CS 410

Final Project

Software Code Documentation

Manu Vinod Shesha, Angus Jyu, Sofia Godovykh

Introduction

This is the code software documentation guide for our team's final project for CS 410. Our team consists of the following people:

1. Manu Vinod Shesha (manuv3) (Captain)
2. Angus Jyu (angusfj2)
3. Sofia Godovykh (sofiag5)

The topic is **"Intelligent Topic Modeling and Index building of Course on Coursera/EducationalWeb"**. This loosely aligns with Theme 2: Intelligent Learning Platform, with the sub-area of "ConceptView". A typical online course on such platforms is structured as a series of lectures, each with some text resources like a transcript and a presentation slide. We aimed to apply Text Mining techniques like Topic Mining (and other NLP techniques) to discover latent topics (concepts) in each of these lectures and represent them in a way that can help students to quickly discover these concepts. We imagined the user interface as a sort of "Topic Index" which links the discovered topics to specific lectures. At the same time, with this project, we wanted to do some exploratory study of the effectiveness of Topic mining on a corpus that has a small size (like a semester-long online course).

Overview of Functions of the Code

What the code does:

- The main purpose of the code is to discover latent topics in a given lecture. It uses Topic Mining, specifically LDA (Latent Dirichlet Allocation) algorithm, as a base technique, to bring out important terms ("concepts") in a lecture.

- It then applies additional heuristics to improve topic differentiation (which means that the topics are interpretable and differentiable from one another). The terms here include single words as well as bigrams. Here we have tried different approaches like using different numbers of topics, boosting topic-specific terms over more overlapping terms using heuristics (described later), using Word2Vec to find similar terms etc. Not all these models were part of final output, but helped us explore various Text Mining techniques.

- The code builds different artifacts:

a. Discovered Topics
b. Document-Terms index: Map of document ids and important “terms” discovered
c. Topic-Documents index: Map of topic ids and documents which cover those topics prominently
d. Word-Documents index: Map of discovered terms and documents which contain them prominently.

- The code exposes these artifacts using an REST API layer, exposed using a Python Flask server.

- The code exposes two HTML pages:
    a. Model-Summary page: This page provides Topic modeling details like topics discovered, overall distribution over the whole corpus, and a term index. Here users can toggle between different models (deployed on server).
    b. Document-Topics page: This page is specific to each document in the corpus and provides details like topics distribution in the document and most important terms discovered by the model, for the given document.

What the code can be used for:

This code can be used as an app to provide the index-like representation of the terms/concepts discovered using Topic Mining, which can provide a mapping to relevant lectures. This has the potential to enhance the learning experience as it will be useful for the learner to quickly navigate to a specific lecture based on "topic". This idea goes beyond the basic search provided by EducationalWeb, where a user can search for a topic and this search task is treated as a simple "bag of words" search to give a whole list of results that may and may not be relevant for the user. Also, a traditional search on such learning platforms is not comparable to a full-fledged browser, and here, the assumption is that the user has some idea of what he is searching for. Instead, we want to provide a push-kind of a model, where we generate an index of relevant topics covered in a course (spread across a series of lectures).

Documentation of the Implementation of the Software

Documentation of Usage of Datasets & Algorithms:

We used the following datasets as input for our project:

- CS410 course transcripts and presentation slides that are available on Coursera (to be used as our Collection Corpus)
- “Brown” corpus and “Reuters” corpus from NLTK, as “background” models

The first challenge was to extract these files from Coursera. We created a simple Crawler using Selenium with Python to crawl through the online course (week by week) and download these files to the local filesystem. The transcripts were already available as “.txt” files but the slides were in “.pdf” which was not suitable for Text Mining. The team also created a scrapper to convert these “.pdf” slides to raw text files.

We explored two different approaches for our models, which are described below:

Model 1: LDA + Relevance-score filtering

- Raw files (transcripts and slides) were tokenized and pre-processed:
    a. Tokenization using NLTK RegexpTokenizer
    b. Removing stop words provided both in NLTK and Gensim
    c. Lemmatization using NLTK WordNetLemmatizer
    d. Generating “bigrams” per “tokenized doc” using collocation statistics, and adding to the corresponding tokenized doc

- Tokenized docs were transformed to Corpus in MM (Market Metrics) format.
    a. Dictionary, representing terms (tokens) as VSM (Vector Space Model), was generated using tokenized docs and persisted to local disk.
    b. Corpus, in MM (Market Metrics) format, was generated, using tokenized docs and the dictionary, and persisted to local disk.

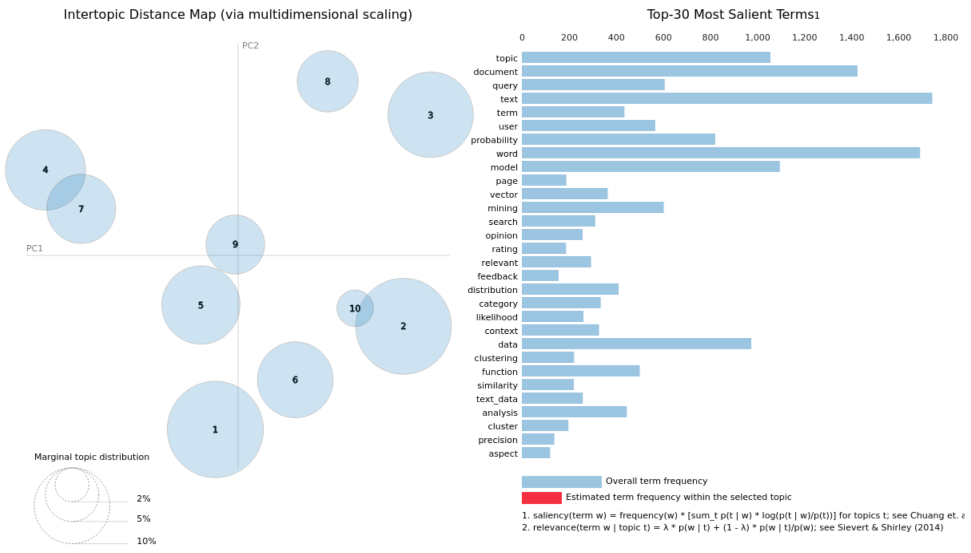
- TF-IDF model was generated using the MM Corpus, and persisting to local disk

- A base model was created by applying **LDA** with the following parameters:
    a. num_topics = 16 ⇒ Number of topics to discover. This needs to be decided empirically
    b. corpus = MM Corpus generated in the previous step
    c. id2word = Id to term mapping, provided by Dictionary generated in the previous step
    d. chunksize = 2000 ⇒ controls how many documents are processed at a time in the training algorithm. We kept this value much bigger than the corpus size, so all documents were used at the same time during training.
    e. alpha = 'auto' ⇒ A-priori belief on document-topic Dirichlet distribution parameter. We left it as "auto" which means that the algorithm will find the best value.
    f. eta = 'auto' ⇒ A-priori belief on topic-word Dirichlet distribution parameter. We left it as "auto" which means that the algorithm will find the best value.
    g. iterations = 800 ⇒ Maximum number of iterations through the corpus when inferring the topic distribution of a corpus.
    h. passes = 40 ⇒ controls how often we train the model on the entire corpus.

  Some of the above parameters like iterations and passes are internal to the Gensim implementation of LDA and can only be decided empirically. The number of topics is also decided based on empirical evidence and depends on how the topics will be leveraged. In our case, we need to interpret the topics, so we kept this value to a lower count. When we set the value too low or too high, the generated topics were hard to interpret. (This is generally a difficult aspect of topic mining and subjective to human interpretation.)

