DEVELOPMENT OF FORECASTING MODEL TO IDENTIFY EMERGING BUSINESS TOPIC VIA TEXT MINING APPROACH USING WEB-CRAWLED PATENT DATA

AeLee Im

aelee.im@gmail.com

GOAL OF THIS PROJECT

- Developing the prototype of program to process and analyze the data
    - To forecast future emerging business topics within specific industry
    - Via designing, developing & implementation of algorithm, analysis framework and process

SCOPE OF THIS PROJECT

- Range of interesting industry for web-crawling of input data
    - Digitalization, Digital Asset Management, Automation of documentation or working process
- Scope of methodology
    - Data acquisition by web-crawling
    - Text analysis & Clustering by Topic Modeling
    - Network & Citation analysis to develop evaluation principle

PROJECT AT A GLACE

Key Phrase identification

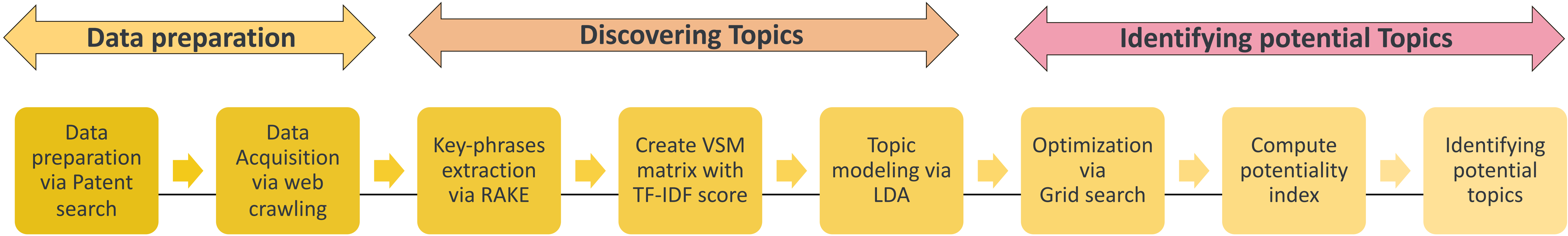
- Patent Search(for setting the analysis range) via web crawling
- Data cleasing due to the duplication issue from ‚Patent Family‘
- Key-phrases extraction : via RAKE(Rapid Automatic Keyword Extraction) approach
- Patent-keyword matrix with the frequency after TF-IDF conversion

Identifying innovative topic

- Topic mdeling via LDA(latent dirichlet allocation) approach
- Topic – keywords distribution matrix
- Patent – topic distribution matrix -> adjusting with threshold via VSM approach & labeling

Identifying potentiality

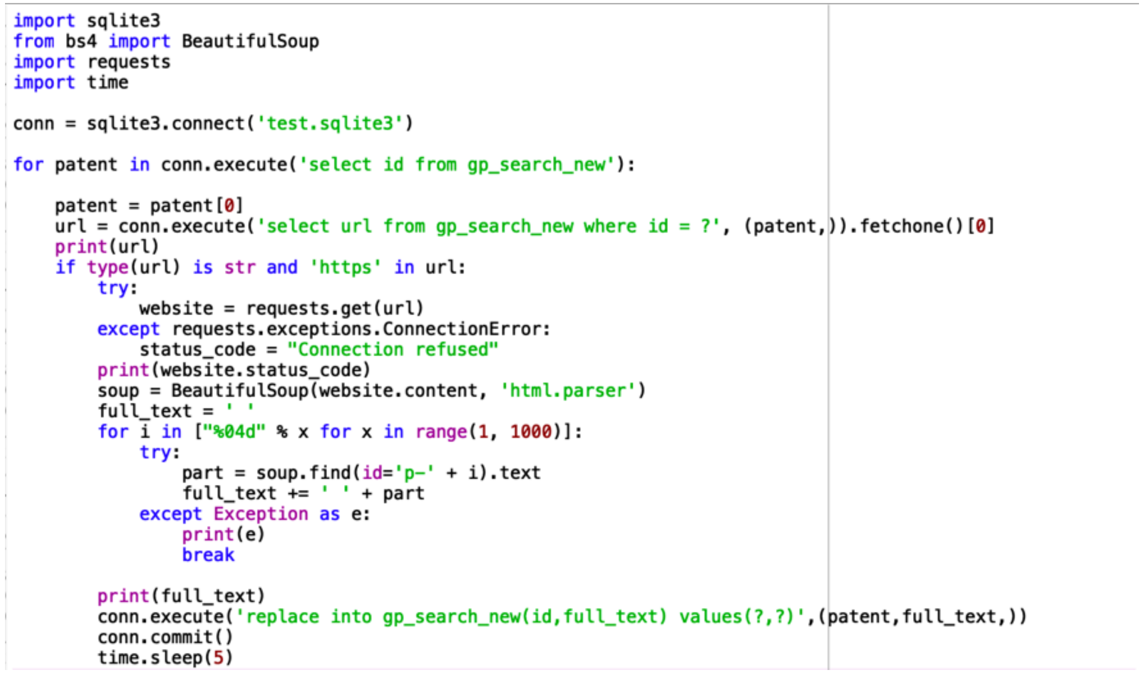
- Compute potentiality and verify emerging topics
- via OFIT(Opportunity-focused innovation topic) map
- Visualization and writing report



Data preparation
Discovering Topics
Identifying potential Topics
Data preparation via Patent search
Data Acquisition via web crawling
Key-phrases extraction via RAKE
Create VSM matrix with TF-IDF score
Topic modeling via LDA
Optimization via Grid search
Compute potentiality index
Identifying potential topics

