**CONFERENCE RECOMMENDER SYSTEM**

In this project we will use topic modelling and document similarity to recommend conferences to users based on the work they have published previously. We use two datasets for this project:

1. Arxiv dataset from Kaggle.com - <https://www.kaggle.com/neelshah18/arxivdataset> - This dataset contains information about the authors, summary of paper, title and tag.
2. Conference details dataset scraped from the website <http://www.wikicfp.com/cfp/> - For information on conference Call for Papers and conference names.

Some of the techniques implemented in this project include:

1. TF-IDF Transformation
2. LDA Topic Modelling
3. Document Similarity

**Setup:**

**# install pandas**

pip install pandas

**# install genism**

pip install –upgrade genism

**# install nltk**

sudo pip install -U nltk

**#install numpy**

sudo pip install -U numpy

**#install BeautifulSoup**

pip install beautifulsoup4

**#install urllib3**

pip install urllib3

**Start:**

Let us start by importing the following libraries:

import json  
from pandas.io.json import json\_normalize  
import pandas as pd  
import gensim  
from nltk.stem import WordNetLemmatizer  
from nltk.stem.porter import \*  
stemmer = PorterStemmer()  
from urllib.request import urlopen  
from bs4 import BeautifulSoup  
from gensim import similarities  
import numpy as np  
np.random.seed(2018)  
  
import nltk  
nltk.download('wordnet')

**JSON file conversion:**

The arxiv dataset available from kaggle.com is a JSON file. The semi-structured JSON data is normalized into a flat table.

output\_json = json.load(open('./CRS\_Data\_Files/arxivData.json'))  
  
# Print one line  
print(output\_json[1])  
  
# Converts json tree structure to single line of text data  
df = json\_normalize(output\_json)  
  
print(df.head())  
  
df["all\_text"] = df["title"] + ". " + df["summary"]  
  
# Converts all /n to spaces, lambda for looping  
df["all\_text"] = df["all\_text"].map(lambda x: x.replace("\n", " "))  
data\_text = df["all\_text"]  
data\_text['index'] = data\_text.index  
documents = data\_text

*The output for print(output\_json[1]):*

{'author': "[{'name': 'Ji Young Lee'}, {'name': 'Franck Dernoncourt'}]", 'day': 12, 'id': '1603.03827v1', 'link': "[{'rel': 'alternate', 'href': 'http://arxiv.org/abs/1603.03827v1', 'type': 'text/html'},

*Output for print(df.head()):*

0 [{'name': 'Ahmed Osman'}, {'name': 'Wojciech S... ... 2018

1 [{'name': 'Ji Young Lee'}, {'name': 'Franck De... ... 2016

2 [{'name': 'Iulian Vlad Serban'}, {'name': 'Tim... ... 2016

3 [{'name': 'Sebastian Ruder'}, {'name': 'Joachi... ... 2017

4 [{'name': 'Iulian V. Serban'}, {'name': 'Chinn... ... 2017

*Snippet of the output for print(df["all\_text"].head(100)):*

0 Dual Recurrent Attention Units for Visual Ques...

1 Sequential Short-Text Classification with Recu...

2 Multiresolution Recurrent Neural Networks: An ...

3 Learning what to share between loosely related...

4 A Deep Reinforcement Learning Chatbot. We pres...

5 Generating Sentences by Editing Prototypes. We...

6 A Deep Reinforcement Learning Chatbot (Short V...

7 Document Image Coding and Clustering for Scrip...

8 Tutorial on Answering Questions about Images w...

9 pix2code: Generating Code from a Graphical Use...

