Information Retrieval and Extraction

# SEMANTIC ANNOTATION OF DOCUMENTS

-Naman Singhal, Rashi Shrishrimal, K S Chandra Reddy

## Abstract

The 2012 ACM Computing Classification System has been developed as a poly-hierarchical ontology that can be utilized in semantic web applications. A research paper in acm digital library should ideally have the categories associated it. Our goal is to automate the process so as to predict the most appropriate category, the research paper lies in.

### Key Words

Classification, Latent Dirichlet Allocation, ACM, Semantic Annotation, Supervise LDA, Word2Vec.

## Introduction

The semantic annotation of documents is an additional advantage for retrieval, as long as the annotations and their maintenance process scale well.

With the rapid growth of the information amount available online and in databases, search engines play an important role within eLearning, since they can support the learner in looking for the needed information for his learning, training or teaching process. However, these information extraction systems are based on terms indexation without taking into account neither the semantics of learning information’s contents nor the context. In general, users usually enter keywords into search engines, that are based on terms indexation without taking into account neither the semantics of the pedagogical content nor the context.

Here, we discuss three approaches to go about the problem, namely basic cosine similarity, latent dirichlet allocation and labeled lda. As the work progresses we see how we get better results and the best with the labeled lda.

## Problem Statement

Given an abstract of a research paper, we need to find the category that it fits into.The 2012 ACM Computing Classification System has been developed as a poly-hierarchical ontology that can be utilized in semantic web applications.The complete classification tree can be found here.

## Datasets

1. A private dataset of the SIEL lab of IIIT-Hyderabad.

Description : For every research paper there are the following fields:

1) Title
2) Author
3) Country of Authors
4) Year of Publication
5) Conference
6) Categories
7) Abstract (Only for some papers)

2. Wiki dataset

Description : For all the acm categories a dataset was build having the summary of the wikipedia page (The First section of the wikipedia document).

3. dblp dataset

Description : For initial tasks, this dataset was used to train the lda model. Though this is not useful for the final proposition of the model.

## Our Solution

We used multiple approaches based on their relevance to our problem statement and their accuracy levels in correctly classifying test data sample.

### Approach 1: Cosine Similarity

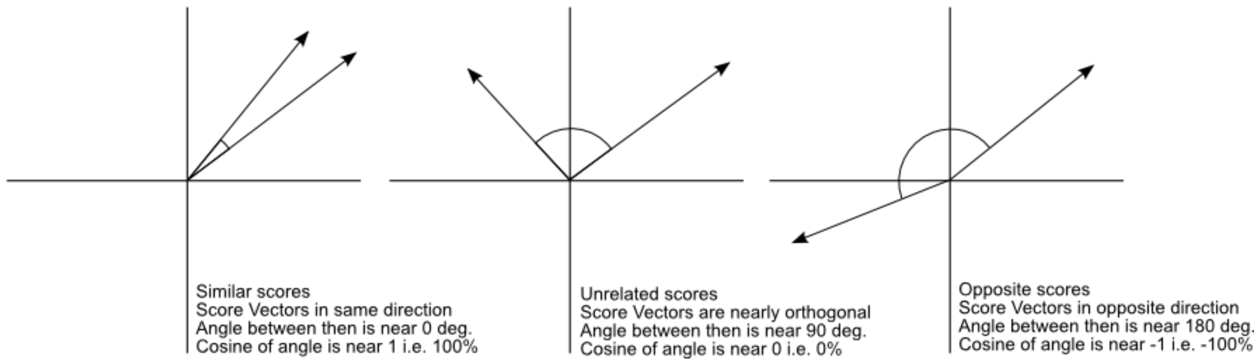
In our approach we took the wikipedia summary of each of the categories as separate documents and measured the cosine similarity of the test document with each of them and assigned the category whose wikipedia document was most similar.

**Brief overview of cosine similarity :**
The cosine similarity between two documents on the Vector Space is a measure that calculates the cosine of the angle between them. This metric is a measurement of orientation and not magnitude, it can be seen as a comparison between documents on a normalized space because we're not taking into the consideration only the magnitude of each word count (tf-idf) of each document, but the angle between the documents. What we have to do to build the cosine similarity equation is to solve the equation of the dot product for the $\cos\theta$:

$$\vec{a} \cdot \vec{b} = \|\vec{a}\|\|\vec{b}\| \cos\theta$$

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\|\|\vec{b}\|}$$

And that is it, this is the cosine similarity formula. Cosine Similarity will generate a metric that says how related are two documents by looking at the angle instead of magnitude, like in the examples below:


Similar scores
Score Vectors in same direction
Angle between then is near 0 deg.
Cosine of angle is near 1 i.e. 100%
Unrelated scores
Score Vectors are nearly orthogonal
Angle between then is near 90 deg.
Cosine of angle is near 0 i.e. 0%
Opposite scores
Score Vectors in opposite direction
Angle between then is near 180 deg.
Cosine of angle is near -1 i.e. -100%

The Cosine Similarity values for different documents, 1 (same direction), 0 (90 deg.), -1 (opposite directions).

