Sangita Gupta

Project 4: Semantic Search

Objective:

The objective of this assignment is to engineer a novel wikipedia search engine using what you've learned about data collection, infrastructure, and natural language processing.

The task has two required sections:

1. Data collection
2. Search algorithm development

Part 1 - Data Collection:

I query the wikipedia API and try to collect all of the articles under the following wikipedia categories and their subcategories:

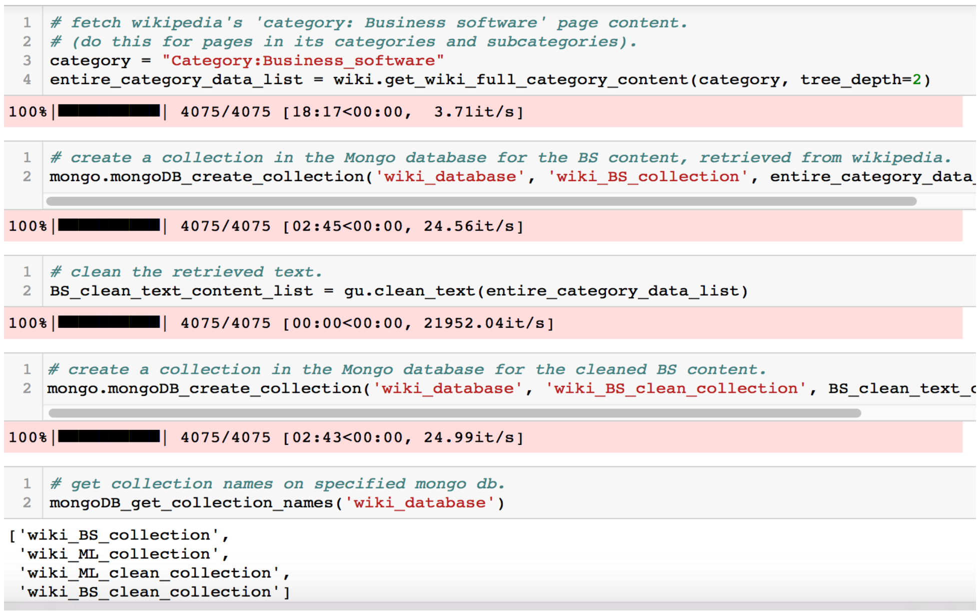
- Machine Learning
- Business Software

Both these categories contain a hierarchy of nested sub-categories.
For ***Machine Learning***, I was able to pull all the pages, which total 1620 pages.
For ***Business Software***, the recursive process for acquiring all pages of all subcategories, seemed to spin off into an infinite loop. To avoid this, I chose to limit the recursive subcategory search to a depth of two levels. I pulled 4075 pages total for Business Software.

Storage:

The raw page text, its category and page id, were stored in a dictionary. Each page was represented by a dictionary and each category was represented by a list of dictionaries. The information was then written to a 'raw data' collection on a Mongo server, running on a dedicated AWS instance. I retrieved this 'raw data' collection, cleaned its text to remove html and newline tags, and stored it in a 'clean data' collection on the Mongo server.

Reference: jupyter notebook "data_acquisition_NB1"



```
# fetch wikipedia's 'category: Business software' page content.
# (do this for pages in its categories and subcategories).
category = "Category:Business_software"
entire_category_data_list = wiki.get_wiki_full_category_content(category, tree_depth=2)
```

```
100%|██████████| 4075/4075 [18:17<00:00,  3.71it/s]
```

```
# create a collection in the Mongo database for the BS content, retrieved from wikipedia.
mongo.mongoDB_create_collection('wiki_database', 'wiki_BS_collection', entire_category_data_
```

```
100%|██████████| 4075/4075 [02:45<00:00, 24.56it/s]
```

```
# clean the retrieved text.
BS_clean_text_content_list = gu.clean_text(entire_category_data_list)
```

```
100%|██████████| 4075/4075 [00:00<00:00, 21952.04it/s]
```

```
# create a collection in the Mongo database for the cleaned BS content.
mongo.mongoDB_create_collection('wiki_database', 'wiki_BS_clean_collection', BS_clean_text_c
```

```
100%|██████████| 4075/4075 [02:43<00:00, 24.99it/s]
```

```
# get collection names on specified mongo db.
mongoDB_get_collection_names('wiki_database')
```

