Capstone Project 2, Final Report:

Academic Paper Recommender

Overview:

The aim of this project is to create a recommender system to help researchers and scientists find related articles to a specific paper. There are thousands of papers published everyday and it is not feasible for a researcher to browse all these papers in order to find a related paper to a desired subject. This recommender system helps scientists with their search and saves them a lot of time looking into article resources.

Dataset:

The data for this project is collected using API for PubMed, which is a repository for biomedical data. The data is extracted in raw XML in a full text format and information such as paper's ID, title, authors last name, year, journal, abstract, tags, and citations is collected from each paper. This is programmatically queried via the NCBI Entrez E-utilities interface.

Below one can see an example of part of a dataset for an article obtained from PubMed before the useful information being extracted from it:

"{'MedlineCitation': DictElement({'CitationSubset': ['IM'], 'OtherID': [], 'OtherAbstract': [], 'KeywordList': [ListElement([StringElement('colorectal cancer', attributes={'MajorTopicYN': 'Y'}), StringElement('diagnostic', attributes={'MajorTopicYN': 'Y'}), StringElement('growth differentiation factor 15', attributes={'MajorTopicYN': 'Y'}), StringElement('meta-analysis', attributes={'MajorTopicYN': 'Y'}), StringElement('prognostic', attributes={'MajorTopicYN': 'Y'})], attributes={'Owner': 'NOTNLM'})], 'SpaceFlightMission': [], 'GeneralNote': [], 'PMID': StringElement('26990020', attributes={'Version': '1'}), 'DateCompleted': {'Year': '2017', 'Month': '07', 'Day': '07'}, 'DateRevised': {'Year': '2017', 'Month': '12', 'Day': '20'}, 'Article': DictElement({'ELocationID': [StringElement('10.1111/jcmm.12830', attributes={'EIdType': 'doi', 'ValidYN': 'Y'})], 'Language': ['eng'], 'ArticleDate': [DictElement({'Year': '2016', 'Month': '03', 'Day': '15'}, attributes={'DateType': 'Electronic'})], 'Journal': {'ISSN': StringElement('1582-4934', attributes={'IssnType': 'Electronic'}), 'JournalIssue': DictElement({'Volume': '20', 'Issue': '8', 'PubDate': {'Year': '2016', 'Month': '08'}}, attributes={'CitedMedium': 'Internet'}), 'Title': 'Journal of cellular and molecular medicine', 'ISOAbbreviation': 'J. Cell. Mol. Med.'}, 'ArticleTitle': 'Growth differentiation factor 15 is a promising diagnostic and prognostic biomarker in colorectal cancer.', 'Pagination': {'MedlinePgn': '1420-6'}, 'Abstract': {'AbstractText': ['Although various studies have demonstrated that growth differentiation factor 15 (GDF15) might be a potential diagnostic and prognostic marker in colorectal cancer (CRC) patients, the results are inconsistent and the statistical power of individual studies is also insufficient. An original study was conducted to explore the diagnostic and prognostic value of serum GDF15 in CRC patients. We also conducted a meta-analysis study which aimed to

summarize the diagnostic and prognostic performance of serum GDF15 in CRC. We searched PubMed and ISI Web of Knowledge up to 1 November 2014 for eligible studies. In order to explore the diagnostic performance of GDF15, standardized mean difference (SMD) and their 95% confidence intervals (CI) were estimated and receiver-operating characteristic (ROC) curves were constructed. For prognostic meta-analysis, study-specific hazard ratios (HRs) of serum GDF15 for survival were summarized. A total of eight studies were included in the meta-analyses. Our results revealed that serum GDF15 levels in CRC patients were higher than those in healthy controls (SMD = 1.08, 95% CI: 0.56-1.59, P < 0.001). For discriminating CRC from healthy controls, the AUC of GDF15 was 0.816 (95% CI: 0.792-0.838). The sensitivity and specificity were 58.9% (95% CI: 55.0-62.8) and 92.08% (95% CI: 89.2-94.4), respectively, when a cut-off value of 1099 pg/ml was established. Besides, higher GDF15 expression level was associated with worse overall survival for CRC patients (pooled HR = 2.09, 95% CI: 1.47-2.96). In conclusion, the present meta-analysis suggests that serum GDF15 may be a useful diagnostic and prognostic biomarker for CRC.'], 'CopyrightInformation': '© 2016 The Authors."

Data Scraping:

The Entrez Programming Utilities (E-utilities) are a set of nine server-side programs that provide a stable interface into the Entrez query and database system at the National Center for Biotechnology Information (NCBI). The E-utilities use a fixed URL syntax that translates a standard set of input parameters into the values necessary for various NCBI software components to search for and retrieve the requested data.