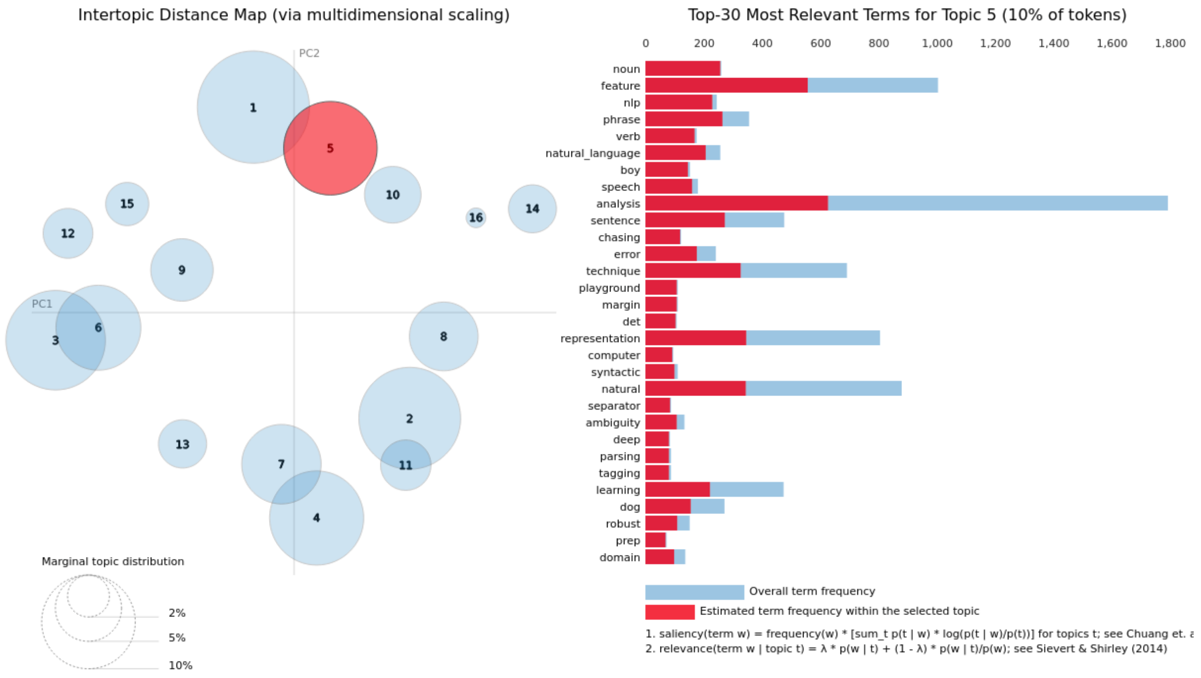
  We inspected pyLDAVis output for our model. The below images capture the output for 10 topics and 16 topics. From the below images, we can infer that with 16 topics we have overall more coverage of the corpus, while still maintaining decent inter-topic distances.

  For topic count = 10:



1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. a
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

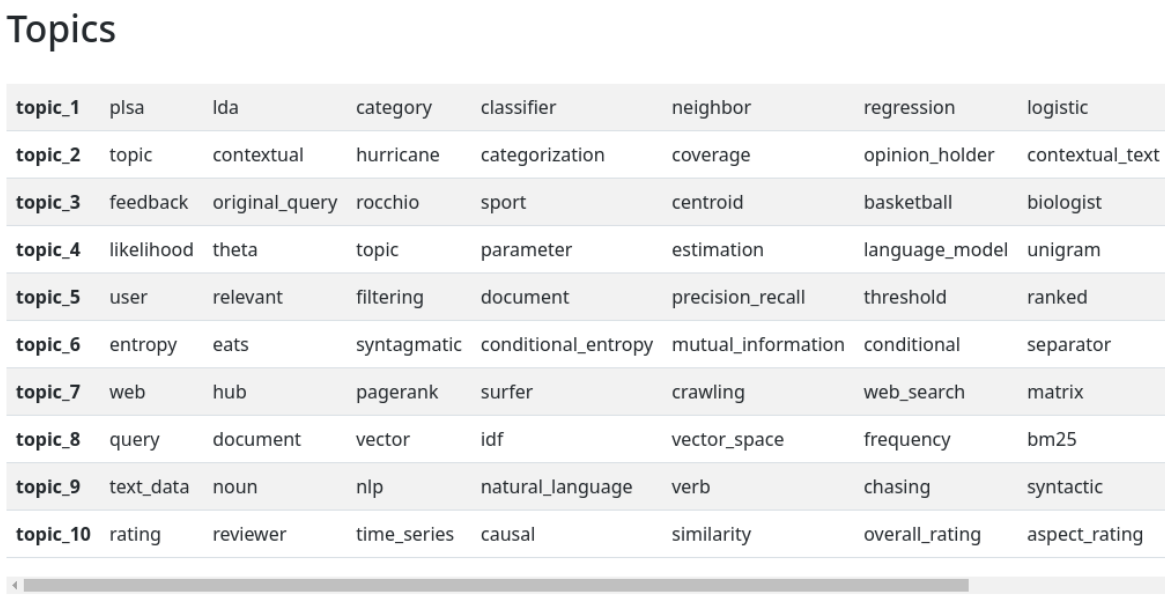
For topic count = 16:



1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. a
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

We also manually inspected the top terms for different models corresponding to different topic sizes. For example, below images show the top terms generated (by final models) when size was chosen as 10 and 16. As observed, some important concepts like “ndcg” in topic 2 of model with topic_size = 16, are missing in model with size = 10. This is probably due to the fact that multiple “independent” concepts are getting “squished” in a single topic and not making it to the top term list of that topic.

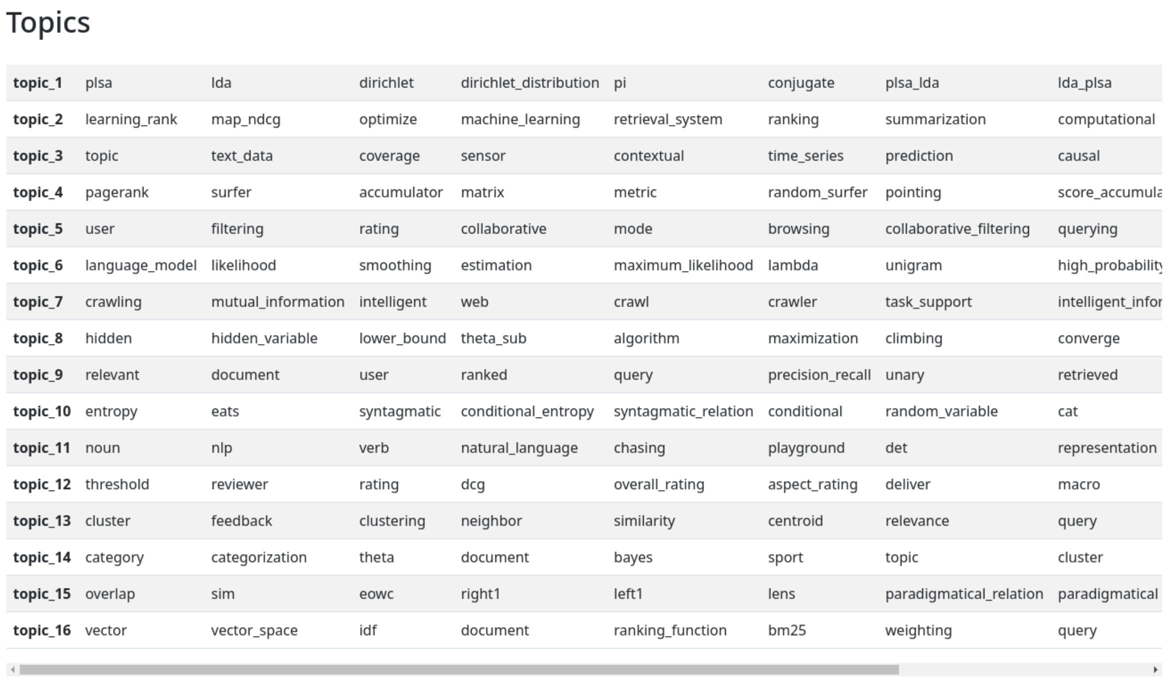
For topic size = 10:



Topics

topic_1	plsa	lda	category	classifier	neighbor	regression	logistic
topic_2	topic	contextual	hurricane	categorization	coverage	opinion_holder	contextual_text
topic_3	feedback	original_query	rocchio	sport	centroid	basketball	biologist
topic_4	likelihood	theta	topic	parameter	estimation	language_model	unigram
topic_5	user	relevant	filtering	document	precision_recall	threshold	ranked
topic_6	entropy	eats	syntagmatic	conditional_entropy	mutual_information	conditional	separator
topic_7	web	hub	pagerank	surfer	crawling	web_search	matrix
topic_8	query	document	vector	idf	vector_space	frequency	bm25
topic_9	text_data	noun	nlp	natural_language	verb	chasing	syntactic
topic_10	rating	reviewer	time_series	causal	similarity	overall_rating	aspect_rating

For topic size = 16:



Topics

topic_1	plsa	lda	dirichlet	dirichlet_distribution	pi	conjugate	plsa_lda	lda_plsa
topic_2	learning_rank	map_ndcg	optimize	machine_learning	retrieval_system	ranking	summarization	computational
topic_3	topic	text_data	coverage	sensor	contextual	time_series	prediction	causal
topic_4	pagerank	surfer	accumulator	matrix	metric	random_surfer	pointing	score_accumula
topic_5	user	filtering	rating	collaborative	mode	browsing	collaborative_filtering	querying
topic_6	language_model	likelihood	smoothing	estimation	maximum_likelihood	lambda	unigram	high_probabilit
topic_7	crawling	mutual_information	intelligent	web	crawl	crawler	task_support	intelligent_infor
topic_8	hidden	hidden_variable	lower_bound	theta_sub	algorithm	maximization	climbing	converge
topic_9	relevant	document	user	ranked	query	precision_recall	unary	retrieved
topic_10	entropy	eats	syntagmatic	conditional_entropy	syntagmatic_relation	conditional	random_variable	cat
topic_11	noun	nlp	verb	natural_language	chasing	playground	det	representation
topic_12	threshold	reviewer	rating	dcg	overall_rating	aspect_rating	deliver	macro
topic_13	cluster	feedback	clustering	neighbor	similarity	centroid	relevance	query
topic_14	category	categorization	theta	document	bayes	sport	topic	cluster
topic_15	overlap	sim	eowc	right1	left1	lens	paradigmatical_relation	paradigmatical
topic_16	vector	vector_space	idf	document	ranking_function	bm25	weighting	query

In the end, we selected the model with 16 pics. As discussed earlier, this is just an empirically decided number and may not reflect the “best” choice.

- We found that the topic terms included a lot of background words with high probability. So, we applied additional filtering using “Brown” corpus and “Reuters” corpus from NLTK, as “background” models and filtering the words which had high probability in these background models. This helped remove a lot of “noise” from topic-word distributions.

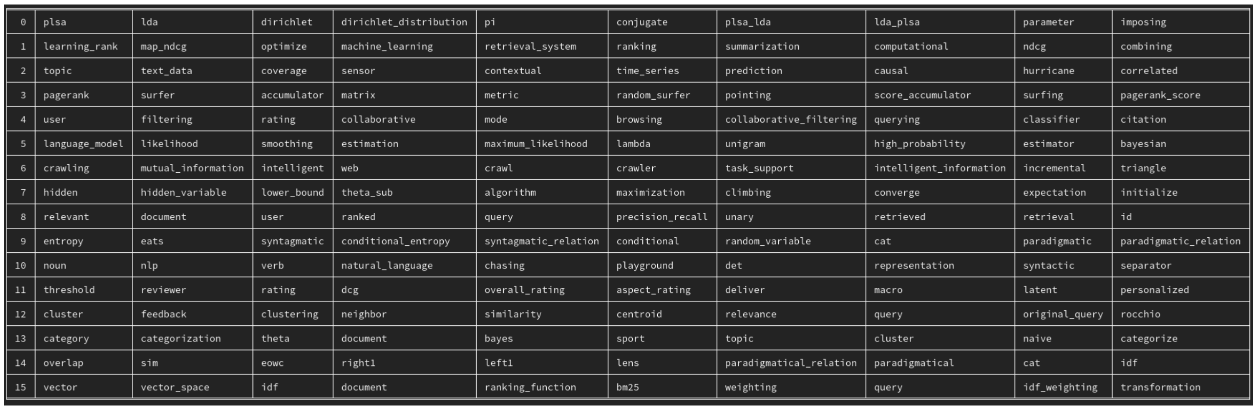
- The multinomial word distributions per topic, generated from basic LDA and above “background words” filtering were not of very good quality. The topics seemed to have a lot of “overlapping” top terms. We then explored the ways to boost probability of terms which could differentiate topics more and provide better interpretability. Here, we applied the concept **“relevance-score”** for topic terms discussed in paper: https://nlp.stanford.edu/events/illvi2014/papers/sievert-illvi2014.pdf, which basically provides a score per term in a topic, as

$$r(w,\ t\,|\,\lambda)\ =\ \lambda\, log(p(w,\ t))\ +\ (1\ -\ \lambda)\, log(p(w,\ t)\ /\ p(w))$$

where,
$r(w,\ t\,|\,\lambda)\ =\ relevance\ score\ for\ term\ w\ in\ topic\ t,\ conditional\ on\ parameter\ \lambda$
$\lambda\ =\ tuning\ parameter.\ Setting\ value\ 1\ will\ make\ this\ behave\ like\ normal\ topic\ model$
$p(w,\ t)\ =\ probability\ of\ term\ w\ in\ topic\ t$
$p(w)\ =\ term\ probability\ in\ corpus$

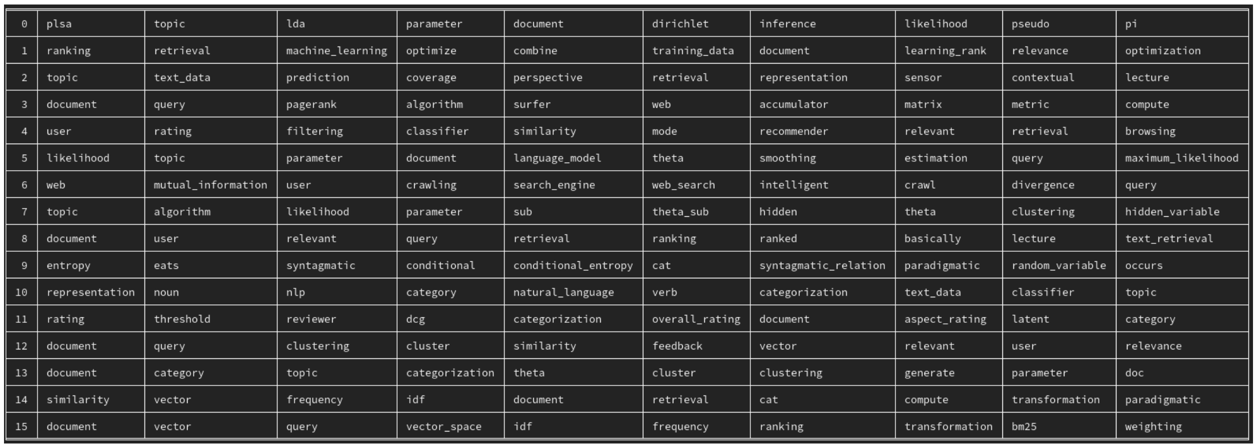
Once the scoring was done for each term in a given topic, we could filter out top “n” terms which resulted in more cohesive easily interpretable topics. Below diagram shows the impact of removing “relevance score” filtering (with λ = 1.0). We observe a lot of common terms like “topic”, “document”, “user” etc. show up as top words in multiple topics. As we decrease the value of tuning parameter λ the topic terms become more topic-specific, helping in differentiating and interpreting topics.

With λ = 0.4



0	plsa	lda	dirichlet	dirichlet_distribution	pi	conjugate	plsa_lda	lda_plsa	parameter	imposing
1	learning_rank	map_ndcg	optimize	machine_learning	retrieval_system	ranking	summarization	computational	ndcg	combining
2	topic	text_data	coverage	sensor	contextual	time_series	prediction	causal	hurricane	correlated
3	pagerank	surfer	accumulator	matrix	metric	random_surfer	pointing	score_accumulator	surfing	pagerank_score
4	user	filtering	rating	collaborative	mode	browsing	collaborative_filtering	querying	classifier	citation
5	language_model	likelihood	smoothing	estimation	maximum_likelihood	lambda	unigram	high_probability	estimator	bayesian
6	crawling	mutual_information	intelligent	web	crawl	crawler	task_support	intelligent_information	incremental	triangle
7	hidden	hidden_variable	lower_bound	theta_sub	algorithm	maximization	climbing	converge	expectation	initialize
8	relevant	document	user	ranked	query	precision_recall	unary	retrieved	retrieval	id
9	entropy	eats	syntagmatic	conditional_entropy	syntagmatic_relation	conditional	random_variable	cat	paradigmatic	paradigmatic_relation
10	noun	nlp	verb	natural_language	chasing	playground	det	representation	syntactic	separator
11	threshold	reviewer	rating	dcg	overall_rating	aspect_rating	deliver	macro	latent	personalized
12	cluster	feedback	clustering	neighbor	similarity	centroid	relevance	query	original_query	rocchio
13	category	categorization	theta	document	bayes	sport	topic	cluster	naive	categorize
14	overlap	sim	eowc	right1	left1	lens	paradigmatical_relation	paradigmatical	cat	idf
15	vector	vector_space	idf	document	ranking_function	bm25	weighting	query	idf_weighting	transformation

With λ = 1.0 (which means no additional filtering)