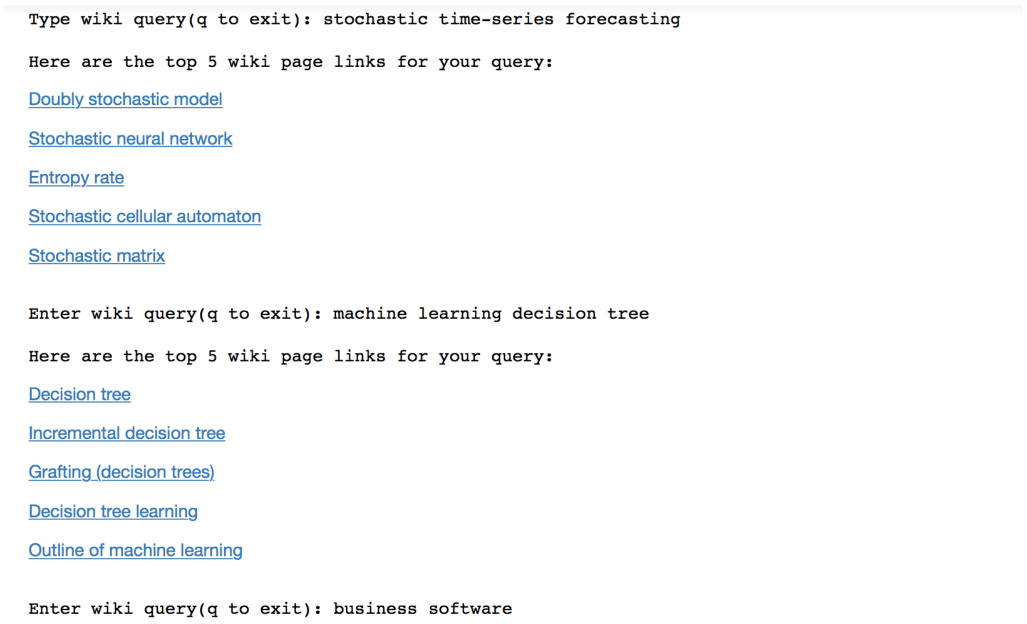
```
['wiki_BS_collection',
 'wiki_ML_collection',
 'wiki_ML_clean_collection',
 'wiki_BS_clean_collection']
```

Part 2 – Search:

Developed a search engine, using Latent Semantic Analysis, to match a "user query" with its most similar wiki articles. I developed an interactive engine, which takes a query from the user and returns a list with its top 5 wiki article hyperlinks.

I used singular vector decomposition (SVD) and cosine similarity, to find the most similar wiki articles.

Reference: jupyter notebooks "semantic_analysis_NB2" and "final_query_app_NB3"



```
Type wiki query(q to exit): stochastic time-series forecasting

Here are the top 5 wiki page links for your query:
```

Doubly stochastic model

Stochastic neural network

Entropy rate

Stochastic cellular automaton

Stochastic matrix

```
Enter wiki query(q to exit): machine learning decision tree

Here are the top 5 wiki page links for your query:
```