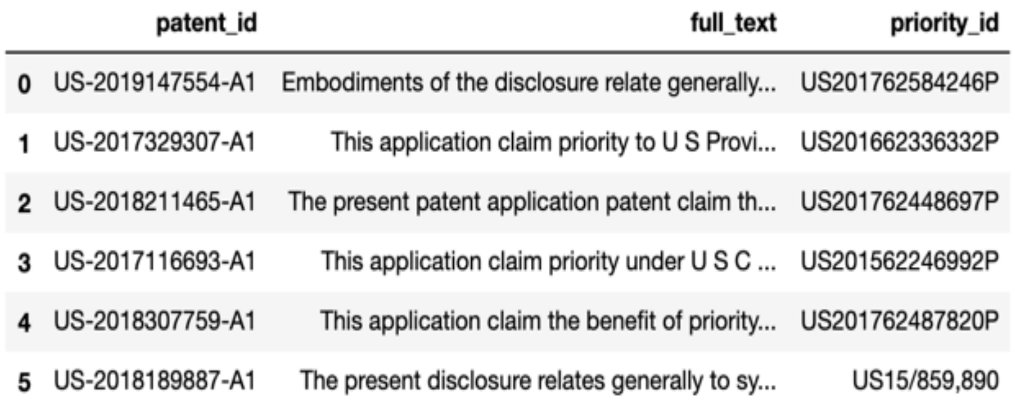
DATA ACQUISITION BY WEB CRAWLING

- Python Script for web crawling



```
import sqlite3
from bs4 import BeautifulSoup
import requests
import time

conn = sqlite3.connect('test.sqlite3')

for patent in conn.execute('select id from gp_search_new'):

    patent = patent[0]
    url = conn.execute('select url from gp_search_new where id = ?', (patent,)).fetchone()[0]
    print(url)
    if type(url) is str and 'https' in url:
        try:
            website = requests.get(url)
        except requests.exceptions.ConnectionError:
            status_code = "Connection refused"
        print(website.status_code)
        soup = BeautifulSoup(website.content, 'html.parser')
        full_text = ' '
        for i in ["%04d" % x for x in range(1, 1000)]:
            try:
                part = soup.find(id='p-' + i).text
                full_text += ' ' + part
            except Exception as e:
                print(e)
                break

        print(full_text)
        conn.execute('replace into gp_search_new(id,full_text) values(?,?)',(patent,full_text,))
        conn.commit()
        time.sleep(5)
```

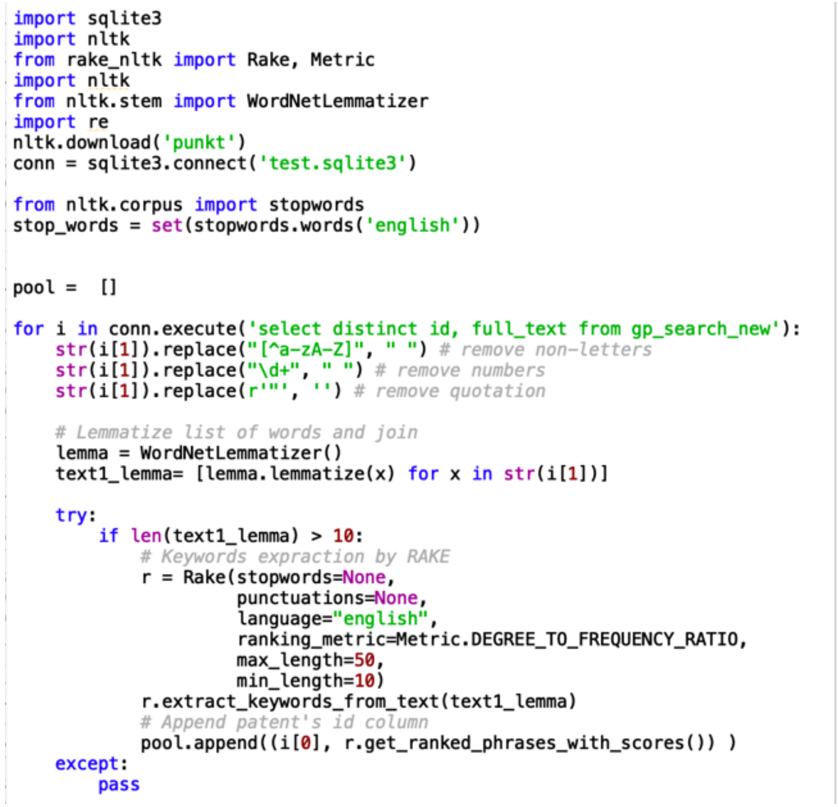
- Web-crawled input data for this project
    - From Google Patents



	patent_id	full_text	priority_id
0	US-2019147554-A1	Embodiments of the disclosure relate generally...	US201762584246P
1	US-2017329307-A1	This application claim priority to U S Provi...	US201662336332P
2	US-2018211465-A1	The present patent application patent claim th...	US201762448697P
3	US-2017116693-A1	This application claim priority under U S C ...	US201562246992P
4	US-2018307759-A1	This application claim the benefit of priority...	US201762487820P
5	US-2018189887-A1	The present disclosure relates generally to sy...	US15/859,890

