Machine Learning Engineer Nanodegree

Capstone Project

Tyler Shumaker
November 12th, 2017

I. Definition

Project Overview

I am currently working on a project to develop a proof of concept implementation of intelligent virtual agents that will provide virtual personal assistant, recommendations, and product socializing capabilities to an user. The users will interact with the intelligent agents via chat, using an open source webchat platform as the interface to the chat bot. One of the jobs being performed in the background by the intelligent agents is to provide recommendations to the users. We intend to recommend chat rooms to users they should consider joining.

Providing recommendations based off of chat discussions has been used to provide complementary information. Recommendation of Complementary Material during Chat Discussions [1] provided recommendations of electronic documents and links to web pages to enable collective and personal learning. This proposed system performed text mining on each message posted to a chat program and compared words in the message to terms in a domain ontology. Terms identified in the message were then passed to a Recommender module where items classified with those terms were then recommended. This proposed system is similar to what we are trying to achieve but we will be recommending chat rooms that represent an area of interest.
Ex. Chat room titled "Ocean City, MD" will be used for users to chat about Ocean City, Maryland. So users chatting about terms like "Ocean City" and "Maryland" will be recommended to join the "Ocean City, MD" chat room.
* Details of the project are intentionally vague in this proposal because the work is on-going.

Problem Statement

A common problem for our clients is that users spend a lot of time finding a subject-matter expert (SME) or relevant information to assist them complete their tasks. Based upon the user's interaction with the chat bot and other users we want to be able to recommend chat rooms where other users are working on similar topics. Users are able to chat with other users in public chat rooms, private chat rooms and direct messages. No communication via private chat rooms or direct messages will be used for recommendations. Users can also interact with the chat bot via direct message and in public chat rooms by starting a message with the chat bot's handle. Ex. @chatbot

All communication between a user and the chat bot will be captured as conversations because the system is designed for the chat bot to perform actions for the user that are relevant to their tasks. Ex. @chatbot search google for "vacation destinations in Maryland" which the chat bot will respond with a list of links to the top 10 responses of a Google search of the search term. Continuous chat exchanges (separated by 30 minutes of inactivity) between multiple users in a public chat room will be recorded as a conversation. The system allows for chat rooms to have titles ("Ocean City, MD") but also associated topics of interest such as "Vacation", "Ocean", and "Maryland" to that chat room. Users also have the ability to self assign topics of interests for themselves. Users will be given recommended chat rooms at predetermined intervals (daily) or when requested from the chat bot (@chatbot get recommended chat rooms). The recommendations will be returned by the chat bot via a direct message to the user.

All of the chat communication from each chat room will be converted to vectors using a word embedding model, Doc2Vec (specifically the “distributed memory” (dm) algorithm) [3]. "Doc2vec (aka paragraph2vec, aka sentence embeddings) modifies the word2vec algorithm to unsupervised learning of continuous representations for larger blocks of text, such as sentences, paragraphs or entire documents." [4] The vectors from the all of the chat rooms will be used to train the model and the conversations of individual users will be also converted to vectors using Doc2Vec. The individual users vector will be inferred to the training corpus and return the rank of the training documents based on self-similarity. The system will return the top 20 chat rooms returned from user's conversation inferred vectors.

Metrics

This solution is returning recommendations of the highest ranked chat rooms to be recommended to users so to evaluate this solution the Mean Average Precision (MAP) will be used. MAP rewards recommendations that are the most relevant recommends. The first step is to define what is a relevant recommendation. For this project a relevant recommendation will be one (chat room) that contains at least one topic of interest that matches at least one topic of interest of the user. Ex. User A has the following topic of interests: "Java", "*Scala*" and "MongoDB". An example of a relevant chat room recommendation may have the following as associated topics of interests: "Kafka", "Akka", "*Scala*" and "Docker". The number of matching topics of interests does not change relevancy in this project.

Recommender system precision is the number of recommendations that are relevant divided by the number of items that are recommended. [10] This project will use Average Precision

(AP@N) where only the Average Precision up the the Nth item is used. For each item that is relevant the precision will be calculated and then averaged up to the Nth item. This is best explained with a table where "1" is a relevant recommendation and "0" is not. In this example N = 10.

Relevant	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**
Precision		1/2	2/3		3/5	4/6		5/8		
		0.5	0.67		0.6	0.67		.625		

AP@10 = (0.5 + 0.67 + 0.6 + 0.67 + 0.625)/5 = 0.613

In the project the top 20 recommendations will be used so N = 20. So the MAP@N is the average of the AP@N over all of the users. For this project a random set of the test users will be used to calculate the MAP. Since AP@N rewards front-loading the recommendations that are the most likely to be correct this is an appropriate evaluation metric to be used. Providing the most relevant chat room recommendations first will save users time and get them to the appropriate data more efficiently. Details on how the dataset used for this project will use MAP@N will be provided in the Data Exploration section.

II. Analysis

Data Exploration

The Cornell Movie-Dialog Corpus [2] will be used because it is a collection of conversations extracted from movie scripts. The dataset consist of 220,579 conversational exchanges between 10,292 pairs of movie characters involves 9,035 characters from 617 movies. The movie scripts were gathered from publicly available scripts that are referenced in the raw_scripts_urls file. The metadata for the movies were gathered from IMDB. The corpus consist of a movie_titles_metadata file that provides; movieID, movie title, movie year, IMDB rating, no. IMDB votes and genres. The movie_characters_metatadata, movie_lines, movie_conversations and movie_titles_metadata files can be can be used to construct the conversations between movie characters. The average length of words (after preprocessing, see Data Preprocessing section) for the movies is 4902 and the average length of words for each characters conversation is 669.