10 A Unified Deep Neural Network for Speaker and ...

**Data Pre-processing:**

For the data pre-processing stage, the following steps are performed:

1. Tokenization: The words are broken down into sentences and then words. The words are then lowercased and the punctuation marks are removed.
2. Words that have fewer than three characters and all the stop words are also removed.
3. The words are then stemmed and lemmatized to return only the base form.

def stemmLemm (text):  
 return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))  
  
def preprocess(text):  
 result = []  
 for token in gensim.utils.simple\_preprocess(text):  
 if token not in gensim.parsing.preprocessing.STOPWORDS and len(token) > 3:  
 result.append(stemmLemm(token))  
 return result

*Output snippet before stemming and lemmatization:*

Sequential Short-Text Classification with Recurrent and Convolutional Neural Networks. Recent approaches based on artificial neural networks (ANNs) have shown promising results for short-text classification. However, many short texts occur in sequences (e.g., sentences in a document or utterances in a dialog), and most existing ANN-based systems do not leverage the preceding short texts when classifying a subsequent one. In this work, we present a model based on recurrent neural networks and convolutional neural networks that incorporates the preceding short texts. Our model achieves state-of-the-art results on three different datasets for dialog act prediction.

['Sequential', 'Short-Text', 'Classification', 'with', 'Recurrent', 'and', 'Convolutional', '', '', 'Neural', 'Networks.', 'Recent', 'approaches', 'based', 'on', 'artificial', 'neural', 'networks', '(ANNs)', 'have', 'shown', 'promising', 'results', 'for', 'short-text', 'classification.', 'However,', 'many', 'short', 'texts', 'occur', 'in', 'sequences', '(e.g.,', 'sentences', 'in', 'a', 'document', 'or', 'utterances', 'in', 'a', 'dialog),', 'and', 'most', 'existing', 'ANN-based', 'systems', 'do', 'not', 'leverage', 'the', 'preceding', 'short', 'texts', 'when', 'classifying', 'a', 'subsequent', 'one.', 'In', 'this', 'work,', 'we', 'present', 'a', 'model', 'based', 'on', 'recurrent', 'neural', 'networks', 'and', 'convolutional', 'neural', 'networks', 'that', 'incorporates', 'the', 'preceding', 'short', 'texts.', 'Our', 'model', 'achieves', 'state-of-the-art', 'results', 'on', 'three', 'different', 'datasets', 'for', 'dialog', 'act', 'prediction.']

*Output snippet after stemming and lemmatization:*

tokenized and lemmatized document:

['sequenti', 'short', 'text', 'classif', 'recurr', 'convolut', 'neural', 'network', 'recent', 'approach', 'base', 'artifici', 'neural', 'network', 'ann', 'show', 'promis', 'result', 'short', 'text', 'classif', 'short', 'text', 'occur', 'sequenc', 'sentenc', 'document', 'utter', 'dialog', 'exist', 'base', 'system', 'leverag', 'preced', 'short', 'text', 'classifi', 'subsequ', 'work', 'present', 'model', 'base', 'recurr', 'neural', 'network', 'convolut', 'neural', 'network', 'incorpor', 'preced', 'short', 'text', 'model', 'achiev', 'state', 'result', 'differ', 'dataset', 'dialog', 'predict']

For ease of running and training the model, the first 10000 are saved in ‘processed\_docs’.

processed\_docs = documents[1:10000].map(preprocess)  
print(processed\_docs)

*Output snippet from processed\_docs:*

1 [sequenti, short, text, classif, recurr, convo...

2 [multiresolut, recurr, neural, network, applic...

3 [learn, share, loos, relat, task, multi, task,...

Next, a dictionary is created from ‘processed\_docs’ that lists how many times a word has appeared in the training data set.

dictionary = gensim.corpora.Dictionary(processed\_docs)  
count = 0  
for k, v in dictionary.iteritems():  
 print(k, v)  
 count += 1  
 if count > 10:  
 break

*Output for the dictionary:*

0 achiev

1 ann

2 approach

3 artifici

4 base

5 classif

6 classifi

7 convolut

8 dataset

9 dialog

10 differ

Next, we are going to filter out tokens that appears in less than 15 documents and not more than 0.5 and keep only the 100000 most frequent tokens.

dictionary.filter\_extremes(no\_below=15, no\_above=0.5, keep\_n=100000)

Now we can preview the Bag of Words and how many times it appears:

bow\_doc\_4000 = bow\_corpus[4000]  
for i in range(len(bow\_doc\_4000)):  
 print("Word {} (\"{}\") appears {} time.".format(bow\_doc\_4000[i][0], dictionary[bow\_doc\_4000[i][0]], bow\_doc\_4000[i][1]))

*Output snippet for the same:*

Word 2 ("approach") appears 1 time.