The code for data scraping can be found in getData.ipynb:

```
from Bio import Entrez
from Bio.Entrez import efetch
```

The information is fetched using the following functions:

```
def create_idlist(n):
    """Return a list of PMID numbers with size n. The earliest paper PMID is 28785052 published on August 2017"""
    idlist = []
    for i in range(27000009-n, 27000009):
        idlist.append(str(i))
    return idlist
```

```
def fetch_details(id_list):
    """A function that gets a list of PMIDs and returns a xml of papers' information"""
    ids = ','.join(id_list)
    handle = Entrez.efetch(db='pubmed',
                           retmode='xml',
                           id=ids)
    results = Entrez.read(handle)
    return results
```

Each useful paper information is then obtained as follows:

```
def get_title(paper):
    """Given a xml paper info, this function returns the paper's title"""
    return paper['MedlineCitation']['Article']['ArticleTitle']
```

```
def get_abstract(paper):
    """Given a xml paper info, this function returns the paper's abstract"""
    try:
        return paper['MedlineCitation']['Article']['Abstract']['AbstractText'][0]
    except:
        return np.nan
```

```
def get_year(paper):
    """Given a xml paper info, this function returns the paper's year of publication"""
    try:
        return paper['MedlineCitation']['Article']['Journal']['JournalIssue']['PubDate']['Year']
    except:
        return np.nan
```

```
def get_journal(paper):
    """Given a xml paper info, this function returns the paper's journal name"""
    try:
        return paper['MedlineCitation']['Article']['Journal']['Title']
    except:
        return np.nan
```

```
def get_citations(paper):
    """Given a xml paper info, this function returns a list containing all the PMIDs for paper's citations"""
    citations = []
    try:
        for citation in paper['MedlineCitation']['CommentsCorrectionsList']:
            citations.append(str(citation['PMID']))
    except:
        citations = np.nan
    return citations
```

```
def get_tags(paper):
    """Given a xml paper info, this function returns a list containing all tags for the paper"""
    tags_list = []
    try:
        for tag in paper['MedlineCitation']['MeshHeadingList']:
            tags_list.append(str(tag['DescriptorName']))
    except:
        tags_list = np.nan
    return tags_list
```

```
def get_authors(paper):
    """Given a xml paper info, this function returns a list containing last names of paper's authors"""
    authors_list = []
    try:
        for auth in paper['MedlineCitation']['Article']['AuthorList']:
            authors_list.append(str(auth['LastName']))
    except:
        authors_list = np.nan
    return authors_list
```

Then the data frame for all papers is created as follows:

```
def get_paper_info(paper, id):
    """Given paper's xml and PMID, it returns a tuple containing infomration about the paper"""

    title = get_title(paper)
    authors = get_authors(paper)
    tags = get_tags(paper)
    citations = get_citations(paper)
    year = get_year(paper)
    abstract = get_abstract(paper)
    journal = get_journal(paper)
    return (id, title, authors, year, journal, abstract, tags, citations)
```

```
if __name__ == '__main__':
    id_list = create_idlist(10000)
    print(len(id_list))
    try:
        papers = fetch_details(id_list)
    except:
        pass

    paper_list = []

    for i, paper in enumerate(papers['PubmedArticle']):
        #if i==20:
        #    print (paper)
        paper_list.append(get_paper_info(paper, id_list[i]))

    df = pd.DataFrame(paper_list, columns=['id', 'title', 'authors', 'year', 'journal', 'abstract', 'tags', 'citations'
```

	id	title	authors	year	journal	abstract	tags	citations
0	26990009	Identifying Older Adults with Serious Illness:...	[Kelley, Covinsky, Gorges, McKendrick, Bollens...	2017	Health services research	To create and test three prospective, increasi...	[Activities of Daily Living, Aged, Aged, 80 an...	[15493448, 17187548, 23838378, 9441588, 198285...
1	26990010	Social rank versus affiliation: Which is more ...	[Wang, Sun, Sheeran, Sun, Zhang, Zhang, Xia, Li]	2016	American journal of primatology	Research on leadership is a critical step for ...	[Animals, Grooming, Leadership, Macaca, Moveme...	NaN
2	26990011	Three-dimensional manometry of the upper esoph...	[Meyer, Jones, Walczak, McCulloch]	2016	The Laryngoscope	High-resolution manometry (HRM) is useful in i...	[Adult, Deglutition, Deglutition Disorders, Es...	[10718434, 23728150, 16410365, 17305278, 44782...