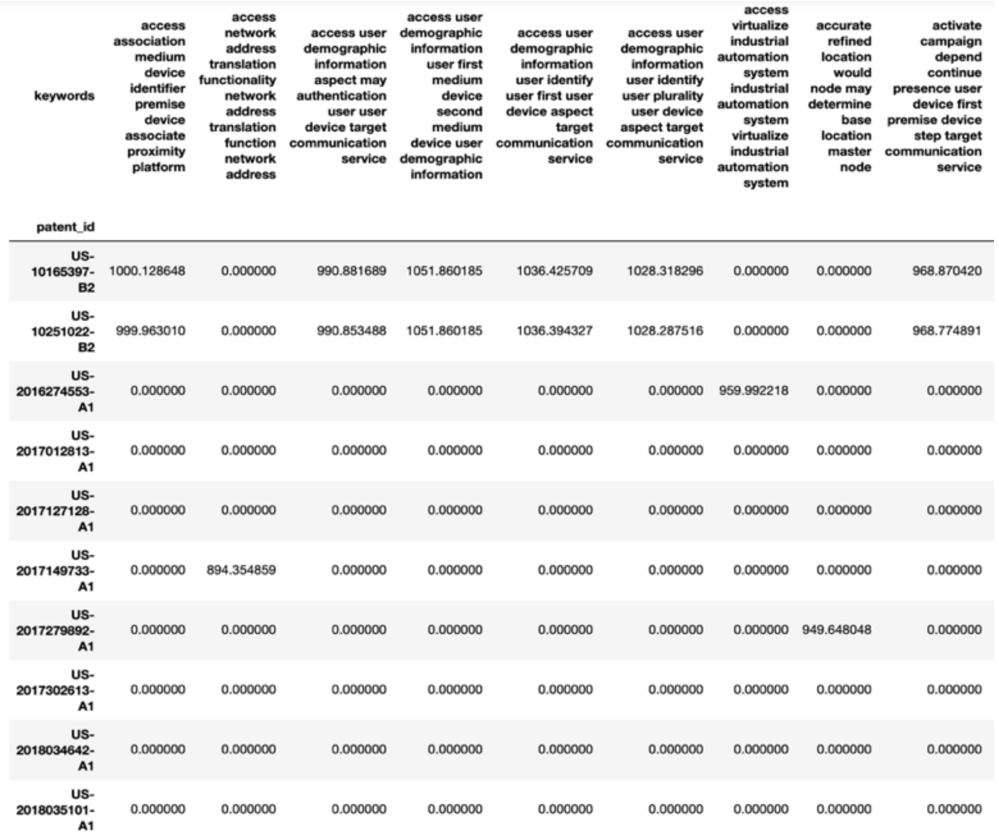
RAKE KEYWORDS EXTRACTION

- Python Script for RAKE keywords extraction



```
import sqlite3
import nltk
from rake_nltk import Rake, Metric
import nltk
from nltk.stem import WordNetLemmatizer
import re
nltk.download('punkt')
conn = sqlite3.connect('test.sqlite3')

from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))


pool =  []

for i in conn.execute('select distinct id, full_text from gp_search_new'):
    str(i[1]).replace("[^a-zA-Z]", " ") # remove non-letters
    str(i[1]).replace("\d+", " ") # remove numbers
    str(i[1]).replace(r'"', '') # remove quotation

    # Lemmatize list of words and join
    lemma = WordNetLemmatizer()
    text1_lemma= [lemma.lemmatize(x) for x in str(i[1])]

    try:
        if len(text1_lemma) > 10:
            # Keywords expraction by RAKE
            r = Rake(stopwords=None,
                     punctuations=None,
                     language="english",
                     ranking_metric=Metric.DEGREE_TO_FREQUENCY_RATIO,
                     max_length=50,
                     min_length=10)
            r.extract_keywords_from_text(text1_lemma)
            # Append patent's id column
            pool.append((i[0], r.get_ranked_phrases_with_scores()) )
    except:
        pass
```

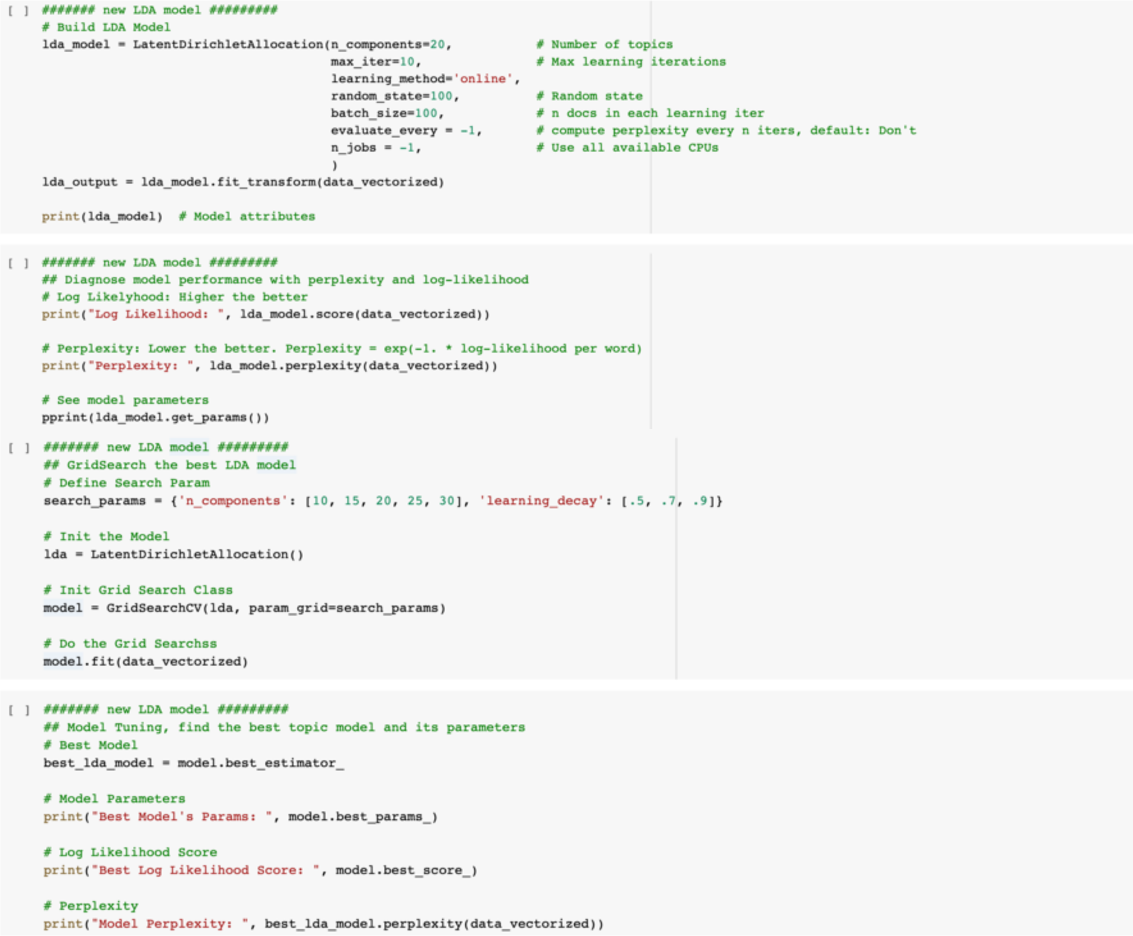
- Fianl VSM matrix after keywords extraction