Sample of the movie_lines.txt

```
'L1045 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ They do not!',
'L1044 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ They do to!',
'L985 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ I hope so.',
'L984 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ She okay?',
"L925 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ Let's go.",
'L924 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ Wow',
"L872 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ Okay -- you're gonna need to lea
'L871 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ No',
'L870 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ I\'m kidding.  You know how som
'L869 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ Like my fear of wearing pastels
```

Sample of the movie_conversations.txt

```
"u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L194', 'L195', 'L196', 'L197']",
"u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L198', 'L199']",
"u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L200', 'L201', 'L202', 'L203']",
"u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L204', 'L205', 'L206']",
"u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L207', 'L208']",
"u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L271', 'L272', 'L273', 'L274', 'L275']",
"u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L276', 'L277']",
"u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L280', 'L281']",
"u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L363', 'L364']",
"u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L365', 'L366']"
```

Characters from the movie corpus are equivalent to users in our chat system and movies will be equivalent to chat rooms. With the movie corpus data I will be recommending movies (chat rooms) to movie characters (users). In our application the users and chat rooms can be associated with topics of interests which will be managed by users. The movie genres will be treated as topic of interests for the movies and the movie genres from the movie a character is from will be the used as the topics of interest for that character.

All of the movie lines from each movie will be converted to vectors using Doc2Vec. The movies from the Cornell Movie-Dialog Corpus will be used to train the model and the conversations of individual characters will be used to test the model. No abnormalities were identified or found with the dataset.

It should be easy determine if the model is performing correctly if a vector for a movie returns the highest rank for itself (if a character is from "movie A", then "movie A" should return a high rank). The model then be tested using the MAP@20 evaluation metric with a set of 500 randomly selected movie characters. Test showed that there was not a significant change in MAP@20 scores in test sets over 150. The AP@20 (top 20 recommendation) will be calculated for the randomly selected movies and averaged. One possible hypothesis is that a character from a movie that is categorized as a genre should recommend other movies with the same genre because of the context of the movie lines. Therefore, relevant movie recommendations will be identified by recommendations that contain at least one matching genre to the genres from the movie that test character is from.

Exploratory Visualization

The bar chart below shows the number of movies per genre in the Cornell Movie-Dialog Corpus. Most of the movies are categorized in multiple genres. If a movie has multiple genres it is included in the count for each genre.