Word 15 ("network") appears 2 time.

Word 16 ("neural") appears 2 time.

Word 20 ("present") appears 1 time.

Word 32 ("system") appears 1 time.

Word 44 ("automat") appears 1 time.

Word 63 ("extract") appears 1 time.

Word 74 ("long") appears 1 time.

Word 87 ("propos") appears 1 time.

Word 96 ("studi") appears 1 time.

Word 102 ("term") appears 1 time.

Word 105 ("train") appears 1 time.

Word 113 ("common") appears 1 time.

**TF-IDF Transformation:**

TF-IDF Transformation is then applied to the whole corpus:

from gensim import corpora, models  
tfidf = models.TfidfModel(bow\_corpus)  
corpus\_tfidf = tfidf[bow\_corpus]  
from pprint import pprint  
for doc in corpus\_tfidf:  
 pprint(doc)  
 break

*Output snippet with scores:*

[(0, 0.05098951855196534),

(1, 0.1911788401793679),

(2, 0.031213402022883453),

(3, 0.09956332323986432),

(4, 0.07788070974553975),

(5, 0.09822243081700553),

(6, 0.0658709209295156),

(7, 0.1320042159043463),

(8, 0.04133106350249891),

(9, 0.34792641221160636),

Now we will run the LDA using TF-IDF:

lda\_model\_tfidf = gensim.models.LdaMulticore(corpus\_tfidf, num\_topics=10, id2word=dictionary, passes=2, workers=4)  
for idx, topic in lda\_model\_tfidf.print\_topics(-1):  
 print('Topic: {} Word: {}'.format(idx, topic))

*This produces the following output:*

Topic: 0 Word: 0.007\*"topic" + 0.005\*"imag" + 0.004\*"text" + 0.004\*"word" + 0.004\*"network" + 0.004\*"semant" + 0.004\*"detect" + 0.003\*"video" + 0.003\*"featur" + 0.003\*"data"

Topic: 1 Word: 0.005\*"user" + 0.004\*"imag" + 0.003\*"method" + 0.003\*"algorithm" + 0.003\*"gener" + 0.003\*"recommend" + 0.003\*"data" + 0.003\*"network" + 0.003\*"polici" + 0.003\*"cluster"

Topic: 2 Word: 0.004\*"train" + 0.004\*"classif" + 0.004\*"network" + 0.004\*"algorithm" + 0.004\*"featur" + 0.003\*"distribut" + 0.003\*"data" + 0.003\*"method" + 0.003\*"gener" + 0.003\*"optim"

Performance Evaluation using the LDA TF-IDF model:

When we check where the test document would be classified:

for index, score in sorted(lda\_model\_tfidf[bow\_corpus[4000]], key=lambda tup: -1\*tup[1]):  
 print("\nScore: {}\t \nTopic: {}".format(score, lda\_model\_tfidf.print\_topic(index, 10)))

*Output:*

Score: 0.610015332698822

Topic: 0.007\*"topic" + 0.005\*"imag" + 0.004\*"text" + 0.004\*"word" + 0.004\*"network" + 0.004\*"semant" + 0.004\*"detect" + 0.003\*"video" + 0.003\*"featur" + 0.003\*"data"

Score: 0.22319552302360535

Topic: 0.004\*"featur" + 0.004\*"cluster" + 0.004\*"data" + 0.004\*"network" + 0.004\*"imag" + 0.004\*"predict" + 0.003\*"algorithm" + 0.003\*"classif" + 0.003\*"sentiment" + 0.003\*"text"

The next step scraps the data from the conference name and information from wikicfp.com.