keywords	access association medium device identifier premise device associate proximity platform	access network address translation functionality network address translation function network address	access user demographic information aspect may authentication user user device target communication service	access user demographic information user first medium device second medium device user demographic information	access user demographic information user identify user first user device aspect target communication service	access user demographic information user identify user plurality user device aspect target communication service	access virtualize industrial automation system industrial automation system virtualize industrial automation system	accurate refined location would node may determine base location master node	activate campaign depend continue presence user device first premise device step target communication service
patent_id									
US-10165397-B2	1000.128648	0.000000	990.881689	1051.860185	1036.425709	1028.318296	0.000000	0.000000	968.870420
US-10251022-B2	999.963010	0.000000	990.853488	1051.860185	1036.394327	1028.287516	0.000000	0.000000	968.774891
US-2016274553-A1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	959.992218	0.000000	0.000000
US-2017012813-A1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
US-2017127128-A1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
US-2017149733-A1	0.000000	894.354859	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
US-2017279892-A1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	949.648048	0.000000
US-2017302613-A1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
US-2018034642-A1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
US-2018035101-A1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

LDA TOPIC MOLDEING AND MODEL OPTIMIZATION

- Python Script for LDA Topic Modeling/GridSearch



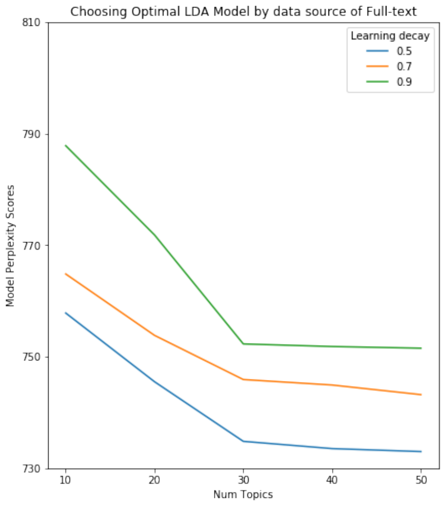
```
[ ] ####### new LDA model #########
    # Build LDA Model
    lda_model = LatentDirichletAllocation(n_components=20,           # Number of topics
                                          max_iter=10,               # Max learning iterations
                                          learning_method='online',
                                          random_state=100,          # Random state
                                          batch_size=100,            # n docs in each learning iter
                                          evaluate_every = -1,       # compute perplexity every n iters, default: Don't
                                          n_jobs = -1,               # Use all available CPUs
                                          )
    lda_output = lda_model.fit_transform(data_vectorized)

    print(lda_model)  # Model attributes
```

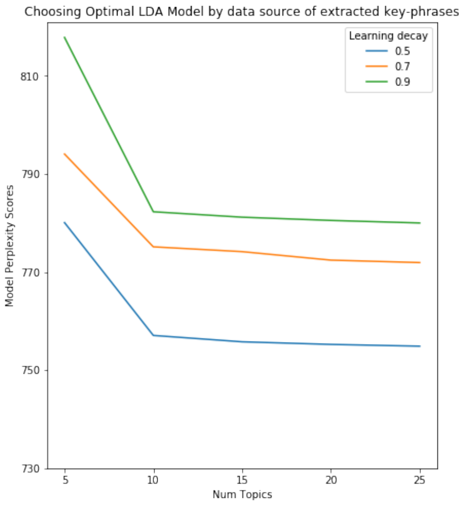
```
[ ] ####### new LDA model #########
    ## Diagnose model performance with perplexity and log-likelihood
    # Log Likelyhood: Higher the better
    print("Log Likelihood: ", lda_model.score(data_vectorized))

    # Perplexity: Lower the better. Perplexity = exp(-1. * log-likelihood per word)
    print("Perplexity: ", lda_model.perplexity(data_vectorized))

    # See model parameters
    pprint(lda_model.get_params())
```

```
[ ] ####### new LDA model #########
    ## GridSearch the best LDA model
    # Define Search Param
    search_params = {'n_components': [10, 15, 20, 25, 30], 'learning_decay': [.5, .7, .9]}

    # Init the Model
    lda = LatentDirichletAllocation()

    # Init Grid Search Class
    model = GridSearchCV(lda, param_grid=search_params)

    # Do the Grid Searchss
    model.fit(data_vectorized)
```

```
[ ] ####### new LDA model #########
    ## Model Tuning, find the best topic model and its parameters
    # Best Model
    best_lda_model = model.best_estimator_

    # Model Parameters
    print("Best Model's Params: ", model.best_params_)

    # Log Likelihood Score
    print("Best Log Likelihood Score: ", model.best_score_)

    # Perplexity
    print("Model Perplexity: ", best_lda_model.perplexity(data_vectorized))
```