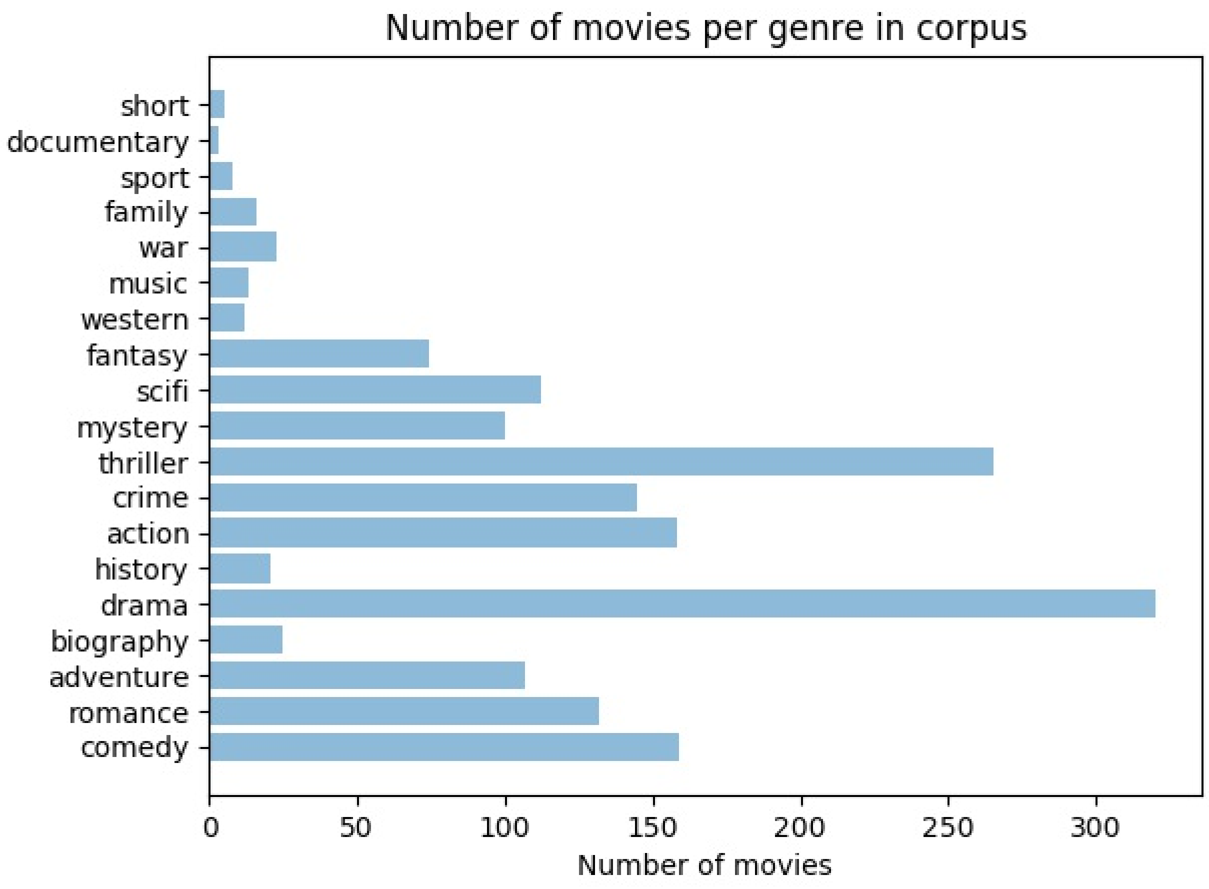


Figure 1: Number of movies per genre in the Cornell Movie-Dialogs Corpus

A genre is a movie category based on similarities in either the narrative elements or the emotional response to the film. Knowing the genres of the movies will identify relevant recommendations when calculating the MAP@20 during testing. This chart shows that there are a large number of genres represented in the corpus. Some genres are represented by a hundreds of movies like 'thriller' and 'drama'. Other genres like 'war', 'western' and 'sport' are only represented by a few movies.

Algorithms and Techniques

The conversational chat data from chat rooms and individual users results in large sets of textual data. A transcript of chat data is most comparable to a document. Doc2Vec allows for a numeric representation of these chat data documents. Doc2Vec is build on Word2Vec which is a word embedding model that represents words as a learning vector. [7] The Doc2Vec distributed memory version of the Paragraph (PV-DM) algorithm takes the Word2Vec Continuous Bag-of-Words (CBOW) algorithm which predicts words from the source context words and adds a unique document feature vector. The resulting document vector represents the concept of the document.

The Doc2Vec model is instantiated with a vector size of 30 words and iterating 20 times over the training corpus. The PV-DM training algorithm is the default algorithm used with gensim Doc2Vec. Different vector size, number of iterations and minimum word count (will give higher frequency words more weight) will be tested to see if increases in those parameters improve results while assessing the model

Doc2Vec will allow the movie lines from each movie in the Cornell Movie-Dialog Corpus, which will be used to represent a chat room's chat data, to be represented as vectors. The conversations of individual characters (users) from the corpus will also be represented as vectors and interfered into the model to calculate the similarity of the character's conversations to the lines of each movie. The movies (chat rooms) that are most similar will be recommended to the character (user) as movies (chat rooms) they could be interested in joining. Gensim calculates cosine similarity between a simple mean of the projection weight of a the given documents and the vectors for each document in the model.

Benchmark

A content-based recommendation system will be used a benchmark model. TF-IDF (Term Frequency - Inverse Document Frequency) will be used to parse of the movie lines for each movie and all of the conversations for the character which is being tested. Cosine similarity will be used to determine which conversations are closest to each other. [6] The movies that the highest cosine similarity to a characters conversation will be recommended.

The recommendation of the Doc2Vec approach will be compared to the results of the content-based recommendation system by using the MAP@20 (Mean Average Precision of the top 20 results) of 200 randomly selected characters from the dataset. The MAP should be computed against all of the characters in the dataset but test showed the MAP score leveled around 150 characters and the content-based recommendation system would not run well on my computer (it would cause it to crash) when tested with 500 random characters.

III. Methodology

Data Preprocessing

The Cornell Movie-Dialog Corpus files need to be parsed to create the text documents needed for training and testing of the model. The fields in the files are separated by " +++$+++ ". To create the document representation of all the lines from each movie the movie_lines.txt will be parsed. The movie_id field will be used as the keys in the dictionary. All the lines for each movie will be converted to list of tokens using the gensim simple pre-processing tool [8](tokenize text into individual words, remove punctuation, set to lowercase, etc) will be used as the value.

To create the document representation of each characters conversations the movie_conversations.txt will be parsed. The fields of this file are the characterID of the first character involved in the conversation, characterID of the second character involved in the conversation, movieID of the movie in which the conversation occurred and list of the utterances that make the conversation (in chronological order). To reconstruct the actual lines the utterances have to be matched to the movie_lines.txt file. A python dictionary will need to be created from the movie_lines.txt file pairing each line_id to the content. A character conversation dictionary will be create where each character_id will be used as the keys with all of the lines from that characters conversations (spoken and spoken by character conversing with) will be used as the value which will also be converted using the gensim simple pre-processing tool.

The training corpus will be created from the movie lines dictionary and the testing corpus will be created from the individual characters conversation dictionary. The corpus will be list of TaggedDocument objects where each entry from the dictionaries will be used to create a TaggedDocument with the tokenized text will be the words and the movie_id or character_id as the tag.

Dictionaries were used when parsing the movie corpus data to allow for movie and character ids to be mapped with textual content via the key - value association. The movie lines dictionary was created by looping over all the entries in the movie_lines.txt file, splitting the line on the ' +++$+++ ' field separator and ensuring the appropriate number of fields were found. An if statement to check if the movie_id field already existed as a key in the dictionary will result in the value being set to the value of the existing set plus the new content. If the movie_id was no found in as a key in the dictionary the value is set to the content of that line. A quick sanity check make sure the movie lines dictionary is was created successfully is performed by getting the length of the dictionary. The length should equal 617 since that is the number of the movies in the movie corpus.