0	plsa	topic	lda	parameter	document	dirichlet	inference	likelihood	pseudo	pi
1	ranking	retrieval	machine_learning	optimize	combine	training_data	document	learning_rank	relevance	optimization
2	topic	text_data	prediction	coverage	perspective	retrieval	representation	sensor	contextual	lecture
3	document	query	pagerank	algorithm	surfer	web	accumulator	matrix	metric	compute
4	user	rating	filtering	classifier	similarity	mode	recommender	relevant	retrieval	browsing
5	likelihood	topic	parameter	document	language_model	theta	smoothing	estimation	query	maximum_likelihood
6	web	mutual_information	user	crawling	search_engine	web_search	intelligent	crawl	divergence	query
7	topic	algorithm	likelihood	parameter	sub	theta_sub	hidden	theta	clustering	hidden_variable
8	document	user	relevant	query	retrieval	ranking	ranked	basically	lecture	text_retrieval
9	entropy	eats	syntagmatic	conditional	conditional_entropy	cat	syntagmatic_relation	paradigmatic	random_variable	occurs
10	representation	noun	nlp	category	natural_language	verb	categorization	text_data	classifier	topic
11	rating	threshold	reviewer	dcg	categorization	overall_rating	document	aspect_rating	latent	category
12	document	query	clustering	cluster	similarity	feedback	vector	relevant	user	relevance
13	document	category	topic	categorization	theta	cluster	clustering	generate	parameter	doc
14	similarity	vector	frequency	idf	document	retrieval	cat	compute	transformation	paradigmatic
15	document	vector	query	vector_space	idf	frequency	ranking	transformation	bm25	weighting

- Next, for defining top terms for each document (in corpus), based on topic distribution for the given document, we chose to include only the terms from (significant) topics, sorting them based on **TF-IDF** score of that term in the given document. Since our end goal was to create a sort of index of the (discovered) concepts and relevant documents (where they occur), we wanted only to consider those topic terms which had a high TF-IDF score for a given document.

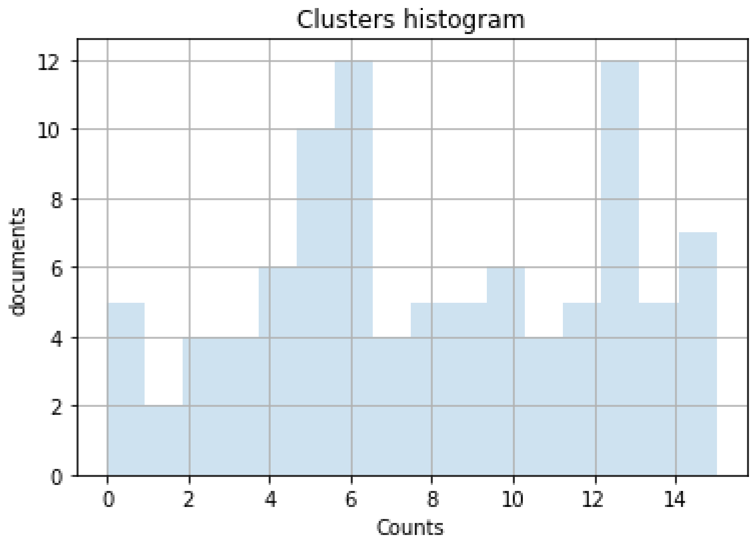
Once all above steps were completed, we ended up with a topic distribution which was somewhat interpretable. We could then build **indexes** for our application:

- Term - Document Index: This index maps each (discovered) term to a set of documents, where the term has occurred with prominence (based on the process described in above steps).
- Topic - Document Index: This index maps each topic to a set of documents, where the topic was covered prominently (based on the process described in above steps).

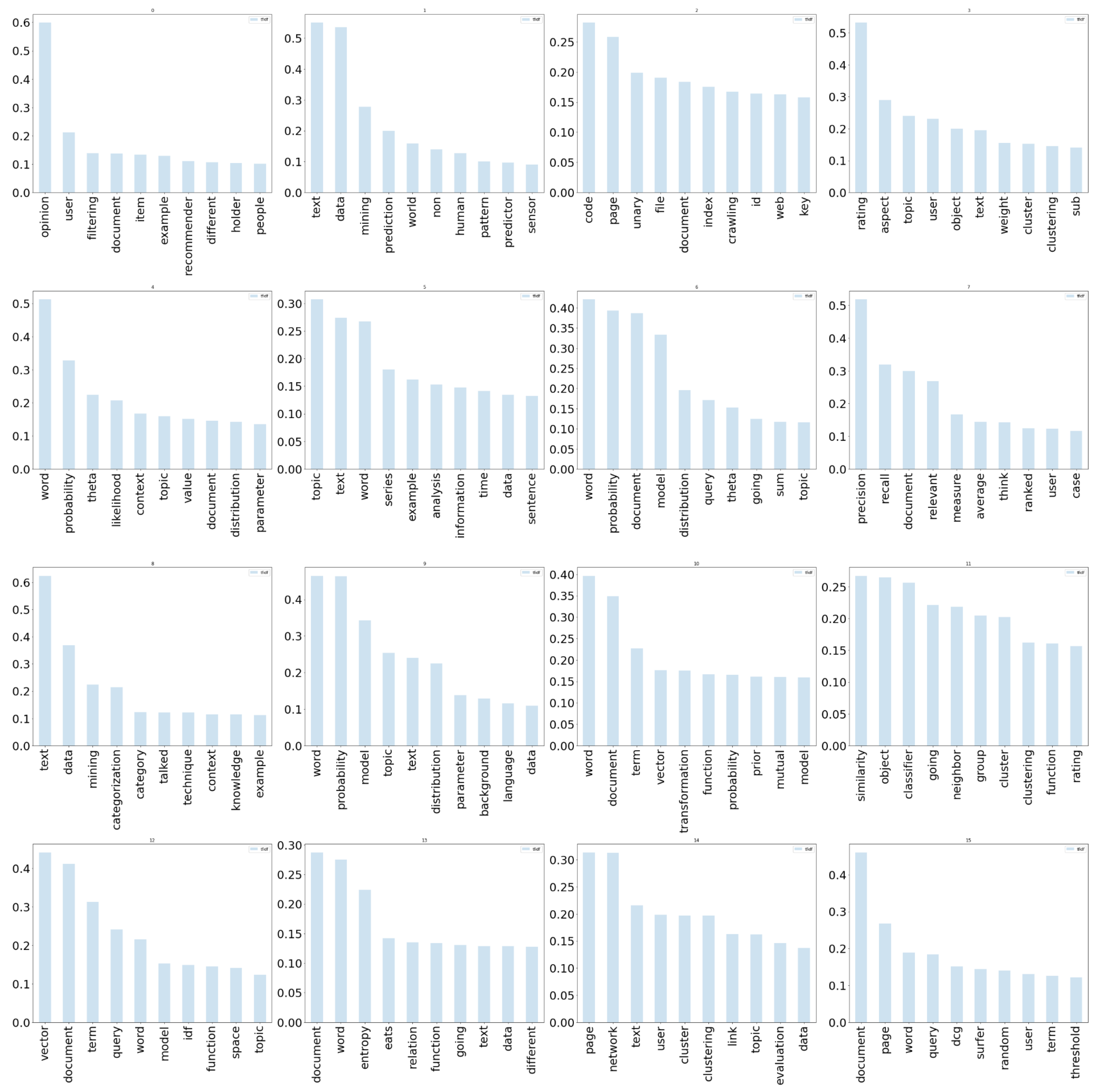
Model 2: KMeans Clustering

This model uses a K-means clustering algorithm and trains the model with the entire dataset. A parameter of 16 clusters was used, and for word representation we utilized Word2vec embeddings with continuous bag-of-words (CBOW) architecture, which took care of the semantic and meaning. Word2vec allows to capture synonyms and word's context by using a neural network pre-trained on a large dataset.

For KMeans clustering we calculate the distance between the document and the centroid of the closest cluster, and the closest centroid will represent the closest matching topic. With this approach, we can say only if the document is a part of a certain cluster, but not to rank the matching topics. There is another limitation of KMeans model: it does not assign weights to features, so extracting meaningful words requires additional heuristic.



For analysis of produced clusters, we created a side-by-side barchart of 5 most popular words in all 16 clusters. These terms are quite generic to provide meaningful information, so we switched to tfidf. This approach helped to get a better understanding of word distribution in documents. However, there is still no easy way to evaluate the performance of a model.