Note that even if we had a vector pointing to a point far from another vector, they still could have an small angle and that is the central point on the use of Cosine Similarity, the measurement tends to ignore the higher term count on documents. Suppose we have a document with the word “sky” appearing 200 times and another document with the word “sky” appearing 50, the Euclidean distance between them will be higher but the angle will still be small because they are pointing to the same direction, which is what matters when we are comparing documents.

**Procedure:**

1) For all the ACM categories, the corresponding first section of wiki pages were retrieved.
2) For each test document  (paper abstract)
    a) Cosine similarity found for all wiki data
    b) The one corresponding to the wiki data with highest score is the category assignment to the test document.

**Reason for choosing this :**
Since the categories will be similar to the abstract of the research paper, we implemented this. By abstracting out the magnitude of the term vector because it takes out the influence of the acm category summary (wiki data) length. Only the relative frequencies between words in the test sample, and training samples, are of importance, and not how big the summaries of each category are. This enables that (e.g. in computation of similarity) category with huge summary  do not always come on top.

**Drawbacks:**
It fails to find semantic similarity of the test sample among category abstracts.

**Results:**

| Test Data # | Category Accuracy | Tag Accuracy |
|---|---|---|
| 100 | 5% | 30% |

**Challenges:**

- If we use a similarity based model like “cosine similarity”, the document is matches to related but wrong category. This is because cosine similarity doesn’t take semantics into consideration. For ex, let’s consider the abstract

“In this talk, I will argue that complex sociotechnical systems like YouTube admit multiple conceptualizations - what YouTube is and is about - and that each of these perspectives ultimately results in different ways of posing research questions in multimedia. I will then contrast the existing lines of YouTube multimedia research under this view. In particular, I will discuss a perspective that put the focus on people - the You in YouTube – and present an overview of ongoing work that aims at developing computational models to characterize YouTube as a collection of communities where individuals express and communicate through video. I will finally discuss opportunities for future research in multimedia under this framework.”

- Because  of the presence of “youtube” many times, the document is classified to category “YouTube” instead of “Human Information Processing” to which the document actually belongs to.

**Observation:**
Cosine Similarity doesn’t take into account any semantic relationship among documents but just word matches in a way, but unlike tf-idf, ignoring the size of the training data. So, for two documents with word-intersection-set of size zero, the cosine similarity is zero, whereas they might be related contextually or domain-wise with non-zero similarity match. This makes cosine similarity an ideal model to “tag” documents rather than “categorize” documents.

## Approach 2: Latent Dirichlet Allocation

In our approach, we used LDA (topic modelling algorithm) to cluster documents and assign a category to each cluster depending on the cluster in which the wikipedia document of the category is matching the most. Then put the test document on same model and assigned the category depending on the cluster to which test document belonged and the category assigned to that cluster.

**Brief overview of LDA:**
Latent Dirichlet allocation (LDA) is a topic model that generates topics based on word frequency from a set of documents. LDA is particularly useful for finding reasonably accurate mixtures of topics within a given document set.

In more detail, LDA represents documents as mixtures of topics that spit out words with certain probabilities. It assumes that documents are produced in the following fashion: when writing each document, we

- Decide on the number of words N the document will have (say, according to a Poisson distribution).
- Choose a topic mixture for the document (according to a Dirichlet distribution over a fixed set of K topics). For example, assuming that we have the two food and cute animal topics above, you might choose the document to consist of 1/3 food and 2/3 cute animals.
- Generate each word in the document by:
    - First picking a topic (according to the multinomial distribution that you sampled above; for example, you might pick the food topic with 1/3 probability and the cute animals topic with 2/3 probability).
    - Then using the topic to generate the word itself (according to the topic's multinomial distribution). For instance, the food topic might output the word "broccoli" with 30% probability, "bananas" with 15% probability, and so on.

Assuming this generative model for a collection of documents, LDA then tries to backtrack from the documents to find a set of topics that are likely to have generated the collection.

