CS 410 (Fall-2020) Final Project – Expert Search

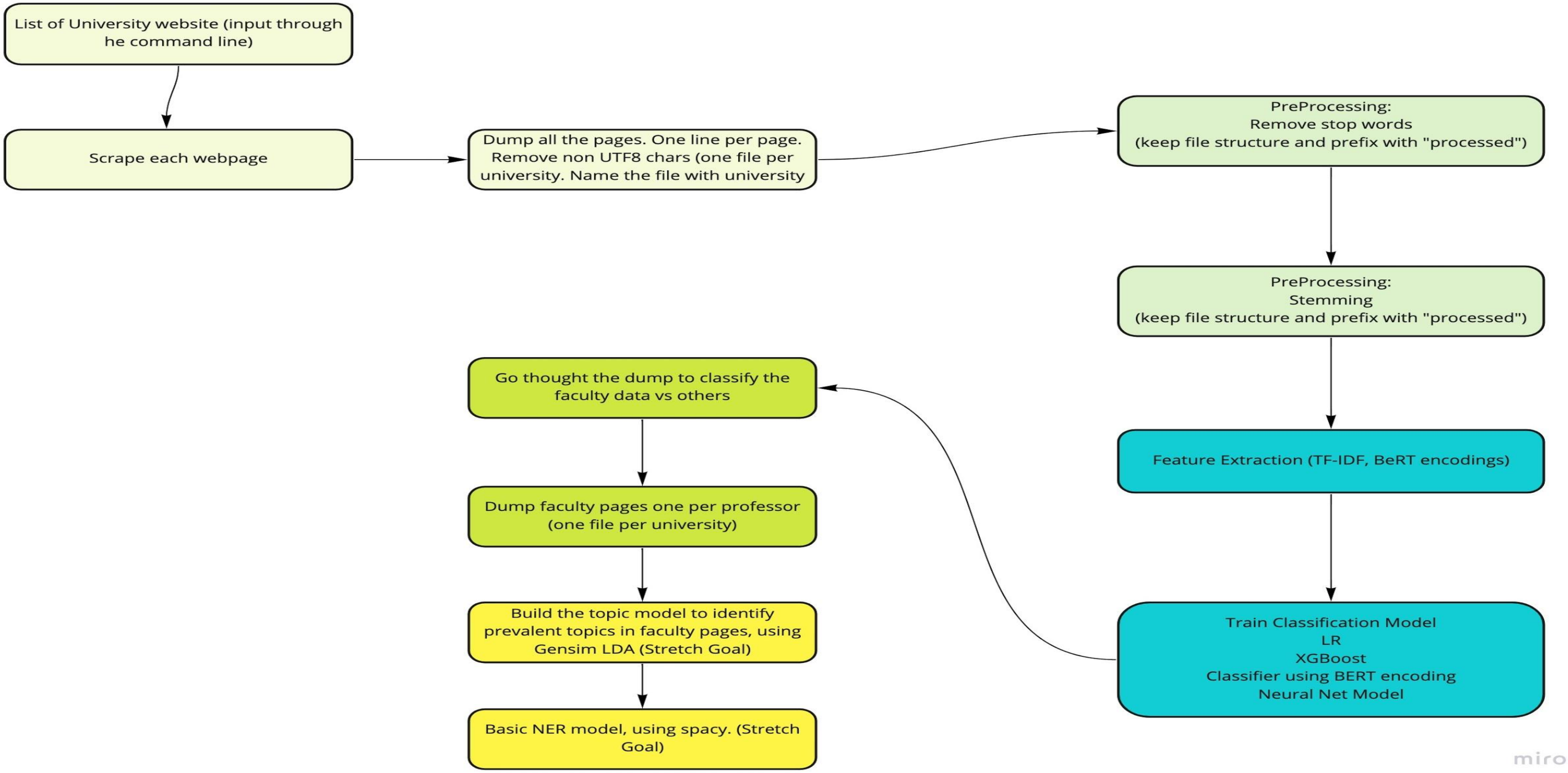
Team – Bay2Bay

Pushpit Saxena (pushpit2)

Govindan Menon (gvmenon2)

Harikrishna Bojja (hbojja2)

Expert Search – Process Flow



Step-1: Web Page - Crawl & Extract

Steps	Script	Input	Output
1	Scrapy (Extract Links)	University Web-Page	Extract all Web-Links which University Web Page links-to
2	Split	Output from Step-1	Create multiple files to achieve parallelism in Step-3
3	Beautiful Soap (Extract Text)	Web-Links from Step-2	Extract all text of Web-Page in a txt file (Each line correspond to one Web-Page)
4	Merge	Output Files from Step-3	Merge file to one txt file.