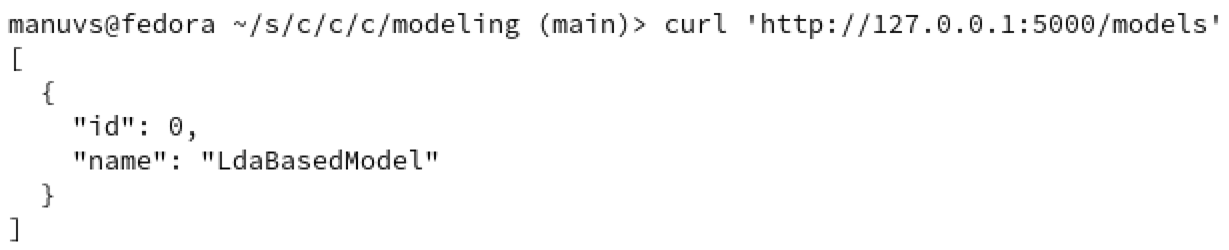


The clustering approach proved to be successful, since it put together transcripts belonging to the same week or having a similar theme, but due to limitations of the model, we did not include it into the app.

REST API

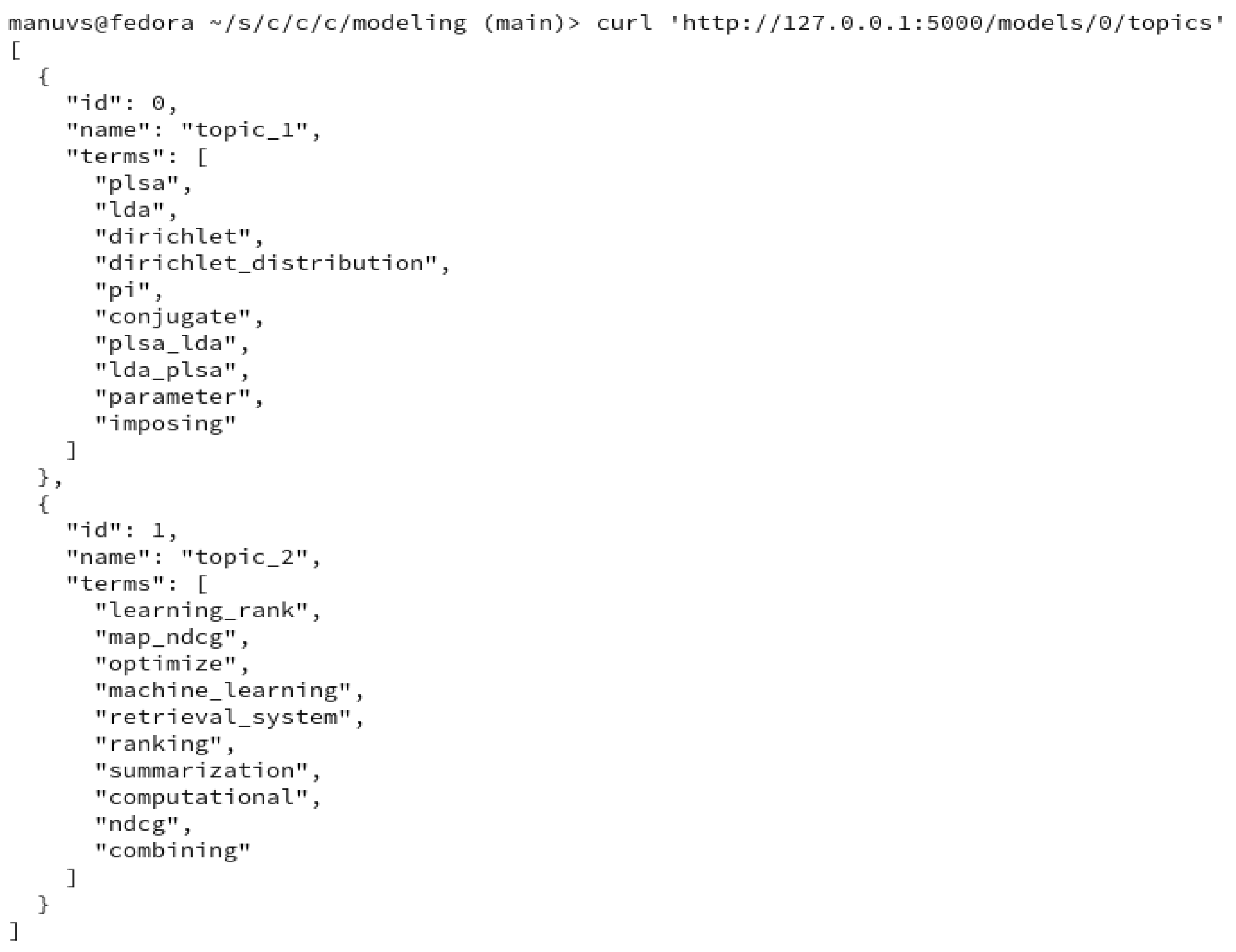
We created an REST API layer to expose the data generated by our model(s), which can be consumed directly or in the UI layer. Below, we describe all the resources exposed by our API:

- **GET /models** ⇒ This returns the models available as part of our application. As described above, we built two different models, using different heuristics.



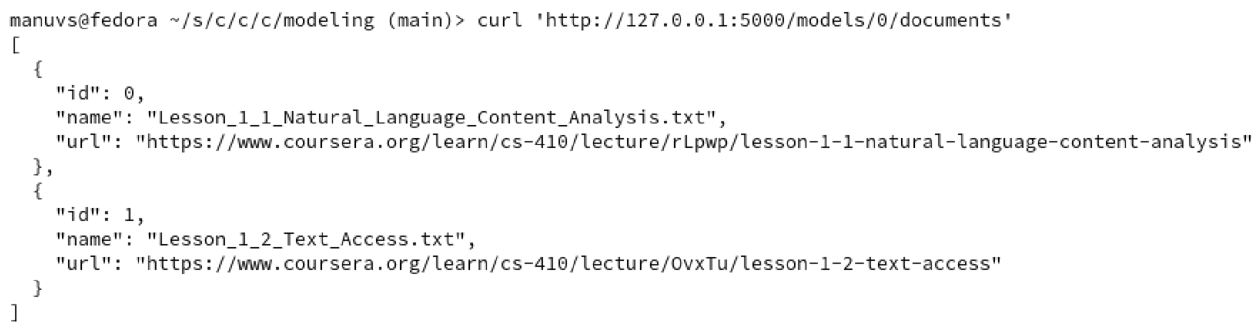
```
manuvs@fedora ~/s/c/c/c/modeling (main)> curl 'http://127.0.0.1:5000/models'
[
  {
    "id": 0,
    "name": "LdaBasedModel"
  }
]
```

- **GET /models/<model_id>/topics** ⇒ This returns topics discovered by this model.



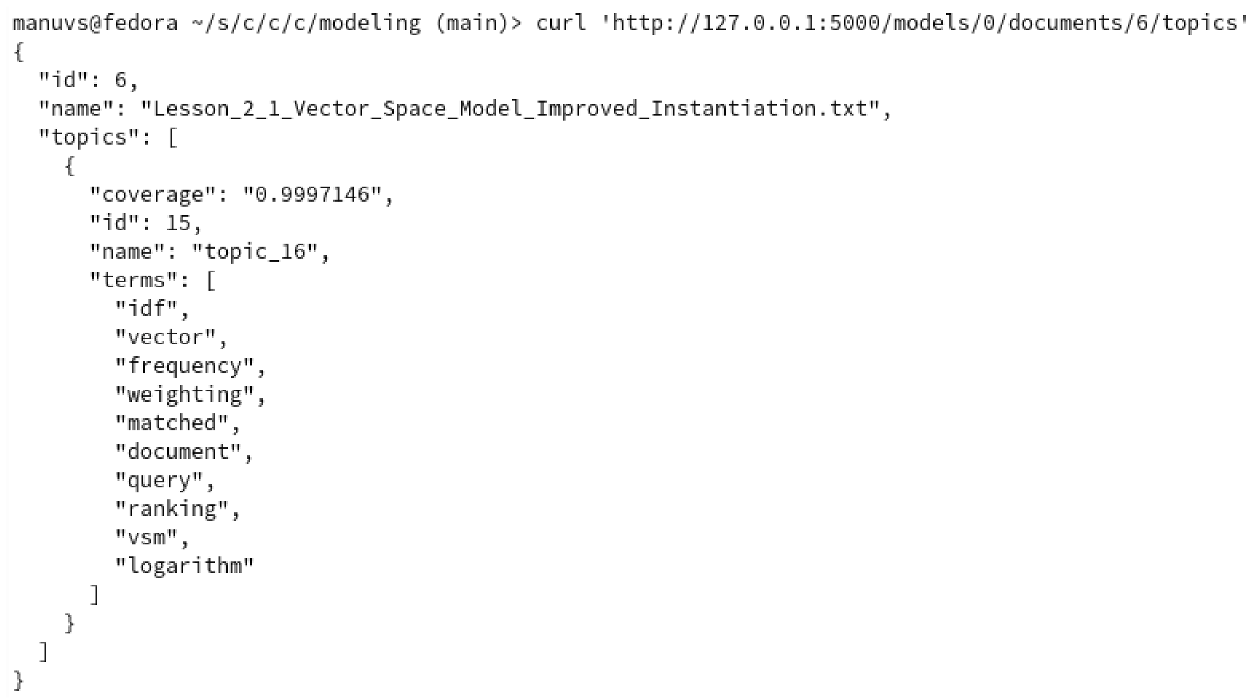
```
manuvs@fedora ~/s/c/c/c/modeling (main)> curl 'http://127.0.0.1:5000/models/0/topics'
[
  {
    "id": 0,
    "name": "topic_1",
    "terms": [
      "plsa",
      "lda",
      "dirichlet",
      "dirichlet_distribution",
      "pi",
      "conjugate",
      "plsa_lda",
      "lda_plsa",
      "parameter",
      "imposing"
    ]
  },
  {
    "id": 1,
    "name": "topic_2",
    "terms": [
      "learning_rank",
      "map_ndcg",
      "optimize",
      "machine_learning",
      "retrieval_system",
      "ranking",
      "summarization",
      "computational",
      "ndcg",
      "combining"
    ]
  }
]
```