**Procedure:**
Both the test data (paper abstract) and the training data go through "Data Cleaning" before being given to lda for processing. Data Cleaning includes

- Tokenizing: converting a document to its atomic elements.
- Stopping: removing meaningless words.
- Stemming: merging words that are equivalent in meaning.

Tokenization segments a document into its atomic elements. In this case, we are interested in tokenizing to words. Tokenization can be performed many ways–we are using NLTK's tokenize.regexp module:

```
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer(r'\w+')
```

The above code will match any word characters until it reaches a non-word character, like a space. This is a simple solution, but can cause problems for words like "don't" which will be read as two tokens, "don" and "t." NLTK provides a number of pre-constructed tokenizers like nltk.tokenize.simple. For unique use cases, it's better to use regex and iterate until your document is accurately tokenized.

Certain parts of English speech, like conjunctions ("for", "or") or the word "the" are meaningless to a topic model. These terms are called stop words and need to be removed from our token list.Removing stop words is now a matter of looping through our tokens and comparing each word to the en_stop list.

In our case, we are using the stop_words package from Pypi, a relatively conservative list.

```
from stop_words import get_stop_words
en_stop = get_stop_words('en')
```

Stemming words is another common NLP technique to reduce topically similar words to their root. For example, "stemming," "stemmer," "stemmed," all have similar meanings; stemming reduces those terms to "stem." This is important for topic modeling, which would otherwise view those terms as separate entities and reduce their importance in the model.

Like stopping, stemming is flexible and some methods are more aggressive. The Porter stemming algorithm is the most widely used method. To implement a Porter stemming algorithm, import the Porter Stemmer module from NLTK:

```
from nltk.stem.porter import PorterStemmer
p_stemmer = PorterStemmer()
```

The result of our cleaning stage is texts, a tokenized, stopped and stemmed list of words from a category summary. Once we looped through all our category summaries and appended each one to texts, texts is a list of lists, one list for each of our category summaries

To generate an LDA model, we need to understand how frequently each term occurs within each document. To do that, we need to construct a document-term matrix with a package called gensim:

```
from gensim import corpora, models
dictionary = corpora.Dictionary(texts)
```

The Dictionary() function traverses texts, assigning a unique integer id to each unique token while also collecting word counts and relevant statistics. To see each token's unique integer id, try print(dictionary.token2id).

Next, our dictionary must be converted into a bag-of-words:

```
corpus = [dictionary.doc2bow(text) for text in texts]
```

The doc2bow() function converts dictionary into a bag-of-words. The result, corpus, is a list of vectors equal to the number of documents. In each document vector is a series of tuples. The tuples are (term ID, term frequency) pairs.

## Applying LDA Model

corpus is a document-term matrix and now we're ready to generate an LDA model

```
ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics=3, id2word = dictionary, passes=20)
```

Parameters:

- num_topics: *required*. An LDA model requires the user to determine how many topics should be generated. Our document set is small, so we're only asking for three topics.
- id2word: *required*. The LdaModel class requires our previous dictionary to map ids to strings.
- passes: *optional*. The number of laps the model will take through corpus. The greater the number of passes, the more accurate the model will be. A lot of passes can be slow on a very large corpus.

Our LDA model is now stored as *ldamodel*. We can review our topics with the print_topic and print_topics methods

```
print(ldamodel.print_topics(num_topics=3, num_words=3))
```

Each generated topic is separated by a comma. Within each topic are the three most probable words to appear in that topic.

Testing:

Now each of the category summaries are ran on lda and based on maximum match to a cluster, the cluster is assigned a "category name". Now the test data (paper abstract) is classified to one of the 'named' clusters and categorized appropriately.

```
testText_bow=dictionary.doc2bow(stemmped_test_tokens)

testText_lda=ldamodel[testText_bow]
```

**Reason for choosing this:**
LDA takes each test document as composed of various categories, and gives the probability of each category in composing the test document and there by categorizing the document to the document with maximum contribution. This fits in well with our problem statement where a paper abstract is to be categorized to its most appropriate category.

**Challenges:**

- Since the size of training data is 1636 each for a leaf category in ACM classification, and if a clustering technique like LDA is used, there'll be 1636 clusters. But each cluster will not have enough data to semantically match and classify a test document (abstract) to its correct category.

- Also given the training data ie wiki summary for each acm category and the test data ie abstract of a research document are distinct in terms of simplicity, diversity and technicality of word usage, training data is not good enough to semantically categorize test data accurately.