Notes –

1. Scrapy Framework –
   - Utilized scrapy framework with Python to crawl web pages and identify connected links.
2. Beautiful Soup –
   - Utilized BeautifulSoup toolkit with Python to extract text from web pages.
   - Removed special characters using regular expressions and extracted text data from web pages.

Classification: Faculty Web page classification

- Train different classifier to classify web pages as Faculty Page (positive: 1) or Non Faculty pages (negative: 0)
- The goal is to automate the indexing for expert search by classifying crawled web pages as faculty pages.
- We have trained different classification models as part of this exercise:
  - Logistic Regression
  - XGBoost
  - Neural Network Model
  - (Experimental)Logistic Regression/SVM classifier based on BeRT encodings for web pages

Step-2.1: Logistic Regression

Steps	Script	Input	Output
1) Training	**train.py** under “source_code.logistic_regression”	Training data ● stop words removed ● stemming performed ● features built using **tfidf vectorizer** ● same as the one we used in NN model later	Trained model ● trained model saved under the directory **logistic_regression** with the name **logit.model**
2) Inference	**inference.py** under “source_code.logistic_regression”	Carawled data file ● source link and doc text written on per line ● Use the separator “#####” between link and doc text	Faculty links classified ● data is read, preprocessed, vectorized ● then classified links are printed one per line

Notes: We have used Scikit learn Logistic regression module as well as scikit learn tf-idf vectorizer for vectorizing the web pages. We have used nltk library to remove stopwords and for stemming.

F1-Score (hold out test set): 0.9702970297029703

Step-2.2: XGboost

Steps	Script	Input	Output
1) Training	**train.py** under "source_code.XGboost"	Training data • stop words removed • stemming performed • features built using **tfidf vectorizer** • same as the one we used in NN model later	Trained model • trained model saved under the directory **XGboost** with the name **xgb.model**
2) Inference	**inference.py** under "source_code.XGboost"	Carawled data file • source link and doc text written on per line • Use the separator "#####" between link and doc text	Faculty links classified • data is read, preprocessed, vectorized • then classified links are printed one per line
3) Hyper parameter tuning	**train_grid_search.py** under "source_code.XGboost"	Training data • stop words removed • stemming performed • features built using **tfidf vectorizer** • different hyperparameter values are defined and then scikit learn GridSearchCV is used to find the optimal parameters	Optimal Hyperparameters • set of optimal hyperparameters can be seen on the console after the hyperparameter grid search

Note: We have used XGBoost library here.

F1-Score (hold out test set): 0.9829683698296837

Step-2.3: Neural Network Model

Steps	Script	Input	Output
1) Training	**model_driver.py** under “source_code.neural_network_ml_classifier.train”	Training data ● stop words removed ● stemming performed ● features built using **tfidf vectorizer**	Trained model ● trained model saved under the directory **target_path + ‘/model’** ● vectorizer saved as **target_path + ‘/tfidf’**
2) Inference	**infer_crawled_data.py** under “source_code.neural_network_ml_classifier.classify”	Carawled data file ● source link and doc text written on per line ● Use the separator “#####” between link and doc text	Faculty links classified ● data is read, preprocessed, vectorized ● then classified links are printed one per line

Notes –

1. Tensorflow –
   - Used tensorflow to build the deep-learning model.
     - Four layered **neural network model**(excluding input layer)
     - Uses Adam optimizer with learning **0.0001**
     - loss is evaluated using **Binary Cross Entropy**
     - F1 Score on test data set: **0.9963**
2. NLTK –
   - Utilized NLTK to pre-process the data. Pre-processing involved following list of steps
     - Remove stop words, Stemming

Step-2.4: Classifier using pre-trained BERT

(Experimental)

Steps	Script	Input	Output
1) Bert Encoding Generation	**bert_model.ipynb** under "BERT_encoding_classifier"	Web Pages text • Both Faculty and Non faculty pages	• BERT encoding generated using a pre-trained BERT model • **"distilbert-base-nli-mean-tokens"** • Encoding saved at: **"BERT_encoding_classifier.bert-embeddings-for-classification.pkl"**
2) Training	**bert_model.ipynb** under "BERT_encoding_classifier"	• BERT encodings generated in step 1	Trained model • trained model saved under the directory "**BERT_encoding_classifier**" as '**logit.model**'

Note: We are using transformers library (https://pypi.org/project/sentence-transformers/)