- **GET /models/<int:model_id>/documents** ⇒ This returns the corpus list used by this model



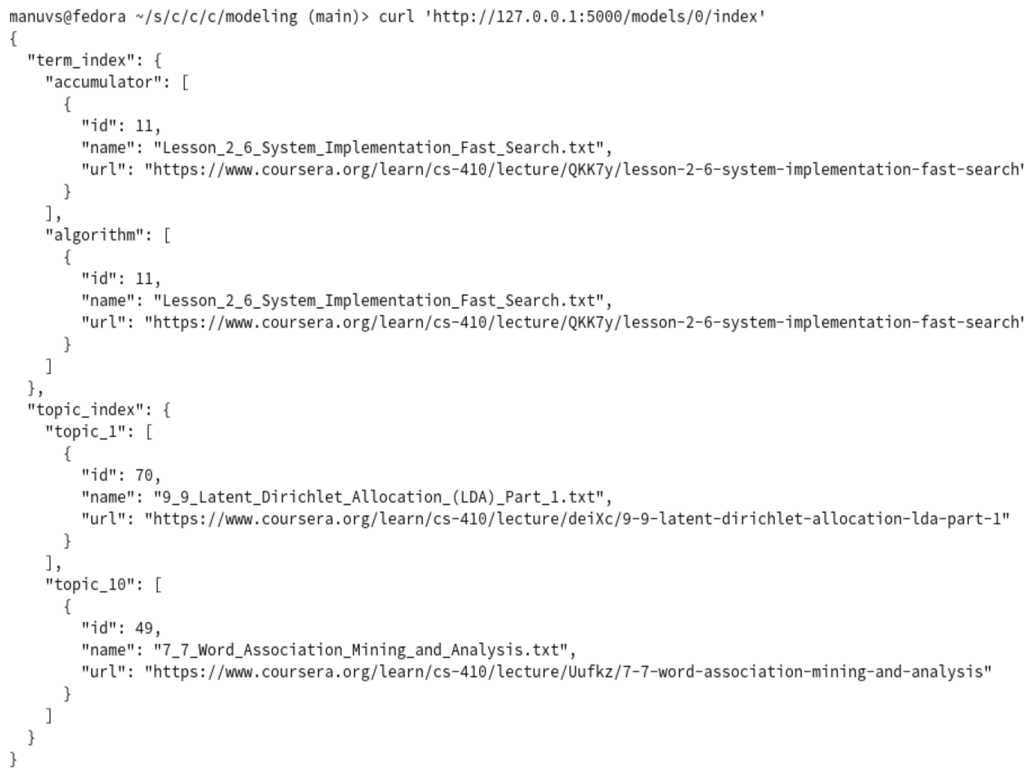
```
manuvs@fedora ~/s/c/c/c/modeling (main)> curl 'http://127.0.0.1:5000/models/0/documents'
[
  {
    "id": 0,
    "name": "Lesson_1_1_Natural_Language_Content_Analysis.txt",
    "url": "https://www.coursera.org/learn/cs-410/lecture/rLpwp/lesson-1-1-natural-language-content-analysis"
  },
  {
    "id": 1,
    "name": "Lesson_1_2_Text_Access.txt",
    "url": "https://www.coursera.org/learn/cs-410/lecture/OvxTu/lesson-1-2-text-access"
  }
]
```

- **GET /models/<int:model_id>/documents/<int:doc_id>/topics** ⇒ This returns the topic distribution of this document.



```
manuvs@fedora ~/s/c/c/c/modeling (main)> curl 'http://127.0.0.1:5000/models/0/documents/6/topics'
{
  "id": 6,
  "name": "Lesson_2_1_Vector_Space_Model_Improved_Instantiation.txt",
  "topics": [
    {
      "coverage": "0.9997146",
      "id": 15,
      "name": "topic_16",
      "terms": [
        "idf",
        "vector",
        "frequency",
        "weighting",
        "matched",
        "document",
        "query",
        "ranking",
        "vsm",
        "logarithm"
      ]
    }
  ]
}
```

- **GET /models/<int:model_id>/index** ⇒ Returns the index (described earlier in this document), built for this model.



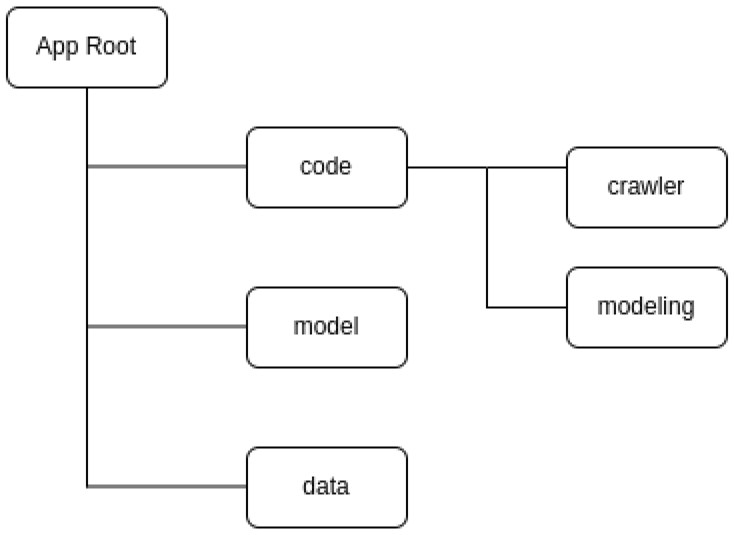
```
manuvs@fedora ~/s/c/c/c/modeling (main)> curl 'http://127.0.0.1:5000/models/0/index'
{
  "term_index": {
    "accumulator": [
      {
        "id": 11,
        "name": "Lesson_2_6_System_Implementation_Fast_Search.txt",
        "url": "https://www.coursera.org/learn/cs-410/lecture/QKK7y/lesson-2-6-system-implementation-fast-search'
      }
    ],
    "algorithm": [
      {
        "id": 11,
        "name": "Lesson_2_6_System_Implementation_Fast_Search.txt",
        "url": "https://www.coursera.org/learn/cs-410/lecture/QKK7y/lesson-2-6-system-implementation-fast-search'
      }
    ]
  },
  "topic_index": {
    "topic_1": [
      {
        "id": 70,
        "name": "9_9_Latent_Dirichlet_Allocation_(LDA)_Part_1.txt",
        "url": "https://www.coursera.org/learn/cs-410/lecture/deiXc/9-9-latent-dirichlet-allocation-lda-part-1"
      }
    ],
    "topic_10": [
      {
        "id": 49,
        "name": "7_7_Word_Association_Mining_and_Analysis.txt",
        "url": "https://www.coursera.org/learn/cs-410/lecture/Uufkz/7-7-word-association-mining-and-analysis"
      }
    ]
  }
}
```

All these APIs were exposed by a **Python Flask** application.