A supporting dictionary will then be created to map line_ids to line contents. The character conversations dictionary is created similarly to how the movie lines dictionary is created. The movie_conversations.txt file is looped over to capture the conversations. There are fields for who the two characters are that are having the conversation and since all of the conversational data (spoke and spoken to in conversation) is used to for each character, each line from the conversation is added to the value for both character in the dictionary. The line id dictionary is utilized to find the content of the conversation since only the line_ids are listed in the text file. A check that character dictionary has a length of 9035 (number of unique characters in the movie corpus) is also performed.

Implementation

Doc2Vec can be trained with a list of TaggedDocument elements so the movie line and character conversation dictionaries will be used to create these lists. The ids are set as the tag and the list of tokenized text is used for the words in each TaggedDocument.

The gensim model is then instantiated and the training corpus is used to build the vocab for the model (all of the unique words extracted from the corpus with the count). The model is trained with the training corpus (list of TaggedDocuments), the model's corpus count as the total examples (because the corpus is the same as the one provided to build_vocab) and model iter as the epochs parameter.

To assess the the Doc2Vec model each movie of the training corpus will be inferred as vectors and compare the inferred vectors with the training corpus and then return the rank of the document based on self-similarity. The python list index method will be used to return the lowest index in the most_similar list that the movie_id appears. If a movie_id is the top similar document returned a "0" will be written to a ranks list. If the movie_id is another index that will be stored in ranks. A python collections.counter method will return a dictionary that will show the index results as the key and the count of that index as the value. For example the initial parameters for the Doc2Vec model return "Counter({0:614, 1:3})". This shows that out of the 617 movie documents 614 (99.5%) return itself as the most similar document. This sanity check shows that model is behaving in a usefully consistent manner. [5]

A function, get_doc2vec_similarity, returns the top 20 movies and score for the identified character using Doc2Vec most_similar function. The conversations document for the character is used as an inferred vector and the most similar movies are generated for the character. The titles of the top 20 movies in the Doc2Vec models most similar (cosine similarity) are returned by a function.

This similarity list is then used calculate the mean average precision @ k (MAP@K) using the calc_mean_average_precision_k (calc_MAP@K) function.

The calc_average_precision function uses the genres from a test characters movie to determine if each of the recommended movies are relevant or not. If one of the genres from the characters movie match a genre from a recommended movie the recommendation is considered relevant. A list the size of the recommendations list (20) is created with "1" indicating relevant recommendation and "0" representing non relevant recommendation. The average precision is then calculated using this "1"/"0" list. This returns a number between 0-1.

The calc_mean_average_precision_k (calc_MAP@K) function takes in a list of character ids as a test list and a Doc2Vec model to allow for models with different parameters to be tested. Each test character id is passed to the get_doc2vec_similarity function to retrieve the top 20 movies related to the character. The list of the top 20 movies and the character id are then passed to the calc_average_precision function. The returned average precision for each test id is then added to a list. The mean of all of the average precisions is then returned from the calc_MAP@K function. This is the main function used for evaluating the model.

To perform some visual inspections of the results of the Doc2Vec model and assist in testing a few utility functions are created. A function to get all of the metadata (characterID, character name, movieID, movie Title, gender and position in credits) for each character by characterID was created. A function to get the title of a movie by the movieID was also created. Another function returns index of the TaggedDocument created for the training and testing corpus by the movieID or characterID. This allows for the words for the movieID or characterID to be displayed for inspection.

The base function for the Doc2Vec visual inspection is the display_character_similarity function that takes the character ID and a boolean (show_words) as a parameter. The conversations document for the character is used as an inferred vector and the most similar movies are generated for the character. Some of the character metadata is then displayed (character name, movie Id the character is from and the title of that movie). If the show_words parameter is "True" then the character's conversation words will be displayed. The top 20 movies (ID and score) from the most similar list are displayed . Finally, the top five movie titles and the score of the characters movie is displayed.

A shorter version of the display_character_similarity function called get_doc2vec_similarity is used to compare the Doc2Vec results (top ten recommended movies from the most similar list) of a character to the Term Frequency - Inverse Document Frequency (TF-IDF) model for benchmarking.

The benchmark model is a content-based recommendation system that uses TF-IDF and cosine similarity to determine which movies conversations lines a character's conversations is most similar. To create this model the movie_lines.txt file from the Cornell corpus is utilized to create python dictionary similar to the on one created for the Doc2Vec model. The gensim simple pre-process tool is not used when adding the words to the dictionary. This dictionary is used to create the training corpus for the TF-IDF model as a line separated string (each line contains the movie ID and all of the text from the movie that are separated by a comma).

A function (tfidf_get_character_lines) is used to create the test strings for a given character. The functions is similar to the Doc2Vec test corpus code that creates the test corpus from the movie_conversations.txt file. This function only captures the conversation text for the given character. All spoken and spoken to lines are concatenated as the words for the character. The returned result is a single line of text that uses the character ID and the words separated by a comma. The TF-IDF model does not feature an infer option like the Doc2Vec model so the tfidf_train_corpus with the result from the tfidf_get_character_lines function for a character are used to train the model.