**Results:**

| Test Data # | Category Accuracy |
|---|---|
| 100 | 15% |

**Observation:**
LDA is a way fails to make best use of training data available ie given the training data is labeled, there is no option to input the category a document belongs to while training the document. Also, given the data set is not very appropriate to our test data, it is affecting the accuracy.

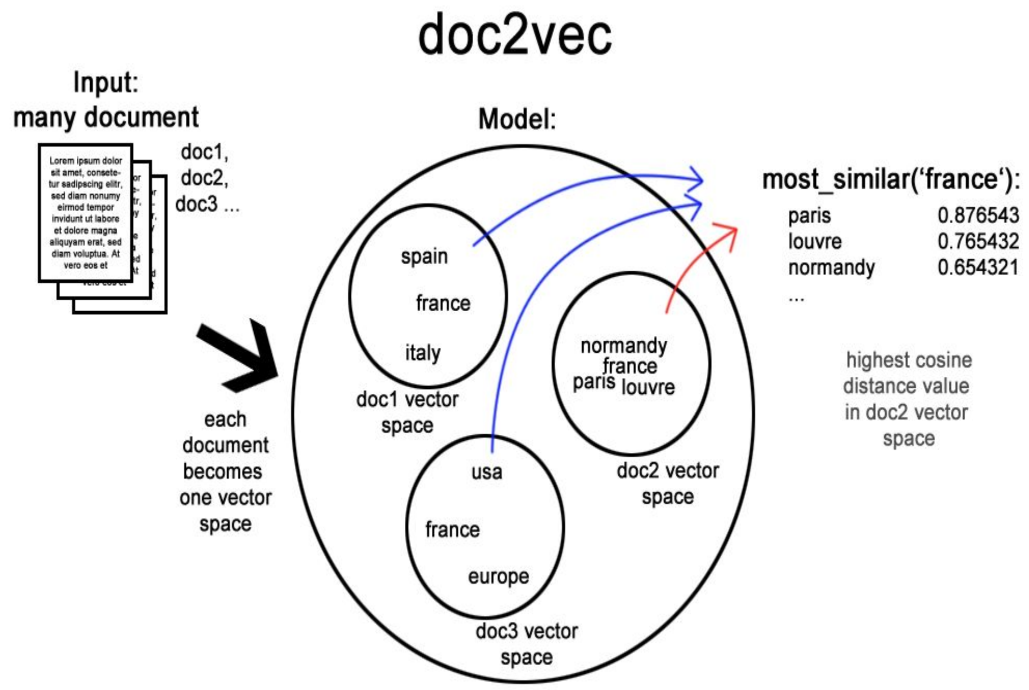
**Approach 3: Labeled Latent Dirichlet Allocation + Doc2Vec**

Accuracy can only be improved by supervised topic modelling i.e. by labelling the documents with topics and taking care of the semantic distance between the research paper and the categories.

**Brief overview of Labeled Latent Dirichlet Allocation :**
Labeled LDA is a probabilistic graphical model that describes a process for generating a labeled document collection. Like Latent Dirichlet Allocation, Labeled LDA models each document as a mixture of underlying topics and generates each word from one topic. Unlike LDA, L-LDA incorporates supervision by simply constraining the topic model to use only those topics that correspond to a document's (observed) label set.

**Brief overview of Doc2Vec :**
Doc2Vec is a tool provided by gensim that maps each sentences to a paragraph vector. Paragraph vector is  an unsupervised framework that learns continuous distributed vector representations for pieces of texts. The texts can be of variable-length, ranging from sentences to documents. The name Paragraph Vector is to emphasize the fact that the method can be applied to variable-length pieces of

texts, anything from a phrase or sentence to a large document.