Decision tree

Incremental decision tree

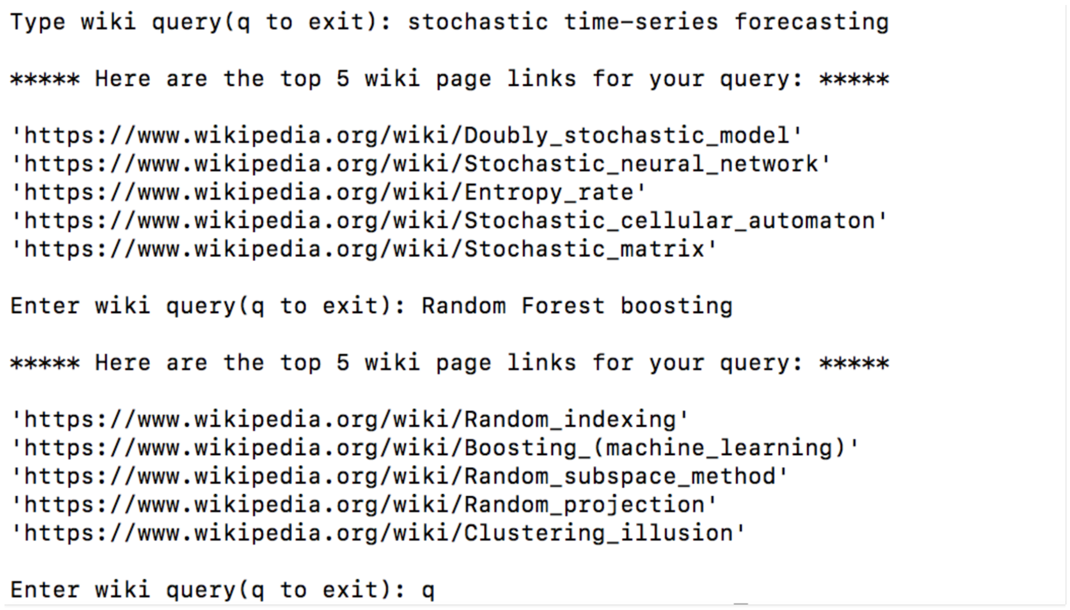
Grafting (decision trees)

Decision tree learning

Outline of machine learning

```
Enter wiki query(q to exit): business software
```

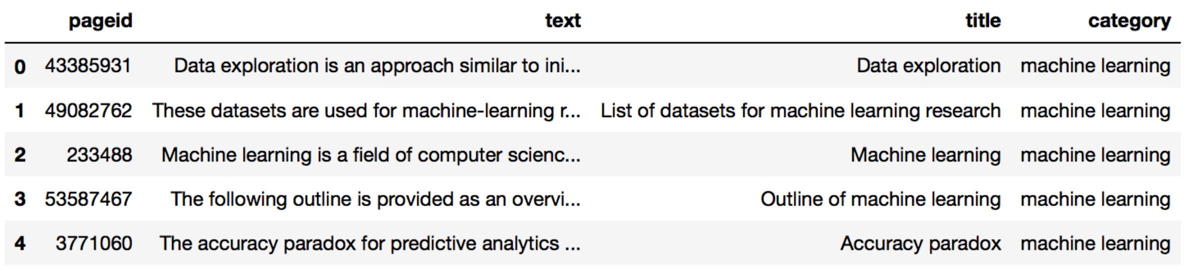
Optional: I made the query app also run via a python script.



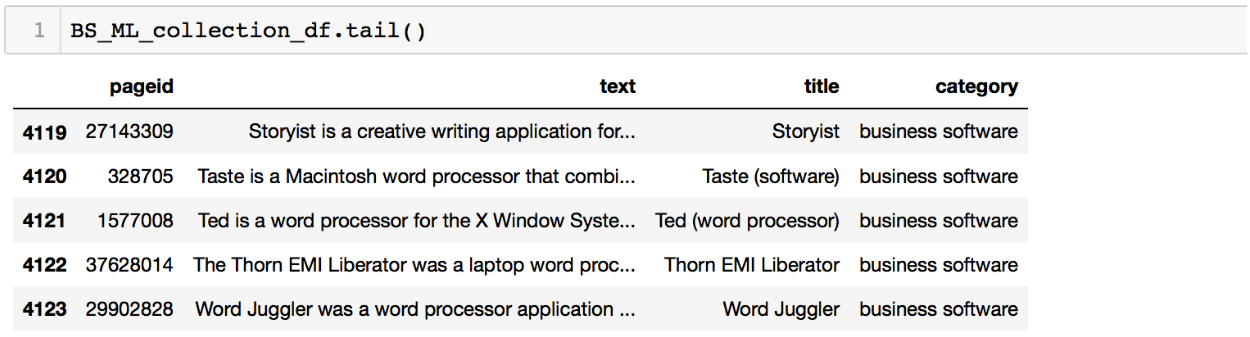
```
Type wiki query(q to exit): stochastic time-series forecasting

***** Here are the top 5 wiki page links for your query: *****

'https://www.wikipedia.org/wiki/Doubly_stochastic_model'
'https://www.wikipedia.org/wiki/Stochastic_neural_network'
'https://www.wikipedia.org/wiki/Entropy_rate'
'https://www.wikipedia.org/wiki/Stochastic_cellular_automaton'
'https://www.wikipedia.org/wiki/Stochastic_matrix'

Enter wiki query(q to exit): Random Forest boosting

***** Here are the top 5 wiki page links for your query: *****

'https://www.wikipedia.org/wiki/Random_indexing'
'https://www.wikipedia.org/wiki/Boosting_(machine_learning)'
'https://www.wikipedia.org/wiki/Random_subspace_method'
'https://www.wikipedia.org/wiki/Random_projection'
'https://www.wikipedia.org/wiki/Clustering_illusion'

Enter wiki query(q to exit): q
```