# query the website and return the html to the variable ‘page’  
data = []  
for pg in quote\_page:  
 page = urlopen(pg)  
  
 # parse the html using beautiful soup and store in variable `soup`  
 soup = BeautifulSoup(page, 'html.parser')  
  
 # Take out the <div> of name and get its value  
 descr\_box = soup.find('div', attrs={'class': 'cfp'})  
 description = descr\_box.text.strip() # strip() is used to remove starting and trailing  
  
 conf\_box = soup.find('span', attrs={'property': 'v:description'})  
 conf\_name = conf\_box.text.strip() # strip() is used to remove starting and trailing  
  
 data.append((conf\_name, description))  
 # print(name)

This is then converted to a dataframe and then passed as the ‘unseen document’ to determine the similarity.

print(data)  
test\_df = pd.DataFrame(data, columns=["conf\_name", "description"])  
unseen\_document = test\_df.loc[1,"conf\_name"] + ' ' + test\_df.loc[1,"description"]

The ‘unseen document’ which is the conference information is then passed to the bow\_vector and then passed on to the model we trained.

bow\_vector = dictionary.doc2bow(preprocess(unseen\_document))  
for index, score in sorted(lda\_model\_tfidf[bow\_vector], key=lambda tup: -1\*tup[1]):  
 print("Score: {}\t Topic: {}".format(score, lda\_model\_tfidf.print\_topic(index, 5)))

The similarity is then determined using Matrix similarity.

lda\_index = similarities.MatrixSimilarity(lda\_model\_tfidf[corpus\_tfidf])  
  
# Let's perform some queries  
similarities = lda\_index[lda\_model\_tfidf[bow\_vector]]  
# Sort the similarities  
similarities = sorted(enumerate(similarities), key=lambda item: -item[1])  
  
# Top most similar documents:  
print(similarities[:10])  
  
# Let's see what's the most similar document  
document\_id, similarity = similarities[0]  
print ("These are the document similarities ")  
print(documents[document\_id][:1000])

In the output this the information below is regarded as the unseen document which is the conference information:

MLDM 2019 : 15th International Conference on Machine Learning and Data Mining MLDM 2019 The Aim of the Conference

The aim of the conference is to bring together researchers from all over the world who deal with machine learning and data mining in order to discuss the recent status of the research and to direct further developments. Basic research papers as well as application papers are welcome.

« top

Topics of the conference

All kinds of applications are welcome but special preference will be given to multimedia related applications, applications from live sciences and webmining.

Paper submissions should be related but not limited to any of the following topics:

association rules

case-based reasoning and learning

classification and interpretation of images, text, video

conceptional learning and clustering

Goodness measures and evaluaion (e.g. false discovery rates)

inductive learning including decision tree and rule induction learning

knowledge extraction from text, video, signals and images

mining gene data bases and biological data bases

mining images, temporal-spatial data, images from remote sensing

mining structural representations such as log files, text documents and HTML documents

mining text documents

organisational learning and evolutional learning

probabilistic information retrieval

Sampling methods

Selection with small samples

similarity measures and learning of similarity

statistical learning and neural net based learning

video mining

visualization and data mining

Applications of Clustering

Aspects of Data Mining

Applications in Medicine

Autoamtic Semantic Annotation of Media Content

Bayesian Models and Methods

Case-Based Reasoning and Associative Memory

Classification and Model Estimation

Content-Based Image Retrieval

Decision Trees

Deviation and Novelty Detection

Feature Grouping, Discretization, Selection and Transformation

Feature Learning

Frequent Pattern Mining

High-Content Analysis of Microscopic Images in Medicine, Biotechnology and Chemistry

Learning and adaptive control

Learning/adaption of recognition and perception

Learning for Handwriting Recognition

Learning in Image Pre-Processing and Segmentation

Learning in process automation

Learning of internal representations and models

Learning of appropriate behaviour

Learning of action patterns

Learning of Ontologies

Learning of Semantic Inferencing Rules

Learning of Visual Ontologies

Learning robots

Mining Images in Computer Vision

Mining Images and Texture

Mining Motion from Sequence

Neural Methods

Network Analysis and Intrusion Detection

Nonlinear Function Learning and Neural Net Based Learning

Real-Time Event Learning and Detection

Retrieval Methods

Rule Induction and Grammars

Speech Analysis

Statistical and Conceptual Clustering Methods

Statistical and Evolutionary Learning

Subspace Methods

Support Vector Machines

Symbolic Learning and Neural Networks in Document Processing

Time Series and Sequential Pattern Mining

Audio Mining

Cognition and Computer Vision

Clustering

Classification & Prediction

Statistical Learning

Association Rules

Telecommunication

Design of Experiment

Strategy of Experimentation

Capability Indices

Deviation and Novelty Detection

Control Charts

Design of Experiments

Capability Indices

Conceptional Learning

Goodness Measures and Evaluation (e.g. false discovery rates)