The 'train' function takes in a DataFrame and creates a TF-IDF matrix unigrams, bigrams and trigrams of all of the movie line documents. The 'stop_words' parameter is used to ignore common english words. The Scikit learn's linear_kernel is used to compute the cosine similarity between all of the movie documents. The similarities and scores are then stored in a List for the 100 most similar movies to each movie (entry in the DataFrame). The predict function then returns the similar items to a particular movie. Movie Id and number of similar item to return are used as parameters for predict.

A version of the calc_mean_average_precision_k that calls the Doc2Vec model and one that calls the TF-IDF 'train' and predict functions are used to compare the Doc2Vec model to the TF-IDF model based on mean average precision.

Refinement

Different parameters were tried when instantiating the Doc2Vec model and compared using the MAP@K function. The vector size, number of iterations, window size (maximum distance between the predicted word and context words used for prediction within a document) and minimum word count parameters were tried to see if the MAP@K (with K being 20) improved.

Round 1 Doc2Vec(size=50, iter=20, min_count=2) : 0.64590552578393612
Doc2Vec(alpha=0.025, min_alpha=0.025)/decrease learning rate : 0.652129159832
Doc2Vec(dm=1, dm_mean=1, size=200, window=8, min_count=19, iter=10) : 0.643467036911
Doc2Vec(dm=1, dm_mean=1, size=200, window=8, min_count=19, iter=20) : 0.641963598382
Doc2Vec(dm=1, dm_mean=1, size=300, window=8, min_count=19, iter=20) : 0.649278035546
Doc2Vec(dm=1, dm_mean=1, size=200, window=8, min_count=19, iter=100) : 0.635627557614
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=19, iter=100) : 0.664204807617
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=20, iter=100) : 0.639810561736
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=10, iter=100) : 0.641714744505
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=15, iter=100) : 0.645888061985
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=15, iter=100, alpha=0.25,

min_alpha=0.01) : 0.64739936922
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=15, iter=150) : 0.6501950632
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=15, iter=200) :
0.653454281673
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=15, iter=300) :
0.644189477281

Round 2 - new random test data
Doc2Vec(alpha=0.025, min_alpha=0.025)/decrease learning rate : 0.616014300118
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=19, iter=100) :
0.644970657008
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=15, iter=150) : 0.634117843088
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=15, iter=200) :
0.638209043472

Round 3 - new random test data
Doc2Vec(alpha=0.025, min_alpha=0.025)/decrease learning rate : 0.607888950913
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=19, iter=100) :
0.614642972259
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=15, iter=150) :
0.619167139798
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=15, iter=200) :
0.619998446705

Average
Doc2Vec(alpha=0.025, min_alpha=0.025)/decrease learning rate : (0.652129159832 +
0.616014300118 + 0.607888950913)/3 = 0.625344137
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=19, iter=100) :
(0.664204807617 + 0.644970657008 + 0.614642972259)/3 = 0.641272812
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=15, iter=150) : (0.6501950632
+ 0.634117843088 + 0.619167139798)/3 = 0.634493349
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=15, iter=200) :
(0.653454281673+ 0.638209043472 + 0.619998446705)/3 = 0.637220591

Round 4 - new random test data min_count 19 vs min_count 20
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=19, iter=100) :
0.654732378791
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=20, iter=100) :
0.649219429736

Round 5 - same random test data min_count 19 vs min_count 20
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=19, iter=100) :

0.647298604327
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=20, iter=100) :
0.647055708097

MAP run with full test corpus
Doc2Vec(dm=1, dm_mean=1, size=200, window=4, min_count=19, iter=100) : 0.64917898177

Refinement testing was completed on 200 random character ids in rounds where models with the highest MAP@20 results were tried in the next round with with new random character ids. 200 random characters were used to save on testing time because testing showed that the the results of MAP@20 were consistent after 150 test cases. The initial model that was used had a vector size of 50, number of iterations of 20 and minimum word count of 2 based of results from the ranking sanity check. This model returned 99% of the training data (movies) as the top result when used as an inferred vector.

Models with various size, window, min_count and iter parameters were tested. A model that used a decreasing learning rate was also tested. Round 1 indicated that the vector size of 200 and window value of 4 resulted in the highest MAP@20 results. The model with decreasing learning rate, model with minimum word count of 15 and 19 and iterations of 100 - 200 resulted in the best results. These models were compared to two additional random test data and averaged over the three rounds of testing. The highest average was the model that had size of 200, window of 4, min_count of 19 and iter of 100 (Doc2Vec(dm/m,d200,n5,w4,mc19,s0.001,t3)). This indicated that iterations over 100 did not create improvements in the results. Additional rounds of testing were preformed to see if increasing the min_count from 19 to 20 improved results indicated that there was no significant change in MAP@20 scores.

IV. Results

Model Evaluation and Validation

The final model, Doc2Vec(dm/m,d200,n5,w4,mc19,s0.001,t3), was then tested against the entire test set (all of the movie characters) and compared to the results for that model against 200 random character ids. The MAP@20 results were withing a few 1/100 of a percentage point. This model is return a consistent MAP@20 of appropriately 0.65 indicating that 2/3 recommendations that are being returned are relevant to the character based on genres.