SOURCE CODE

The source code is available at https://github.com/manuv3/cs410-project

The application has following structure:



- code: This directory contains two modules crawler and modeling. The crawler directory contains Selenium based crawler, which was used to browse and extract transcripts and slides from Coursera CS410 course. The modeling directory contains all the modules used to build models for this project. The main modules are described below:
  a. corpus.py ⇒ Module to build and persist Dictionary, Corpus, and TF-IDF model.
  b. lda.py ⇒ Module to build and persist base LDA model
  c. topics.py ⇒ Module to generate wrapper models over base LDA model, by applying additional heuristics (described earlier).
  d. cluster_topics.py ⇒ Module to generate topics based on Document clustering.
  e. app.py ⇒ Flask app module to expose the API and UI resources.

- model: This directory contains all the generated Dictionary, MM Format Corpus, and LDA model, Word2Vec model etc.

- data: This directory contains the raw corpus (transcripts and slides) and NLTK-provided corpuses.

Documentation of Usages of Languages & Tools:

We used the following tools for the project:
- Gensim ⇒ For topic modeling (LDA, Word2Vec) and other NLP tasks
- Flask ⇒ For web application server
- Google charts ⇒ For charts in HTMLs
- Bootstrap ⇒ For responsive HTML layouts

- Jinja2 ⇒ For server side HTML templating

Additionally, we used the following languages for the project:
- Python
- Javascript
- HTML5

Documentation of the Usage of the Software

How to Install/Start the Software:

The software has two parts:

- Model creation: The models were created in "offline" mode (as described in the previous section). Any future tweaking of models or experimenting with new models will need to be done in the same way. The idea is to develop models offline and persist them in the file system and then load them on server startup.

- The web application server is Python Flask based. It exposes the API (described in previous section) as well as UI component.

  The application is largely Python based. So having Python 3.x and pip installed locally is a prerequisite. The application uses many libraries (like Gensim, pandas etc.). All necessary packages need to be installed in the local Python environment. This can be done by executing following commands (in a bash-like CLI) from application root directory:

  ```
  cd code
  pip install -r requirements.txt
  ```

  The flask server can be run by executing following command:

  ```
  cd modeling
  export FLASK_ENV=development; flask run
  ```

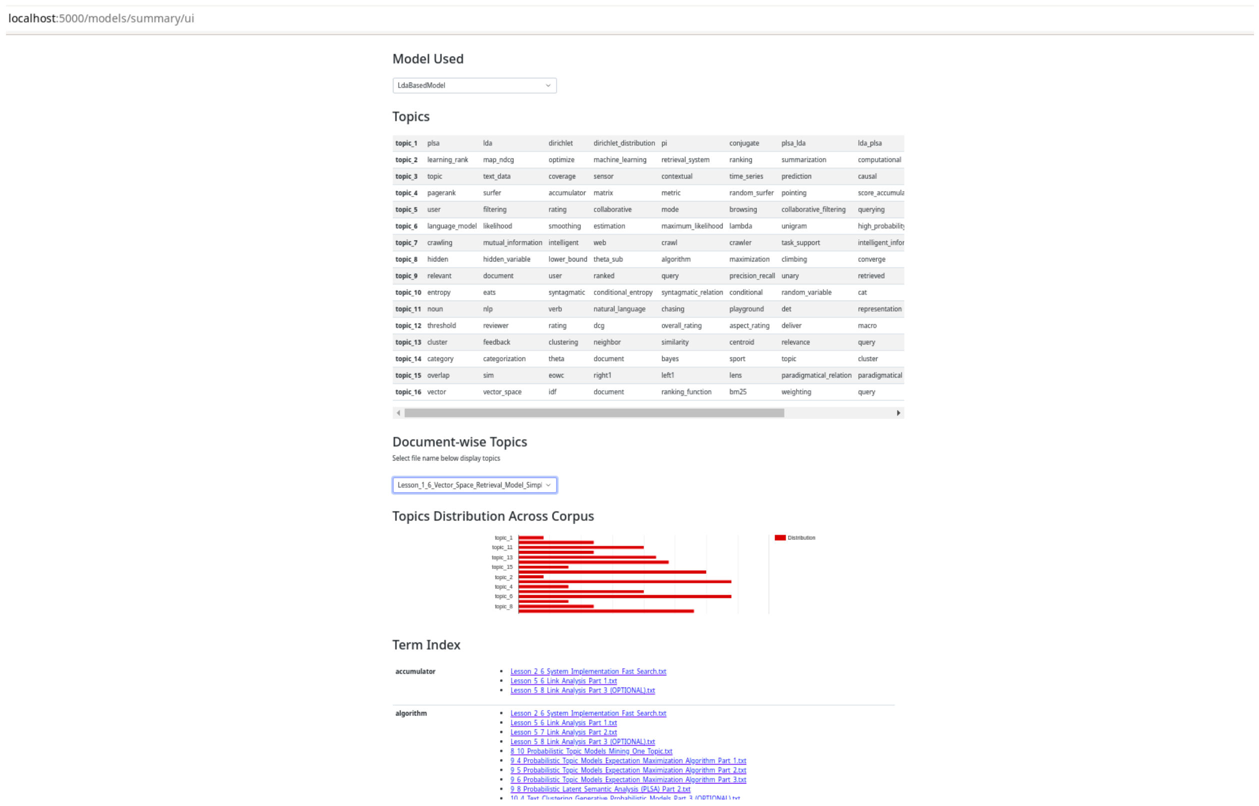
How to Run/Operate the Software:

Our original goal was to enhance the EducationalWeb UI directly, but we were not able to access the source code for EducationWeb. So, we decided to build a standalone UI (hosted by the same Python Flask application, which also exposes the APIs). These UI pages are light weight and can be integrated into EducationalWeb in future. The UIs are built using Bootstrap, Jinja2 template and Google charts.

The UI consists of 2 pages or views which are described below:

Model Summary page

This page can be accessed using resource: **GET /models/summary/ui**


localhost:5000/models/summary/ui

Model Used

LdaBasedModel

Topics

topic_1	plsa	lda	dirichlet	dirichlet_distribution	pi	conjugate	plsa_lda	lda_plsa
topic_2	learning_rank	map_ndcg	optimize	machine_learning	retrieval_system	ranking	summarization	computational
topic_3	topic	text_data	coverage	sensor	contextual	time_series	prediction	causal
topic_4	pagerank	surfer	accumulator	matrix	metric	random_surfer	pointing	score_accumula
topic_5	user	filtering	rating	collaborative	mode	browsing	collaborative_filtering	querying
topic_6	language_model	likelihood	smoothing	estimation	maximum_likelihood	lambda	unigram	high_probability
topic_7	crawling	mutual_information	intelligent	web	crawl	crawler	task_support	intelligent_infor
topic_8	hidden	hidden_variable	lower_bound	theta_sub	algorithm	maximization	climbing	converge
topic_9	relevant	document	user	ranked	query	precision_recall	unary	retrieved
topic_10	entropy	eats	syntagmatic	conditional_entropy	syntagmatic_relation	conditional	random_variable	cat
topic_11	noun	nlp	verb	natural_language	chasing	playground	det	representation
topic_12	threshold	reviewer	rating	dcg	overall_rating	aspect_rating	deliver	macro
topic_13	cluster	feedback	clustering	neighbor	similarity	centroid	relevance	query
topic_14	category	categorization	theta	document	bayes	sport	topic	cluster
topic_15	overlap	sim	eowc	right1	left1	lens	paradigmatical_relation	paradigmatical
topic_16	vector	vector_space	idf	document	ranking_function	bm25	weighting	query

Document-wise Topics

Select file name below display topics

Lesson_1_6_Vector_Space_Retrieval_Model_Simpl

Topics Distribution Across Corpus

Term Index

accumulator
- Lesson_2_6_System_Implementation_Fast_Search.txt
- Lesson_5_6_Link_Analysis_Part_1.txt
- Lesson_5_8_Link_Analysis_Part_3_(OPTIONAL).txt

algorithm
- Lesson_2_6_System_Implementation_Fast_Search.txt
- Lesson_5_6_Link_Analysis_Part_1.txt
- Lesson_5_7_Link_Analysis_Part_2.txt
- Lesson_5_8_Link_Analysis_Part_3_(OPTIONAL).txt
- 8_10_Probabilistic_Topic_Models_Mining_One_Topic.txt
- 9_4_Probabilistic_Topic_Models_Expectation_Maximization_Algorithm_Part_1.txt
- 9_5_Probabilistic_Topic_Models_Expectation_Maximization_Algorithm_Part_2.txt
- 9_6_Probabilistic_Topic_Models_Expectation_Maximization_Algorithm_Part_3.txt
- 9_8_Probabilistic_Latent_Semantic_Analysis_(PLSA)_Part_2.txt
- 10_4_Text_Clustering_Generative_Probabilistic_Models_Part_3_(OPTIONAL).txt

This page has following sections:

- **Model Used** ⇒ This section provides a dropdown to select one of the available models. This can be used to look at the results of different models. The whole design is extensible, in the sense that we can create more models in future and such a model will be available here.