Inductive Learning Including Decision Tree and Rule Induction Learning

Organisational Learning and Evolutional Learning

Sampling Methods

Similarity Measures and Learning of Similarity

Statistical Learning and Neural Net Based Learning

Visualization and Data Mining

Deviation and Novelty Detection

Feature Grouping, Discretization, Selection and Transformation

Feature Learning

Frequent Pattern Mining

Learning and Adaptive Control

Learning/Adaption of Recognition and Perception

Learning for Handwriting Recognition

Learning in Image Pre-Processing and Segmentation

Mining Financial or Stockmarket Data

Mining Motion from Sequence

Subspace Methods

Support Vector Machines

Time Series and Sequential Pattern Mining

Desirabilities

Graph Mining

Agent Data Mining

Applications in Software Testing

Authors can submit their paper in long or short version.

Long Paper

The paper must be formatted in the Springer LNCS format. They should have at most 15 pages. The papers will be reviewed by the program committee. Accepted long papers will be published by Springer Verlag in the LNAI Series in the book Advances in Data Mining, edited by Petra Perner.

Short Paper

Short papers are also welcome and can be used to describe work in progress or project ideas. They can have 5 to max. 15 pages, formatted in Springer LNCS format. Accepted short papers will be presented as poster in the poster session. They will be published in a special poster proceedings book.

« top

Program Committee

Chair

Petra Perner IBaI Leipzig, Germany

Committee

Sergey Ablameyko Belarus State University, Belarus

Reneta Barneva The State University of New York at Fredonia, USA

Michelangelo Ceci Universtiy of Bari, Italy

Patrick Bouthemy INRIA VISTA, France

Xiaoqing Ding Tsinghua University, China

Christoph F. Eick Universtiy of Houston, USA

Ana Fred Technical University of Lisboa, Portugal

Giorgio Giacinto University of Cagliari, Italy

Makato Haraguchi Hokkaido University of Sapporo, Japan

Dimitris Karras Chalkis Institute of Technology, Greece

Adam Krzyzak Concordia University, Canada

Thang V. Pham University of Amsterdam, The Netherlands

Linda Shapiro University of Washington, USA

Tamas Sziranyi MTA-SZTAKI, Hungary

Francis E.H. Tay National University of Singapore, Singapore

Alexander Ulanov HP Labs, Russia

Zeev Volkovich ORT Braude College of Engineering, Israel

Patrick Wang Northeastern University, USA

An industrial exhibition running in connection with the conference will give you the opportunity to look at new trends and systems in industry and to present your research to industry.

The document below was found out to be the top scoring document with the highest similarity. As we look at the contexts for both documents we can see that the both of them talk about new trends in the industry and processing of signals, telecommunications and speech analysis are some of the topics mentioned in the Call for Papers.

Graph-Based Manifold Frequency Analysis for Denoising. We propose a new framework for manifold denoising based on processing in the graph Fourier frequency domain, derived from the spectral decomposition of the discrete graph Laplacian. Our approach uses the Spectral Graph Wavelet transform in order to per- form non-iterative denoising directly in the graph frequency domain, an approach inspired by conventional wavelet-based signal denoising methods. We theoretically justify our approach, based on the fact that for smooth manifolds the coordinate information energy is localized in the low spectral graph wavelet sub-bands, while the noise affects all frequency bands in a similar way. Experimental results show that our proposed manifold frequency denoising (MFD) approach significantly outperforms the state of the art denoising meth- ods and is robust to a wide range of parameter selections, e.g., the choice of k nearest neighbor connectivity of the graph.

There is a lot of room for improvement in this project, but I have tried to keep the scope achievable in the given time-frame. This would be a good opportunity to send marketing emails for conferences to those folks who have published similar papers in the past.