The final Doc2Vec model was also tested against a modified training corpus. Movie lines were randomly ignored when creating an alternative movie lines dictionary from the movie_lines.txt file. All 617 movies from the data set were still represented by the resulting dictionary but ~10% (274412/304713) of the total movie lines were not included. The alternative training corpus was then used to train the model. The MAP@20 for 200 random character ids was calculated using the alternative data resulted in a 0.66. These results were in line with the MAP@20 results that were determined with the original training data.

The model was also tested against the testing corpus which consist of 9035 separate documents that represent the conversation data for all of the characters in the Cornell Movie data set. The testing corpus was then used to train the final Doc2Vec model. The MAP@20 for 200 random character ids was calculated using the character data resulted in a 0.60. The testing corpus is a larger data set but the individual documents were not as long as the original training set. Three out of five recommendation being relevant based on movie genres assigned to the movies the character were from is an acceptable result. The character conversation corpus also contains documents that are identical for two character that only have conversations with each other.

Over all the final model provides reliable results based off of various experiments and assessments. Additional test were preformed that compared some results of movie characters by eye as a sanity check.

The first test of the final Doc2Vec model was a test of the last movie in the training corpus to ensure that it returned itself as the most similar movie. The first movie was "Star Wars" and it returned itself as the most similar document with a similarity score of 0.972. A random movie was then selected from the training corpus and tested to see if it returned itself as the most similar. The movie "Notting Hill" was randomly selected and returned itself as the most similar document with as similarity score of 0.9851. The second most similar movie was "Bean" which returned a score of 0.3918 which clearly shows the similarity score for the second ranked document is significantly lower.

The Doc2Vec model was tested against characters that are from the same movies, multiple characters that had conversations with once other in one movie and characters that only had conversations with each other. The characters "Eddie" and "Keri" are both from the movie "Halloween H20: 20 Years Later". The conversation data for these two characters were used as inferred vectors into the movie trained final model. "Eddie" is a minor character in the movie so the size of conversation document for "Eddie" is small but the movies "Halloween" (0.75), "Friday the 13th Part III" (0.72), "A Nightmare on Elm Street 4: The Dream Master" (0.64) and "A Nightmare on Elm Street Part 2: Freddy's Revenge" (0.63) are returned in the top 5 most similar documents for "Eddie". The data for "Eddie" consist of the terms "Michael Meyers", "meat cleaver", "butcher knife" and "serial killer" so these movie recommendations seem to be appropriate. The movie which "Eddie" is from ("Halloween H20..") is the 78th ranked most similar document with a score of 0.48 which makes sense because of his minor role in the movie.

The character "Keri" is the main character from "Halloween H20: 20 Years Later" but her top 5 most similar movies are "Midnight Express" (0.72), "The Black Dahlia" (0.70), "What Women

Want" (0.60), "The Salton Sea" (0.59) and "The Getaway" (0.58). The conversation data for "Keri" does contain the terms "Micheal Myers" and "serial killer" but the character is a mother and head of a school so the context of most of her conversations contain exchanges with her son or school related. Because the size of the conversation data for "Keri" is larger the source movie, "Halloween H20:..." is the 44th ranked most similar document with a score of 0.48.

The next test looked at three characters that had conversations with each other in a movie. The test used "Han", "Luke", and "Vader" from the movie "Star Wars". "Han" had conversations with "Luke" and other characters; "Luke" had conversations with "Han", "Vader" and other characters; and "Vader" only had conversations with "Luke". The top 10 most similar movies for "Han" and "Luke" shared 5 of 10 movies. Since these two characters have a large number of conversations with each other through out the movie this is an expected result. "Vader" and a randomly chosen character, "Mrs. Fusari" from "Marty", only share 1 movie ("The Black Dahlia") with "Han" and no movies with "Luke". This is due to "Vader" having a small conversation document size.

Some of the characters that only have conversations with each other which results in the words in the documents for those characters conversations being identical were also tested. The characters "Peter" and "Con" from the movie "Get Carter" resulted in 3 of the top 5 most similar movies being the same. While the characters "Papageno" and "Ugly Old Woman" from the movie "Amadeus" resulted in 2 of the top 5 most similar movies being the same. These results are because the Doc2Vec algorithm starts by giving distinct document-IDs an initial random vector; also most training modes include some randomized steps. So even identical runs-of-words won't necessarily result in identically-trained vectors. Rather, they'll tend to become closer over training – perhaps arbitrarily close with enough passes, but never identical.

Justification

The paragraph vector (Doc2Vec) model to the TF-IDF content-based recommendation system results were compared MAP@20 results for 200 random character ids. The TF-IDF model was not tested using the entire data set because training for the TF-IDF model was required for every test character so only 200 characters were used to save testing time in this project. Because of this four separate tests of MAP@20 were run with the results averaged.

	Doc2vec	TF-IDF
Test 1	0.676496682884	0.759294826931
Test 2	0.642398098475	0.725412679863
Test 3	0.654554960505	0.725322139645
Test 4	0.67126449499	0.757527079912
Average	0.661178559	0.741889182

The Doc2Vec model resulted in an average MAP@20 of 0.66 and the TF-IDF model resulted in an average MAP@20 of 0.74. The TF-IDF model performed slightly better but the Doc2Vec model performed well enough by providing 2 of 3 recommendations as relevant based on genres.