**Procedure:**

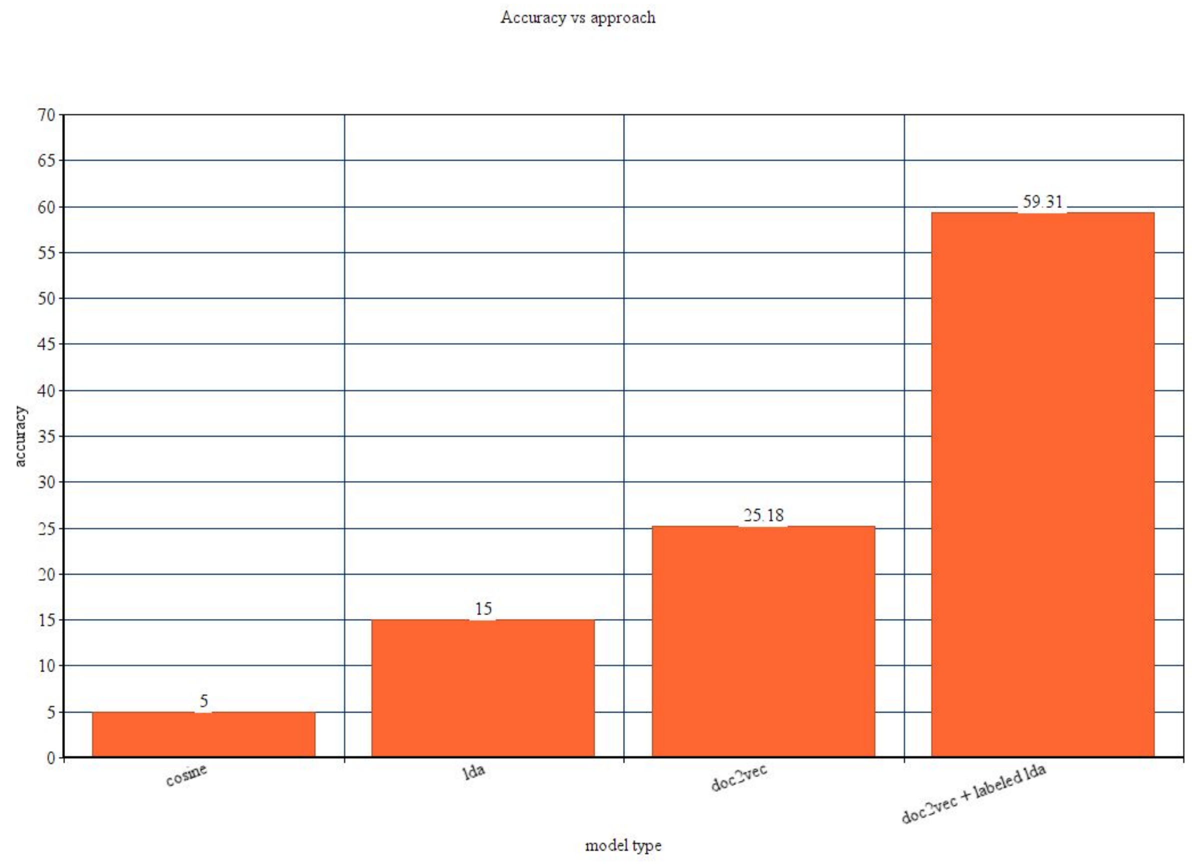
1. Generate data containing only the title and abstract of research paper from the training data. Run labeled LDA on this data by labeling the categories mentioned in the data to get the top 20 words for each category.
2. Generate data containing only the title and abstract of research paper from the testing data.
3. Merge the data from 1 and 2 and prepare a doc2vec model.
4. Assign category depending on the cosine similarity between the vector formed by the 20 words of the category and the vector of title and abstract.

**Results:**

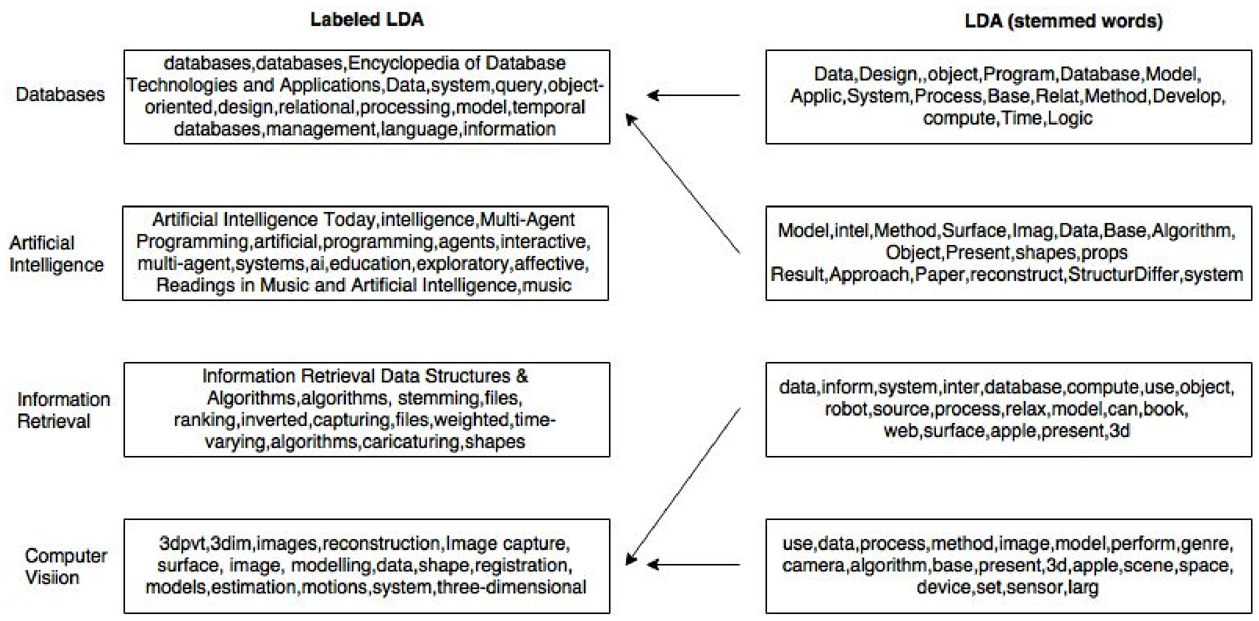
1) The simple cosine similarity approach could not encompass the semantic part of the documents and hence the categories. The Keyword matching though was acceptable. It is because no latent idea was incorporated in the model. 5% for the categories and 30% for the keywords.
2) Latent Dirichlet Allocation was able to incorporate the idea of semantic annotation as latent terms were found. But, then also the latent terms it