- Model Optimization & Final selection of best model

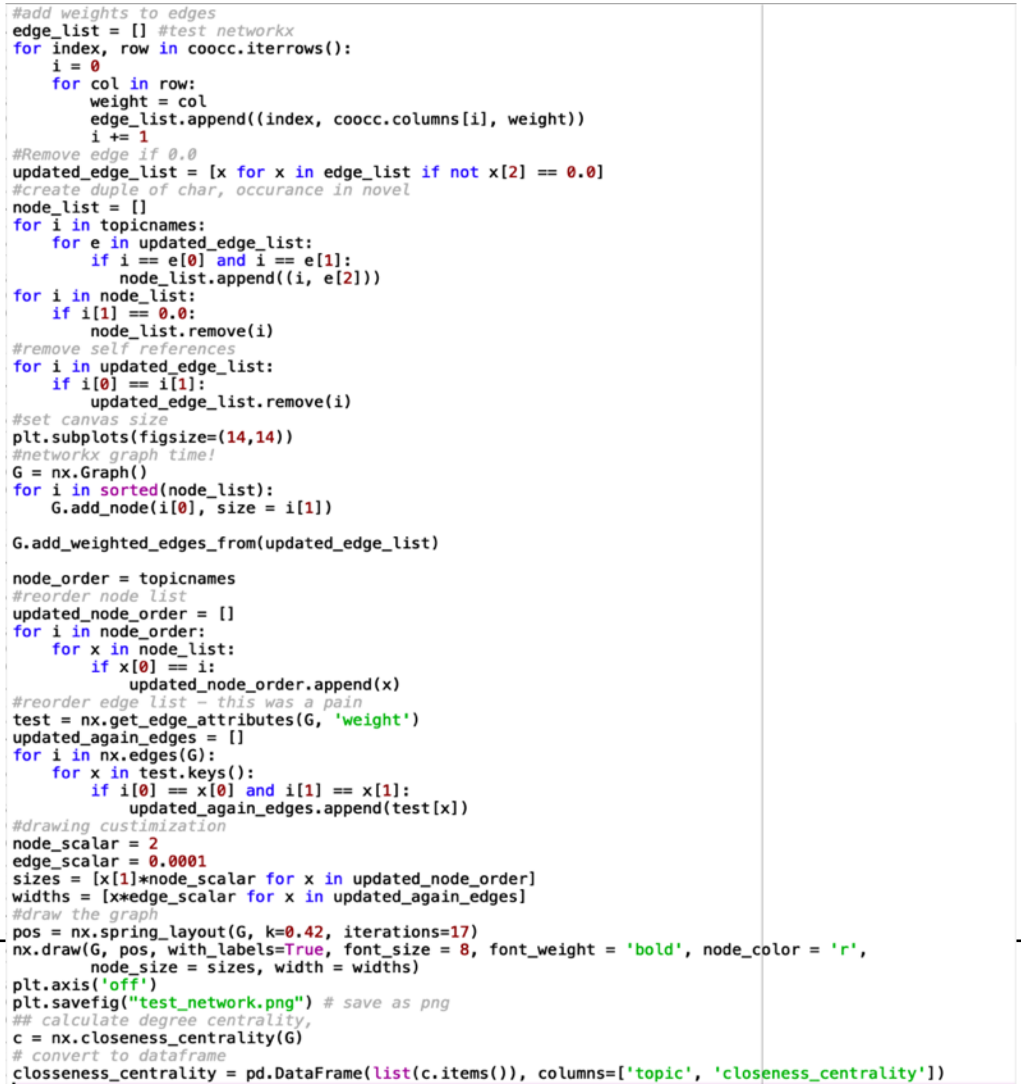

Choosing Optimal LDA Model by data source of Full-text


Choosing Optimal LDA Model by data source of extracted key-phrases

Metric		Value for the model by full-text data source	Value for the model by extracted key-phrase
Best model's parameters	Learning decay	0.5	0.5
	Number of topics	30	10
Best Log Likelihood Score		-35130325.64	-37261401.92
Best Model Perplexity		734.83	755.79