Part 3 -- Predictive Model (optional)

Built a predictive model from the wikipedia text data collected. Below is a sample of the data.



	pageid	text	title	category
0	43385931	Data exploration is an approach similar to ini...	Data exploration	machine learning
1	49082762	These datasets are used for machine-learning r...	List of datasets for machine learning research	machine learning
2	233488	Machine learning is a field of computer scienc...	Machine learning	machine learning
3	53587467	The following outline is provided as an overvi...	Outline of machine learning	machine learning
4	3771060	The accuracy paradox for predictive analytics ...	Accuracy paradox	machine learning



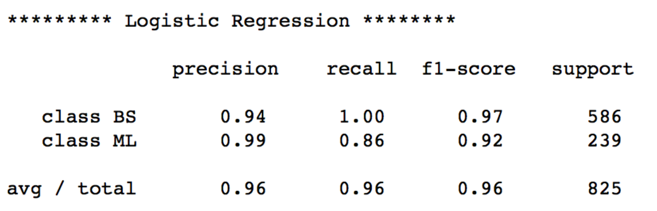
```
BS_ML_collection_df.tail()
```

	pageid	text	title	category
4119	27143309	Storyist is a creative writing application for...	Storyist	business software
4120	328705	Taste is a Macintosh word processor that combi...	Taste (software)	business software
4121	1577008	Ted is a word processor for the X Window Syste...	Ted (word processor)	business software
4122	37628014	The Thorn EMI Liberator was a laptop word proc...	Thorn EMI Liberator	business software
4123	29902828	Word Juggler was a word processor application ...	Word Juggler	business software

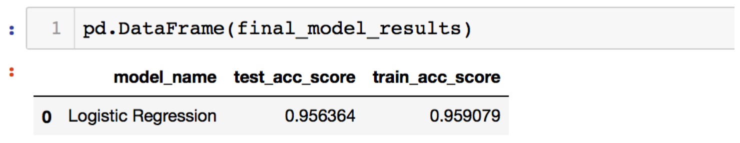
I used the Logistic Regression model to build a binary predictor, since we want to predict between the "machine learning" and "business software" categories.

Process:

- I performed latent semantic analysis on the wikipedia page text corpus collected.
    - To do LSA, the corpus was tfidf 'fit and transformed' and then SVD 'fit and transformed'.
- The svd_matrix along with the 'hot one encoded' category labels, were used to train the logistic regression model.



```
********* Logistic Regression ********

             precision    recall  f1-score   support

   class BS       0.94      1.00      0.97       586
   class ML       0.99      0.86      0.92       239

avg / total       0.96      0.96      0.96       825
```



```
pd.DataFrame(final_model_results)
```