F1-Score : 0.9874055415617129

Step-3: Web Page classification

- Common classifier can be found at the location: source_code/faculty_page_classifier/faculty_page_classifier.
- This is the entry point for the classification module
- Users can initialize this classifier object with appropriate type and run classification tasks on wep pages. Only requirement is that each webpage is represented as a single str, i.e. the predict method can take a list of webpages each represented as single str and can run prediction using the selected method.
- Methods available:
  - logit (default) → Logistic Regression
  - xgb → XGBoost
  - nn → Custom Neural Network Model

Step-4: Topic Modeling on Compiled bios

- Topic modeling
  - Using Gensim Library
  - Script location: source_code/topic_modeller/TopicModelling.ipynb

Topics identified by Gensim LDA (k = 10 topics) on compiled bios:

[(0, '0.019*"graphics" + 0.018*"paper" + 0.015*"image" + 0.014*"siggraph" + 0.010*"computer"'),

(1, '0.033*"conference" + 0.027*"international" + 0.018*"systems" + 0.014*"proceedings" + 0.013*"networks"'),

(2, '0.015*"translation" + 0.012*"speech" + 0.010*"blandford" + 0.010*"rogers" + 0.009*"nadia"'),

(3, '0.022*"research" + 0.019*"function" + 0.019*"study" + 0.017*"details" + 0.014*"state"'),

(4, '0.014*"programming" + 0.013*"system" + 0.011*"architecture" + 0.011*"software" + 0.010*"memory"'),

(5, '0.017*"electrical" + 0.014*"engineering" + 0.012*"power" + 0.009*"control" + 0.009*"signal"'),

(6, '0.015*"learning" + 0.008*"conference" + 0.008*"machine" + 0.007*"model" + 0.007*"paper"'),

(7, '0.030*"research" + 0.026*"university" + 0.025*"computer" + 0.024*"science" + 0.018*"engineering"'),

(8, '0.016*"theory" + 0.012*"algorithms" + 0.011*"algorithm" + 0.009*"graph" + 0.008*"complexity"'),

(9, '0.008*"guohong" + 0.008*"ghosh" + 0.008*"patrick" + 0.008*"veeravalli" + 0.008*"thomas"')]

Step-5: NER model

- Built a NER model using spacy library (https://spacy.io/)
- Script: source_code/ner_model/spacy_ner.ipynb
- Also, extracted faculty names as well as different organizations mentioned on the faculty pages. These two data points will help in indexing the faculty bios for searching.
- Names are extracted and saved in source_code/data/compiled_bios_names/
- Organizations are extracted and saved in source_code/data/compiled_bios_orgs
- We have used the same filenames as faculty bios for easier retrieval when this information is needed later

References:

1. Scrapy - https://docs.scrapy.org/en/latest/
2. Beautiful Soup - https://pypi.org/project/beautifulsoup4/
3. Beautiful Soup - https://www.kite.com/python/docs/bs4.BeautifulSoup
4. Scikit Learn - https://scikit-learn.org
   - Logistic Regression - https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
   - TFIDf vectorizer- https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
5. Gensim - https://pypi.org/project/gensim/
6. XGBoost - https://xgboost.readthedocs.io/en/latest/python/python_api.html
7. Spacy - https://spacy.io/ , https://spacy.io/models/en#en_core_web_md
8. Tensorflow - https://www.tensorflow.org/api_docs/python/tf/all_symbols

Challenges Faced:

1. Extracting and crawling web pages using lower powered CPUs and less memory on personal machines is posing a challenge from performance and scalability perspective.
2. Extracting/ crawling web pages of different universities using a scrapy framework requires understanding of the page structure. Page structure could be different for different web pages and this is a challenge for crawling and scraping required contents.
3. While we were able to collect positive training set for classifiers, collecting "quality" negative data set could be tricky
   - Collected positive training set from CS410-MP2
   - Trying to use general web crawled data for negative examples.

Future Enhancements

- Integration with the end to end expert search system
  - As part of this project we undertook some tasks to improve the portions of the expert search system
    - We came up with a robust web scraping module
    - Built an ensemble of classifier to classify web pages as faculty vs non faculty
    - Built a topic model on faculty pages
    - Built a named entity recognition module
  - So, the next step is to integrate these enhanced modules to generate appropriate search index (outside the scope of our project for now).
  - Please note: that entry points for each of these are already developed.
- Improve the classification models:
  - If we get more time, we can enhance the classification model and improve the accuracy further. Currently we only consider the text on the pages to generate features. We can augment these features with additional features like web page link structure, web page citation count etc.