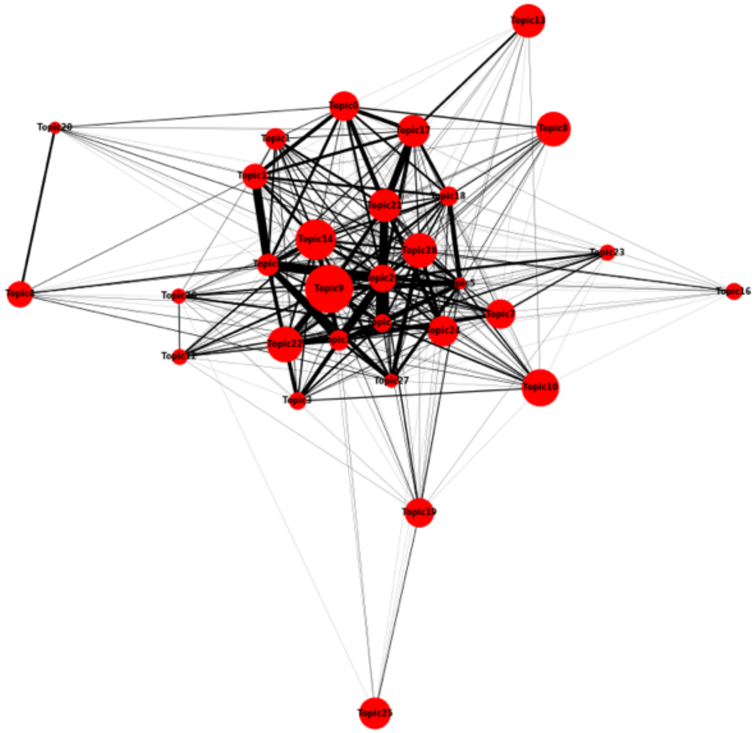
NETWORK ANALYSIS FOR IMPORTANCE INDEX

- Python Script for Network Analysis



```
#add weights to edges
edge_list = [] #test networkx
for index, row in coocc.iterrows():
    i = 0
    for col in row:
        weight = col
        edge_list.append((index, coocc.columns[i], weight))
        i += 1
#Remove edge if 0.0
updated_edge_list = [x for x in edge_list if not x[2] == 0.0]
#create duple of char, occurance in novel
node_list = []
for i in topicnames:
    for e in updated_edge_list:
        if i == e[0] and i == e[1]:
           node_list.append((i, e[2]))
for i in node_list:
    if i[1] == 0.0:
        node_list.remove(i)
#remove self references
for i in updated_edge_list:
    if i[0] == i[1]:
        updated_edge_list.remove(i)
#set canvas size
plt.subplots(figsize=(14,14))
#networkx graph time!
G = nx.Graph()
for i in sorted(node_list):
    G.add_node(i[0], size = i[1])

G.add_weighted_edges_from(updated_edge_list)

node_order = topicnames
#reorder node list
updated_node_order = []
for i in node_order:
    for x in node_list:
        if x[0] == i:
            updated_node_order.append(x)
#reorder edge list - this was a pain
test = nx.get_edge_attributes(G, 'weight')
updated_again_edges = []
for i in nx.edges(G):
    for x in test.keys():
        if i[0] == x[0] and i[1] == x[1]:
            updated_again_edges.append(test[x])
#drawing custimization
node_scalar = 2
edge_scalar = 0.0001
sizes = [x[1]*node_scalar for x in updated_node_order]
widths = [x*edge_scalar for x in updated_again_edges]
#draw the graph
pos = nx.spring_layout(G, k=0.42, iterations=17)
nx.draw(G, pos, with_labels=True, font_size = 8, font_weight = 'bold', node_color = 'r',
        node_size = sizes, width = widths)
plt.axis('off')
plt.savefig("test_network.png") # save as png
## calculate degree centrality,
c = nx.closeness_centrality(G)
# convert to dataframe
closseness_centrality = pd.DataFrame(list(c.items()), columns=['topic', 'closeness_centrality'])
```

- Model Optimization & Final selection of best model



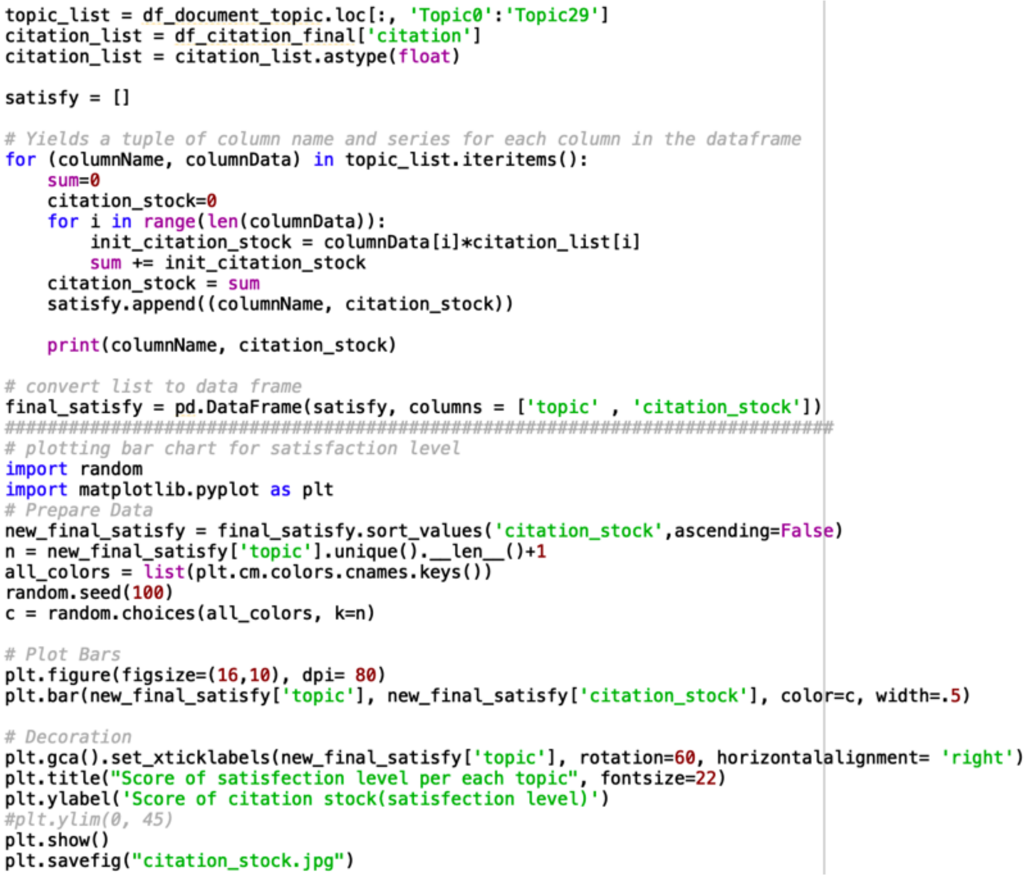
- Compute importance level ($C_c(v)$) : via closeness centrality

$$C_c(v) = {}^{(N-1)}\Big/_{\sum_{v \neq t \epsilon V} d(v,t)},$$

- *d(v,t)* = the distance, which is the shortest path, between nodes v and t
- N = the number of nodes in the network for obtaining normalized closeness centrality values