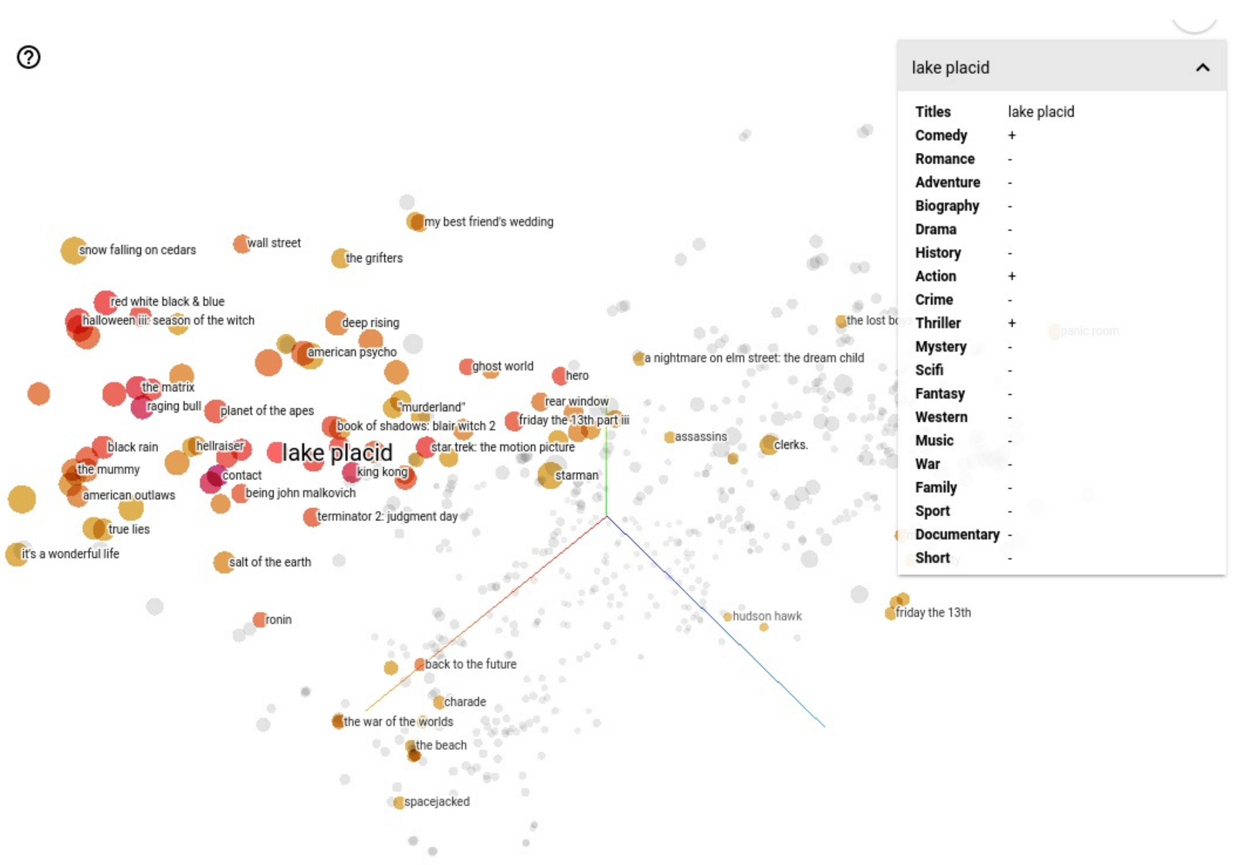
It is difficult to construct relationships between movies (documents) based on textual content. Movie lines do not read like documents that are written to relay information such as educational or informative documents. Besides using prior domain knowledge (have seen the movie) or genres it is difficult to determine if the plot of the movie deals specific domains such as technology or health care. By reading over the movie transcripts by eye this was not possible. In a real-life chat application it would be easier to determine that the software engineers chat conversations should result in similar results compared to conversations of users that are graphic artists.

V. Conclusion

Free-Form Visualization

The TensorBoard visualizer (Embedding Projector) was used to create an interactive visualization of the final Doc2Vec model. The training corpus generated from the movie_lines.txt file (list of TaggedDocument that were tagged with the movie ids) was used with the final Doc2Vec model. The model was then saved to Word2Vec format utilizing the gensim save_word2vec_format function. A Python script (word2vec2tenser.py) from the gensim library was then used to convert the saved word2vec format to tsv format. TenserBoard requires this format. The two files generated were cornell-movie_tensor.tsv (embeddeding vectors) and cornell-movie_metadata.tsv (doctags). The doctags don't contain any relevant data to be used when visualizing so the cornell-movie_metadata.tsv was modified to contain the titles of the movies and a boolean ("+"/"-") for each genre the movie was categorized. The tsv documents were then uploaded to http://projector.tensorflow.org. [9]

The TenserBoard T-SNE dimensionality reduction method was then used to visualize the embeddings. "The idea of T-SNE is to place the local neighbors close to each other, and almost completely ignoring the global structure. It is useful for exploring local neighborhoods and finding local clusters." [9] The t-sne algorithm was run using a perlexity of 50 and learning rate of 10 over 1000 iterations.