- **Topics** ⇒ This section lays down a (responsive) table depicting the topics and top-10 terms for each topic. This will provide a clear picture to the user about the topics discovered using the model.

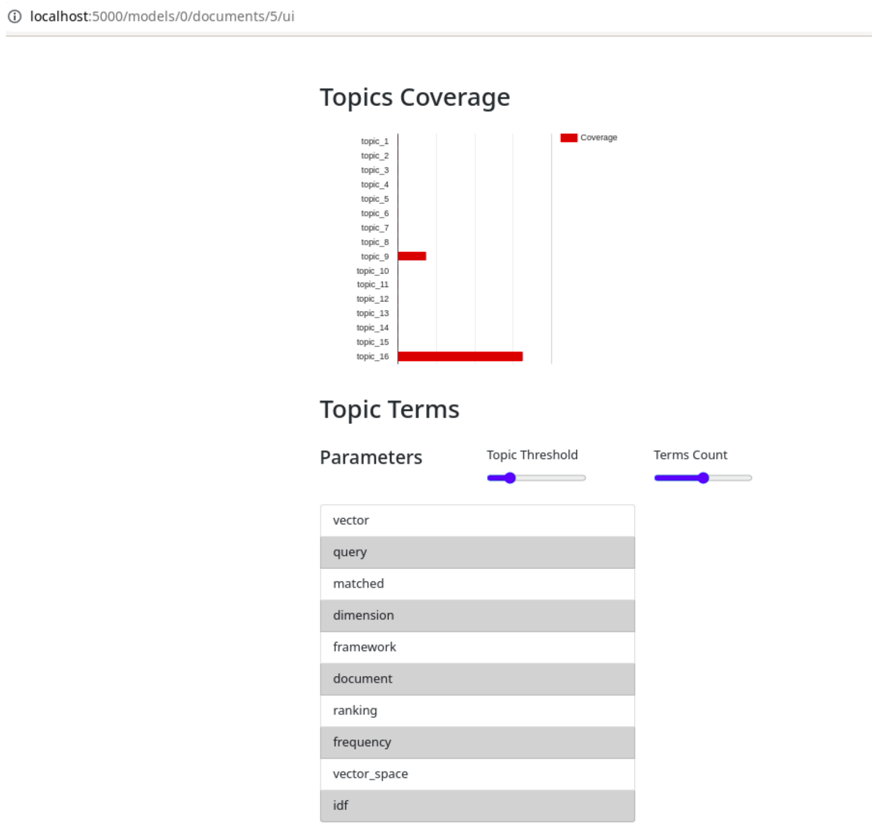
- **Document-wise Topics** ⇒ This section provides a dropdown list of documents (in the corpus). The user can select one at a time to open up another page which gives details of the topics covered in the selected document. (More details about this page are provided later in this document.)

- **Topic Distribution Across Corpus** ⇒ This section provides a bar-chart depicting the distribution of topics over the whole corpus. This can be useful in interpreting the performance of the current model and may bring more insights.

- **Term Index** ⇒ This section lays out the index of all “discovered” terms and the URLs of lectures in which they occur. A user can look at this index and get a gist of what this course covers. This section can be “potentially” used in the same way as one uses the “index” at the back of a book.

Document-topic page

This page is accessed using resource: GET
/models/<model_id>/documents/<document_id>/ui


localhost:5000/models/0/documents/5/ui

Topics Coverage

Topic Terms

Parameters

vector
query
matched
dimension
framework
document
ranking
frequency
vector_space
idf

This page has following sections:

- **Topics Coverage** ⇒ This section depicts a bar chart showing the topic distribution in the given document.
- **Topic Terms** ⇒ This section provides a list of top n terms discovered for this document, based on the model selected. There are basic controls: Topic Threshold and Terms Count, which can be changed to reflect fewer or more terms for this document.

Description of Team Member Contributions

- **Manu Vinod Shesha:**
    - Conceived and curated the project idea.
    - Researched on viability of the project by going through research papers and multiple discussions with the Professor during OH sessions.
    - Created the proposal document and coordinated with the team to review.
    - Built the LDA based along with all the additional heuristics.
    - Built the API and UI layers
    - Provided the project progress report (after doing research and evaluation of different algorithms)
    - Provided the major chunk of final software code documentation and also the demo.
    - Was also the captain of the team. So performed all necessary admin tasks.
- **Angus Jyu:**
    - Researched the EducationalWeb source and looked into the UI to understand and try and project its viability for application use in the project.
    - Explored the viability of using other datasets from other websites, as well as coming up with possible blueprints as to what the application for our project could look like (ex. Looked into using YouTube videos as a possible dataset instead of Coursera videos).
    - Directed some of the discussions to discuss implementation of Sofia's model/code into the project.
    - Reviewed the topics and corpus files.

- Worked on the entirety of the presentation video and powerpoint, writing all the slides and directing the slides.
- Wrote a lot of the software code documentation and progress reporting and also helped source the references.

★ **Sofia Godovykh:**

- Built the word2vec model and fed it to a k-means clustering model, and added some heuristical data cleaning.
  - This included removing stopwords like sound and music, and additionally cleaned out pdf data.
  - Reproduced Manu's modelling, to get an alternative model performance.
- Examined what LDA2vec has to offer through having to read articles like "LDA2vec: Word Embeddings in Topic Models", if you don't have better ideas.
- Researched topic modeling using Word Embeddings and Latent Dirichlet Allocation, as well as articles regarding topic Modelling from a million news headlines.

References

Gensim: Topic modeling for humans. Radim Å˜ehÅ¯Å™ek: Machine learning consulting. (2021, August 30). Retrieved December 8, 2021, from https://radimrehurek.com/gensim/.

Blei, D. M., Ng, A. Y., &amp; Jordan, M. I. (2003). Latent Dirichlet Allocation. Journal of Machine Learning Research, 3, 1–30.

Pyldavis¶. pyLDAvis. (n.d.). Retrieved December 1, 2021, from https://pyldavis.readthedocs.io/en/latest/readme.html

NLTK. (n.d.). Retrieved December 1, 2021, from https://www.nltk.org/.

Google. (n.d.). Charts &nbsp;|&nbsp; google developers. Google. Retrieved December 10, 2021, from https://developers.google.com/chart.

Mark Otto, J. T. (n.d.). Bootstrap. Bootstrap · The most popular HTML, CSS, and JS library in the world. Retrieved December 10, 2021, from https://getbootstrap.com/.

Template designer documentation¶. Template Designer Documentation - Jinja Documentation (2.11.x). (n.d.). Retrieved December 10, 2021, from https://jinja.palletsprojects.com/en/2.10.x/templates/.

Welcome to flask¶. Welcome to Flask - Flask Documentation (2.0.x). (n.d.). Retrieved December 10, 2021, from https://flask.palletsprojects.com/en/2.0.x.

Garodia, S. (2020, January 21). Topic modelling using word embeddings and latent Dirichlet allocation. Medium. Retrieved December 10, 2021, from https://medium.com/analytics-vidhya/topic-modelling-using-word-embeddings-and-latent-dirichlet-allocation-3494778307bc.

Garodisk. (n.d.). Topic-modelling-on-a-million-news-headlines/topic_modelling_from_a_million_news_headlines.ipynb at master · garodisk/topic-modelling-on-a-million-news-headlines. GitHub. Retrieved December 10, 2021, from https://github.com/garodisk/Topic-Modelling-on-a-million-news-headlines/blob/master/Topic_Modelling_from_a_million_news_headlines.ipynb.

LDA2vec topic modelling. DataCamp Community. (n.d.). Retrieved December 10, 2021, from https://www.datacamp.com/community/tutorials/lda2vec-topic-model.

Mei, Q., Shen, X., &amp; Zhai, C. X. (2007). Automatic labeling of multinomial topic models. Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '07. https://doi.org/10.1145/1281192.1281246

University of Illinois. (n.d.). Timan102. CS at University of Illinois. Retrieved December 8, 2021, from http://timan102.cs.illinois.edu/explanation/.