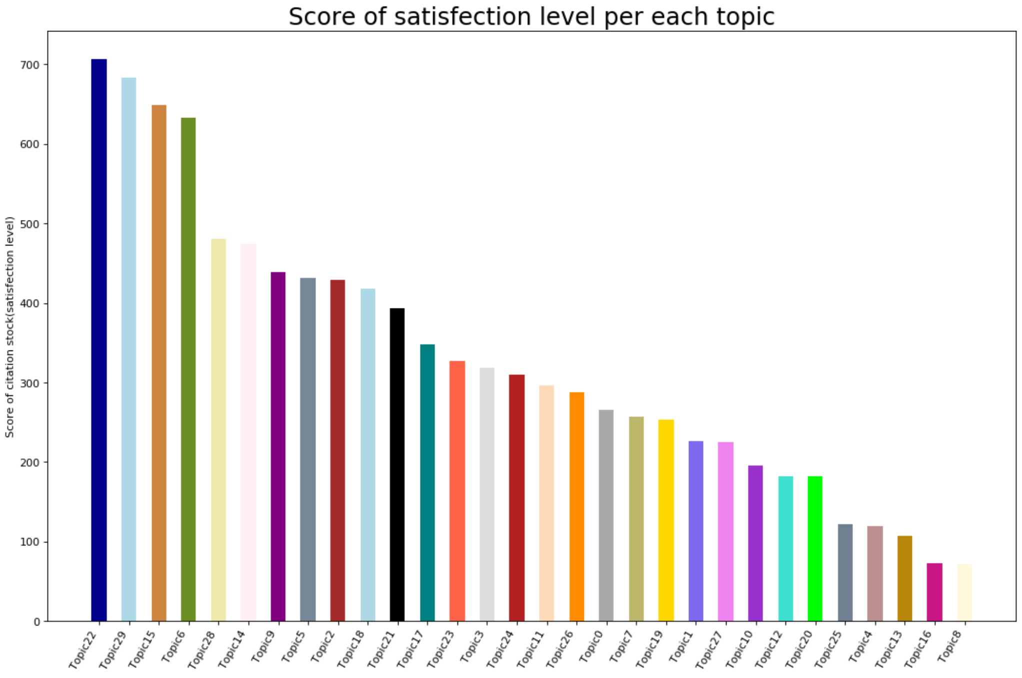
CITATION ANALYSIS FOR MATURITY INDEX

- Python Script for Network Analysis



```
topic_list = df_document_topic.loc[:, 'Topic0':'Topic29']
citation_list = df_citation_final['citation']
citation_list = citation_list.astype(float)

satisfy = []

# Yields a tuple of column name and series for each column in the dataframe
for (columnName, columnData) in topic_list.iteritems():
    sum=0
    citation_stock=0
    for i in range(len(columnData)):
        init_citation_stock = columnData[i]*citation_list[i]
        sum += init_citation_stock
    citation_stock = sum
    satisfy.append((columnName, citation_stock))

    print(columnName, citation_stock)

# convert list to data frame
final_satisfy = pd.DataFrame(satisfy, columns = ['topic' , 'citation_stock'])
##############################################################################
# plotting bar chart for satisfaction level
import random
import matplotlib.pyplot as plt
# Prepare Data
new_final_satisfy = final_satisfy.sort_values('citation_stock',ascending=False)
n = new_final_satisfy['topic'].unique().__len__()+1
all_colors = list(plt.cm.colors.cnames.keys())
random.seed(100)
c = random.choices(all_colors, k=n)

# Plot Bars
plt.figure(figsize=(16,10), dpi= 80)
plt.bar(new_final_satisfy['topic'], new_final_satisfy['citation_stock'], color=c, width=.5)

# Decoration
plt.gca().set_xticklabels(new_final_satisfy['topic'], rotation=60, horizontalalignment= 'right')
plt.title("Score of satisfection level per each topic", fontsize=22)
plt.ylabel('Score of citation stock(satisfection level)')
#plt.ylim(0, 45)
plt.show()
plt.savefig("citation_stock.jpg")
```

- Model Optimization & Final selection of best model



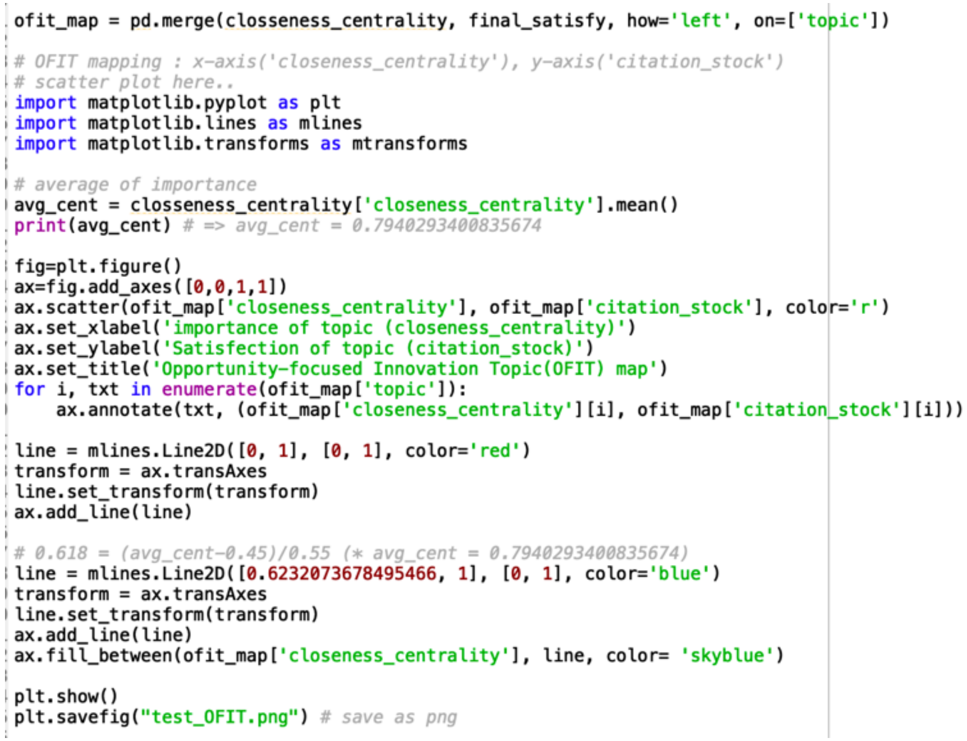
- Copute satisfaction level : via computing citation stock (CSi)