lake placid

Titles	lake placid
Comedy	+
Romance	-
Adventure	-
Biography	-
Drama	-
History	-
Action	+
Crime	-
Thriller	+
Mystery	-
Scifi	-
Fantasy	-
Western	-
Music	-
War	-
Family	-
Sport	-
Documentary	-
Short	-

Figure 2: Capture of TenserBoard Visualizer Showing Doc2Vec Model via T-SNE in 3D

In this visualization the movie 'Lake Placid' is selected and the nearest points based on cosine similarity are colored. The darker the 'shade of the red' the closer the other movies are to the selected movie. Movies like 'King Kong', 'Planet of the Apes', "Contact" and "The Matrix" are shown to be some of the closest movies to 'Lake Placid' in this space. By choosing different movies and viewing the genres of the closest movies this visualization tool shows results that make sense based off of movie knowledge which indicates the Doc2Vec model is providing adequate results.

Reflection

This project can be summarized by the following steps: 1. Identify problem and find an appropriate dataset. 2. Access data and determine preprocessing required to utilized the dataset. 3. Assess the model to determine appropriate parameters. 4. Test the model and create functions to aid in testing the model. 5. Create the benchmark model to compare recommendations. 6. Create the metadata to be used with the model in Embedding Projector (TensorBoard visualizer)

Testing the paragraph vector (Doc2Vec) model and comparing the recommendation results of the TF-IDF model used for benchmarking were the most difficult. It is difficult to view a document and mentally associate it with the numeric representation of the document. It is easier to associate the recommendations of the TF-IDF model to the documents because one can see that a word or set of words appears in two documents at a high frequency so they are similar. It is difficult to look at different movie scripts and subsets of character conversations and determine the one is talking about 'cats' and the other 'dogs' so they are similar because both are 'pets'. Using knowledge of movies and characters from movies provided to be the best solution for determining appropriate recommendations.

Through the testing performed the Doc2Vec model proved to be an reasonable approach to the finding chat rooms that are relevant to users in a chat application. The ability to train the model based off of the chat content in each chat channel can be preformed once or at spaced out intervals and finding recommendation for users is quick to accomplish by inferring the vector of that users chat conversations. This model has shown to find the similarities of the context of these conversations which can help bring users together in the system.

Improvement

By gaining access to a data from an actual chat platform a better data set for test would exist that could improve the implementation. The movie scripts with the character conversation linkage from the Cornell Movie Dialogs Corpus mapped well to the chat platform's chat rooms and user conversations but only provided fictitious dialogs and lacked easily identifiable domains. Chat data that contained user added topics of interests for users and chat rooms from the proposed system would provide better testing data. This would allow for additional refinement that could result in changes to parameters of the model.

The use of Word Mover's Distance (WMD) was looked at to find the similarities in word embedding space to improve the testing done to determine the best model to solve this problem. This would require the use of word2vec from the vocabulary of each document and is recommended to be used in short documents. The use of WMD would require a lot of computation and is generally slow.

References

[1]Loh, S., Lichtnow, D., Kampff, A. J., & Moreira de Oliveira, J. P. (2010). Recommendation of Complementary Material during Chat Discussions. Knowledge Management & E - Learning: An International Journal, 2(4), 385-399. Retrieved July 11, 2017, from http://www.kmel-journal.org/ojs/index.php/online-publication/article/viewFile/20/63

[2]Danescu-Niculescu-Mizil, C. (2011). Cornell Movie--Dialogs Corpus. Retrieved July 6, 2017, from http://www.cs.cornell.edu/~cristian//Cornell_Movie-Dialogs_Corpus.html

[3]Quoc V. Le, and Tomas Mikolov, Distributed Representations of Sentences and Documents ICML, 2014 from https://cs.stanford.edu/~quocle/paragraph_vector.pdf

[4]Rehurek, R. (2014) Doc2vec tutorial. Retrieved July 18, 2017, from https://rare-technologies.com/doc2vec-tutorial/

[5]RaRe-Technologies (2017, June 10), Doc2Vec Tutorial on the Lee Dataset. Retrieved July 18, 2017, from https://github.com/RaRe-Technologies/gensim/blob/develop/docs/notebooks/doc2vec-lee.ipynb

[6]Clark, C. (2016, June 09). A Simple Content-Based Recommendation Engine in Python. Retrieved July 6, 2017, from http://blog.untrod.com/2016/06/simple-similar-products-recommendation-engine-in-python.html

[7]TenserFlow (2017, August 17), Vector Representations of Words. Retrieved September 1, 2017 from https://www.tensorflow.org/tutorials/word2vec

[8]Rehurek, R., & Sojka, P. (2010). Software framework for topic modelling with large corpora. THE LREC 2010 WORKSHOP ON NEW CHALLENGES FOR NLP FRAMEWORKS, 45-50. Retrieved July 17, 2017, from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.695.4595

[9]RaRe-Technologies (2017, June 10), TensorBoard Visualization. Retrieved September 18, 2017, from https://github.com/RaRe-Technologies/gensim/blob/develop/docs/notebooks/Tensorboard_visualizations.ipynb

[10]Sawtelle, Sonya (2016, Oct 25). Mean Average Precision (MAP) For Recommender Systems. Retrieved October 27, 2017. from http://sdsawtelle.github.io/blog/output/mean-average-precision-MAP-for-recommender-systems.html