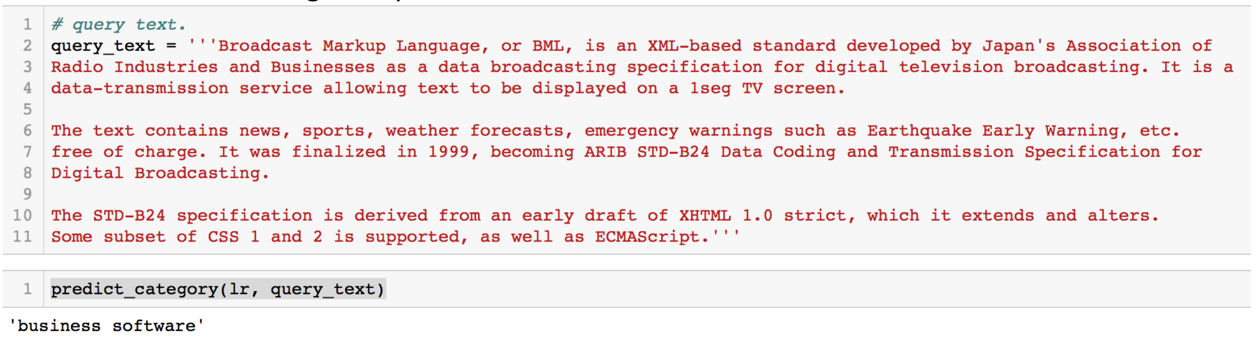
	model_name	test_acc_score	train_acc_score
0	Logistic Regression	0.956364	0.959079

From the precision, recall and accuracy metrics, we see that the logistic regression model performed well in predicting the wiki category from the input text. It achieves around a 96% accuracy rate.

When a new article from wikipedia comes along, we would like to be able to predict what category the article should fall into.

- To test this out, I randomly copied and pasted some text from very embedded subcategory pages for the Business and Machine Learning categories, on wikipedia.

Here are some following test queries I did and their results.



```
# query text.
query_text = '''Broadcast Markup Language, or BML, is an XML-based standard developed by Japan's Association of
Radio Industries and Businesses as a data broadcasting specification for digital television broadcasting. It is a
data-transmission service allowing text to be displayed on a 1seg TV screen.

The text contains news, sports, weather forecasts, emergency warnings such as Earthquake Early Warning, etc.
free of charge. It was finalized in 1999, becoming ARIB STD-B24 Data Coding and Transmission Specification for
Digital Broadcasting.

The STD-B24 specification is derived from an early draft of XHTML 1.0 strict, which it extends and alters.
Some subset of CSS 1 and 2 is supported, as well as ECMAScript.'''
```

```
predict_category(lr, query_text)
```

```
'business software'
```

```
query_text='''Cuneiform is an open-source workflow language for large-scale scientific data analysis.[1][2] It is
a workflow DSL in the form of a functional programming language promoting parallelizable algorithmic skeletons.
External tools and libraries, in, e.g., R or Python, can be integrated via a foreign function interface.
Cuneiform's data-driven evaluation model and integration of external software originate in scientific workflow
languages like Taverna, KNIME, or Galaxy while its algorithmic skeletons (second-order functions) for parallel
execution originate in data-parallel programming models like MapReduce or Pig Latin. Cuneiform is implemented in
Erlang, and therefore must run on an Erlang Virtual Machine (BEAM) similar to the way Java must run on a JVM
(Java Virtual Machine). Cuneiform scripts can be executed on top of Hadoop.[3][4][5][6][7]'''
```

```
predict_category(lr, query_text)
```

```
'business software'
```

```
# query text.
query_text ='''Latent growth modeling is a statistical technique used in the structural equation modeling (SEM)
framework to estimate growth trajectory. It is a longitudinal analysis technique to estimate growth over a period
of time. It is widely used in the field of behavioral science, education and social science. It is also called
latent growth curve analysis. The latent growth model was derived from theories of SEM. General purpose SEM
software, such as OpenMx, lavaan (both open source packages based in R), AMOS, Mplus, LISREL, or EQS among others
may be used to estimate the trajectory of growth.

Latent Growth Models [1] [2] [3] [4] represent repeated measures of dependent variables as a function of time and
other measures. Such longitudinal data share the features that the same subjects are observed repeatedly over time,
and on the same tests (or parallel versions), and at known times. In latent growth modeling, the relative standing
of an individual at each time is modeled as a function of an underlying growth process, with the best parameter
values for that growth process being fitted to each individual.'''
```

```
predict_category(lr, query_text)
```

```
'machine learning'
```

I repeatedly tested this many times. The model seems to miss predict, when a small amount of text is entered, with very neutral words. Otherwise as we can see above, the model predicts well on embedded subcategory text I randomly got from wikipedia.