$$CS_i = \sum_{j \in S} CP_{ij} \times FC_{j,}$$

- $CPij$ = contribution probability of innovation topic i in patent j
- FCj = the number of forward citations of j

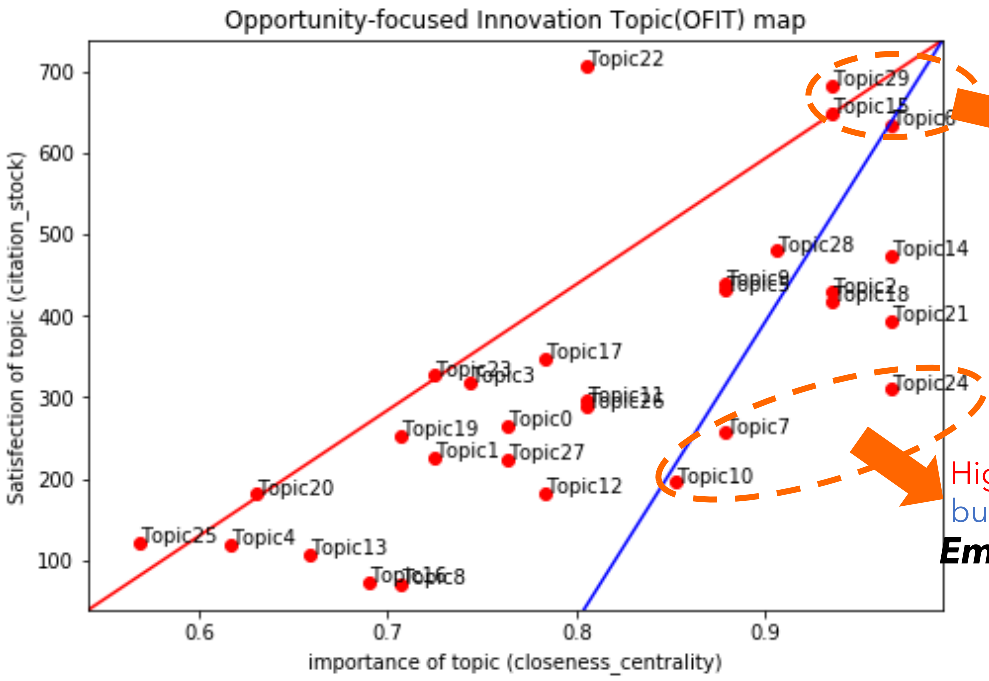
FORECASTING FUTURE EMERGING OPPORTUNITY

- Python Script for Network Analysis



```
ofit_map = pd.merge(closseness_centrality, final_satisfy, how='left', on=['topic'])

# OFIT mapping : x-axis('closeness_centrality'), y-axis('citation_stock')
# scatter plot here..
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
import matplotlib.transforms as mtransforms

# average of importance
avg_cent = closseness_centrality['closeness_centrality'].mean()
print(avg_cent) # => avg_cent = 0.7940293400835674

fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.scatter(ofit_map['closeness_centrality'], ofit_map['citation_stock'], color='r')
ax.set_xlabel('importance of topic (closeness_centrality)')
ax.set_ylabel('Satisfection of topic (citation_stock)')
ax.set_title('Opportunity-focused Innovation Topic(OFIT) map')
for i, txt in enumerate(ofit_map['topic']):
    ax.annotate(txt, (ofit_map['closeness_centrality'][i], ofit_map['citation_stock'][i]))

line = mlines.Line2D([0, 1], [0, 1], color='red')
transform = ax.transAxes
line.set_transform(transform)
ax.add_line(line)

# 0.618 = (avg_cent-0.45)/0.55 (* avg_cent = 0.7940293400835674)
line = mlines.Line2D([0.6232073678495466, 1], [0, 1], color='blue')
transform = ax.transAxes
line.set_transform(transform)
ax.add_line(line)
ax.fill_between(ofit_map['closeness_centrality'], line, color= 'skyblue')

plt.show()
plt.savefig("test_OFIT.png") # save as png
```

- Model Optimization & Final selection of best model



High importance
& high maturity:
Already Red Ocean!

High importance
but low maturity:
Emerging business topic!

POSSIBLE APPLICATION

Supporting **data driven strategy**

- Identifying investment opportunity

- Forecasting emerging opportunity

Data driven Long-term planning

- Identifying potential M&A candidate

- Finding out potential Start-Up with emerging technology

Market analysis

- Landscaping the market’s interest

-Examination of customer’s interest trend