Concept View

Tutorial Presentation Video Link: https://youtu.be/QRdiEo0Clgs

Overview of Functions:

Given the .vtt transcripts of a MOOC lecture, our project will output the concepts taught in the lecture in the form of a list of words/phrases that represent each concept. Our desired output would focus on the concepts taught in the lecture and exclude related concepts that are not taught in this lecture. We took two different approaches to this problem (Word2Vec and LDA) which will be described below.

Word2Vec-Based Approach (under 'lib' folder):

Implementation:

The workflow of the application is as follows:

1. Scan through the target corpus and background corpus to identify topic phrases.
2. Embed user corpus into a word2vec model trained with text8 dataset.
3. Mark target corpus with 1 and background corpus with 0. Train a ML classifier to identify interesting clusters of word vectors.
4. Use the previous ML model to rank sentences in target corpus and skim the documents by a user-specified factor.
5. Build a LDA model to learn topics covered in target and background corpuses.
6. Generate output with skimmed target corpus obtained in step 4.

We mainly used Gensim (4.1.3.dev0) for the project in achieving the goal of topic identification with LDA and as an interface to the word2vec model. We incorporated TensorFlow (2.3.0) and Keras API to build the ML model. We also incorporated numpy (1.20.3) and spacy (2.3.5) to speed up steps in this project.

Usage:

All text files should be in utf-8 encoding. Each document should be in a line and separated by a period.

LDA-Based Approach (under 'lda' folder):

Implementation:

1. Convert all lecture transcripts (".vtt" files) into a document with one line. Concatenate all lecture transcripts into a corpus D where each line represents a lecture document.
2. Tokenize D using NLTK RegexpTokenizer. Remove all numbers and single-character words. Lemmatize using WordNetLemmatizer in NLTK.
3. Generate all bigrams in D using gensim.models.Phrases. Generate all trigrams and 4-grams in D using the bigrams.
4. Append all n-grams (n>1) generated in step 3 to D and remove all unigrams in D because we observe from the human annotated dataset that few unigrams can represent concepts.
5. Turn D into a corpus C represented by bag-of-words using gensim.corpora.dictionary.
6. Train gensim.models.LdaModel on the entire corpus C with number of topics set to 5 * number_of_lectures.
7. For each lecture document, find the top topic and use the top 20 topic_words as the concept of the lecture.

Example outputs (lda_result.txt):

```
lecture 2
topic 1
pull_mode, 0.05688284710049629
help_user, 0.037936534732580185
type_in, 0.0284633785486213
high_level_strategy, 0.0284633785486213
don_know, 0.0284633785486213
information_access, 0.0284633785486213
recommender_system, 0.0284633785486213
tends_to_be, 0.0284633785486213
text_data, 0.0284633785486213
```

Results:

Use the human annotated dataset (lda/ConceptView_ Student Generated Concepts (1).xlsx) for evaluation. Calculate the recall of each lecture (i.e. how many concepts are retrieved by our method) We set a threshold t such that a concept C1 in the human annotated corpus is considered retrieved in our result if there is a concept C2 in our result such that (#words that co-appear in both C1 and C2) / (#words in C2) >= t. We did this because there are slight differences between our results and the human annotated dataset due to lemmatization.

Example outputs of evaluation (t=0.4):

```
➜ py evaluation.py
lecture, #concepts retrieved, recall
lecture 0: 6, 0.46153846153846156
lecture 1: 9, 0.6923076923076923
lecture 2: 1, 0.3333333333333333
lecture 3: 0, 0.0
lecture 4: 4, 0.6666666666666666
lecture 5: 10, 1.0
lecture 6: 7, 0.875
lecture 7: 3, 0.5
lecture 8: 4, 0.8
lecture 9: 5, 0.38461538461538464
lecture 10: 5, 0.45454545454545453
lecture 11: 4, 0.5714285714285714
```

Usage:

Under the ‘lda’ folder:

Using Python 3.6.8
Install Dependencies: run “pip3 install -r requirements.txt”

Transcript .vtt files to one .txt file corpus: run “python3 cleandata.py“; right now we use the .vtt files in ‘transcripts/textretrieval’ as the input, you can change the directory

name and file names in the source code to accustom to other datasets; output: a corpus stored in ‘textretrieval.txt’

Find concepts taught in each lecture: run “python3 lda.py” or “python3 lda.py <input_filename> <output_filename>”; input: ‘textretrieval.txt’ by default (basically any corpus where a line represents a document); output: results are stored in “lda_results.py” by default where concepts for each lecture are sorted in the descending order

Evaluation: run “python3 evaluation.py”; you may change the threshold in the source file; we compare “lda_result.txt” with “annotated_set.csv”

Contribution:

Bohan Liu:

1. Investigation into Gensim Phraser, Wordvec and other preprocessing.
2. Design and implementation of notebook version of ConceptPhraser, WordVecFilter and ConceptExtractor
3. Design and implementation of the python lib version of ConceptPhraser, WordVecFilter and ConceptExtractor
4. Collection and cleaning of data: cs125 transcript (cs125.dat), random Youtube video transcripts(noncs.dat, bkgd.txt)
5. Implementation of omakase testing function.

Leo Yang:

1. Implemented transcript to corpus conversion (cleandata.py).
2. Design and implementation of the LDA-based approach (lda.py) including data preprocessing (lemmatization and tokenization), n-grams (n<=4) generation, and applying the Gensim LDA model to retrieve the top topics for each lecture.
3. Design and implementation of the evaluation process, including processing the given .xlsx human annotated dataset into a .csv file (annotated_set.csv), extracting concepts of each lecture from the annotated set, and calculating the recall of each lecture (evaluation.py).
4. Exploration of word2vec-based methods (lda/wordvecs.py) (which didn’t work as well as LDA).

Yipeng Yang:

1. Designed and built topic generating model for classification with similarity functions (optim.py).

2. Worked on the optimization of similarity functions and result interpretation with the help of TF-IDF.
3. Testing of the code, processing of the evaluation data files and evaluation of final results generated.