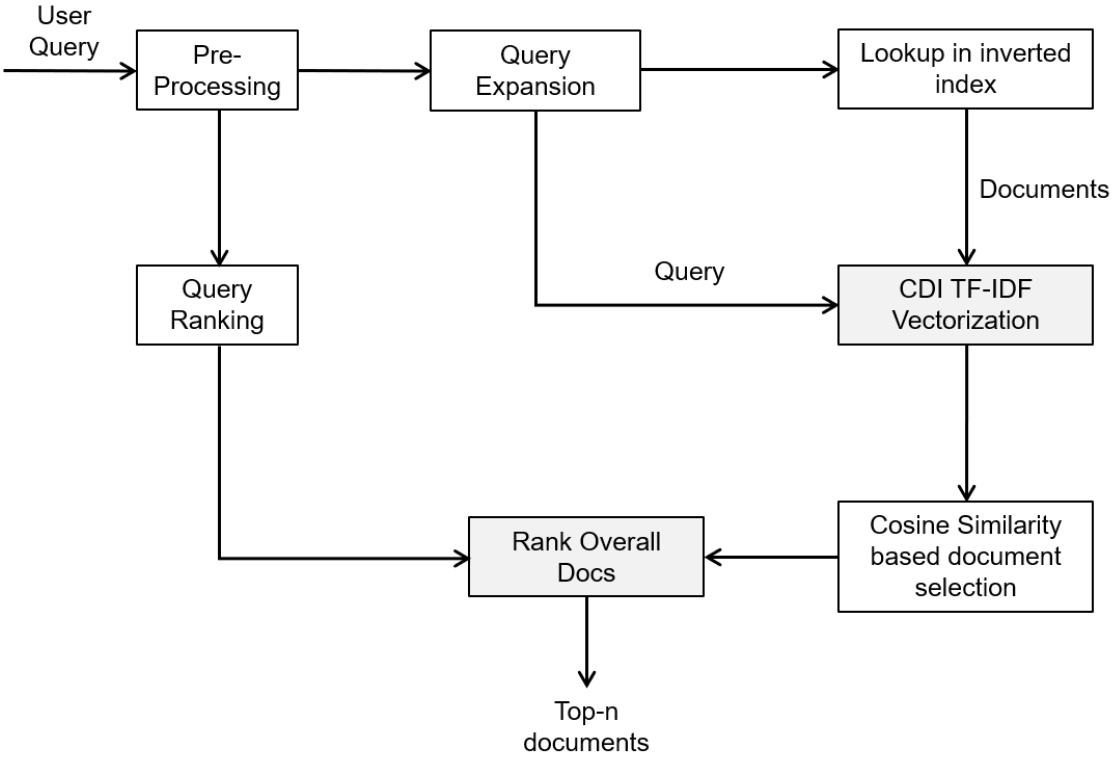


Why precision is our evaluation metric of choice?

The dataset that we are working on has a higher number of positive sentiment news. So, even if all the articles are classified positive, it will still result in a high value of recall. The graph shown below, shows a more or less constant value for all the news categories due to this reason. Hence, we don't consider recall for evaluation. Since f-measure is also dependent on recall, we do not consider it either. Precision on the other hand denotes the fraction of relevant instances among the retrieved instances**.** Hence, precision is our metric of choice.

MODULE IV: Ranked Retrieval

In this module, user queries are accepted and relevant documents are ranked and returned to the user. The different steps involved are as follows:



Block Diagram of Ranked Retrieval

1. Pre-processing of query: In this step, standard pre-processing (tokenization, stop word and punctuation removal, lemmatization) is performed on the user query.
2. Query expansion: In this step, the original query is expanded by generating semantically equivalent alternatives. This is done by generating synonyms for every token in the query and then combining them to obtain all possible alternate queries.
   Eq: If a user query is *Firms that flout network*, then 735 alternate queries are generated.
3. Query Ranking: In this step, every alternate query is assigned a score with respect to the original query which is a measure of similarity between the two. This score is calculated by finding the semantic similarity between word pairs from the alternate and original queries and adding them up.
   Eg: For the query (Q_{org}) *Firms that flout network,* an alternate query (Q) is *firm flout web*. The score for this query is given by
   $$QScore(Q) = Sim(firm, firm) + Sim(flout, flout) + Sim(network, web) = 2.22$$
4. Inverted index generation: In this step, each document from the dataset is parsed and tokenized. Further standard preprocessing tasks like stop word removal and lemmatization are performed. For every token, a posting list of document number (doc-ID) is created. In the end of procedure, a hashmap data-structure is created for all tokens where key is token and value is posting list.
5. Lookup in inverted index: In this step, every query word is searched in the inverted index and a list of all the documents relevant to a query is obtained by performing a union operation on the list of documents for every query term.
6. CDI TF-IDF Vectorization: In this step, the relevant documents for each query are further used as a training set for CDI TF-IDF scheme. According to mentioned algorithm in

preprocessing module, CDI based TF-IDF vector is calculated for each document. On the similar line, each query is also transformed in to the CDI TF-IDF vector.

7. Cosine Similarity based document selection: In this step, cosine similarity measure is calculated on TF-IDF vector of each query and all set of matched documents.
   For each query, similarity score values are sorted in descending order and top *K* documents are selected. For m queries (including original), at most m*k documents are selected as some of the documents could be overlapping match.
8. Rank overall documents: In this step, all the selected documents are ranked based upon a cumulative score which takes respective query set into consideration.
   Let's say if a particular document is found matched in t queries, then score is calculated as

$$Score(D_i) = \sum_{j=1}^{t} cosineSim\left(D_i, Q_j\right) * QScore(Q_j)$$

Evaluation:

With TF-IDF	Recall (R)	MAP
N = 5	79.04	70.58
N = 10	81.62	73.39
N = 15	80.23	78.06

With CDI TF-IDF	Recall (R)	MAP
N = 5	86.91	75.82
N = 10	83.11	76.74
N = 15	85.37	76.51