predicts are not guaranteed to match the actual categories due to its unsupervised nature. The cluster has many wiki documents and hence the acm categories. The downsizing in the number of clusters resulted in the problem. The accuracy was less than 15%.

3) Doc2Vec : **MAP value : 25.18%, NDCG = 35.26%**
4) Labeled lda : This is the most sensible algorithm for this type of problem. The results show the **MAP value of 59.31%** and **NDCG = 45.03%**



**Comparision:**


Labeled LDA
LDA (stemmed words)
Databases
databases,databases,Encyclopedia of Database Technologies and Applications,Data,system,query,object-oriented,design,relational,processing,model,temporal databases,management,language,information
Data,Design,,object,Program,Database,Model, Applic,System,Process,Base,Relat,Method,Develop, compute,Time,Logic
Artificial Intelligence
Artificial Intelligence Today,intelligence,Multi-Agent Programming,artificial,programming,agents,interactive, multi-agent,systems,ai,education,exploratory,affective, Readings in Music and Artificial Intelligence,music
Model,intel,Method,Surface,Imag,Data,Base,Algorithm, Object,Present,shapes,props Result,Approach,Paper,reconstruct,StructurDiffer,system
Information Retrieval
Information Retrieval Data Structures & Algorithms,algorithms, stemming,files, ranking,inverted,capturing,files,weighted,time-varying,algorithms,caricaturing,shapes
data,inform,system,inter,database,compute,use,object, robot,source,process,relax,model,can,book, web,surface,apple,present,3d
Computer Visiion
3dpvt,3dim,images,reconstruction,Image capture, surface, image, modelling,data,shape,registration, models,estimation,motions,system,three-dimensional
use,data,process,method,image,model,perform,genre, camera,algorithm,base,present,3d,apple,scene,space, device,set,sensor,larg

If we take top twenty words from each category it can be seen that labeled LDA has greatly increased the number of relevant words in the cluster, thereby affecting and increasing the accuracy. It can also be seen that cluster belonging to a particular category given by LDA has a lot of noise or say words which can be related to other categories as well, thereby it is less reliable to start with and so can’t be expected to accurately classify test data.

**Observation:**
We started off with Cosine Similarity, which doesn’t cash-in semantic similarity in classifying test data. Latent Dirichlet Allocation though takes semantics into account, it doesn’t make the best use of training data, where label of each training document can’t be used to train the model. Labeled LDA improved on LDA, by taking in category each document belongs to as well during training, thereby providing most relevant set of words per category, so does a better job in categorizing test data.

## References

**For Labelled LDA:**

- Python implementation of Labeled LDA (Ramage+ EMNLP 2009)

**For Doc2Vec:**

- Doc2Vec tutorial using Gensim
- models.doc2vec – Deep learning with paragraph2ve

**Test Data Papers (used for testing cosine similarity and lda):**

- Test Doc 1
- Test Doc 2
- Test Doc 3
- Test Doc 4
- Test Doc 5
- Test Doc 6
- Test Doc 7
- Test Doc 8
- Test Doc 9