10000 papers up to the PMID of 27000000 is extracted and data for PMID, title, authors, publications, year, name of journal, abstract, tags, and citations is used in a dataframe.

Data Cleaning:

Since the search is going to be based on title and/or abstract and also tags, the papers with missing title and tags were removed from the data. We did not remove papers which have title but no abstract. In this case, only title is used to find similar papers. The resulting dataset information is as follows:

```
df.dropna(axis=0, subset=['title','tags'], inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7690 entries, 0 to 9631
Data columns (total 8 columns):
id           7690 non-null object
title        7690 non-null object
authors      7554 non-null object
year         7315 non-null object
journal      7690 non-null object
abstract     6806 non-null object
tags         7690 non-null object
citations    2656 non-null object
dtypes: object(8)
memory usage: 540.7+ KB
```

The data is saved as a .csv file and is uploaded in Databricks, which is a web-based platform for working with Spark, and provides automated cluster management and IPython-style notebooks.

After loading the full data back into the Databricks as an RDD, the text needs to be cleaned. remove_punctuation function is defined as below. This function removes the punctuation like commas and quotes from the text (string). We also want to keep contractions together. The method also makes the words lower cased. The function returns a list or words in the text. The code can be found in recommender.ipynb

```
def remove_punctuation(text):
    """ This method removes the punctuation like commas and quotes from the text (string).
    We also want to keep contractions together. The method also make the words lower cased.
    It returns a list or words in the text
        Args:
            text (string): the text we want to clean
        Return:
            A list with cleaned words
    """
    # split into words by white space
    words = text.split()
    words_lower = [w.lower() for w in words]

    # Remove punctuation from each word
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in words_lower]
    return ' '.join(stripped)
```

“Tokenizer” and “StopWordsRemover” from pyspark.ml.feature are also used to clean the data.

Tokenization is the process of taking text (such as a sentence) and breaking it into individual terms (usually words).

Stop words are words which should be excluded from the input, typically because the words appear frequently and don’t carry as much meaning. StopWordsRemover takes as input a sequence of strings (e.g. the output of a Tokenizer) and drops all the stop words from the input sequences. The list of stopwords is specified by the stopWords parameter.

Modeling Approach:

Different methods and information is used to come up with a recommender for a paper.

The first approach is text similarity-based recommender. In this method, title and abstract of each paper is used and a TF-IDF based similarity is calculated in order to recommend the n number of related papers.

The second method is semantic similarity based recommender where the tags are used for determine similarity between papers. Papers which are sharing the most number of tags with the reference paper would be recommended.

The recommendation algorithms are implemented in Spark using Python, and are run using the web-based platform, Databricks, on their provided automated cluster.

TF-IDF:

TF-IDF stands for term frequency-inverse document frequency. TF-IDF is a feature vectorization method used in text mining to evaluate the importance of a word in a document or corpus. The importance increases proportionally to the number of times a word appears in the document but is reduced if the frequency of the word is high in the corpus.

Denote a term by t, a document by d, and the corpus by D. Term frequency $TF(t,d)$ is the number of times that term t appears in document d, while document frequency $DF(t,D)$ is the number of documents that contains term t. If we only use term frequency to measure the importance, it is very easy to over-emphasize terms that appear very often but carry little information about the document. If a term appears very often across the corpus, it means it doesn't carry special information about a particular document. Inverse document frequency is a numerical measure of how much information a term provides:

$$IDF(t, D) = \log \frac{|D| + 1}{DF(t, D) + 1},$$

where $|D|$ is the total number of documents in the corpus. The TF-IDF is then defined as:

$$TFIDF(t, d, D) = TF(t, d) \cdot IDF(t, D).$$

TF: Here, HashingTF from spark.ml is used to generate the term frequency vectors. HashingTF is a transformer which takes sets of terms and converts those sets into fixed-length feature vectors.

IDF: IDF is an estimator which is fit on a dataset and produces an IDFModel. The IDFModel takes feature vectors (Created from HashingTF) and scales each feature. It also down-weights features which appear frequently in a corpus.

```
print("... Computing TF-IDF ...")
hashingTF = HashingTF(inputCol="filtered", outputCol="rawFeatures")
tf = hashingTF.transform(wordsData).cache()

idfModel = IDF(inputCol="rawFeatures", outputCol="features")
idf = idfModel.fit(tf)
tfidf = idf.transform(tf)
```

The output of the TF-IDF is then passed through a Normalizer. Normalizer is a transformer which transforms a dataset of vector rows, normalizing each vector to have unit norm. It takes parameter p, which specifies the p-norm used for normalization. (p=2 by default). This normalization can help standardize the input data and improve the behavior of learning algorithms.

Cosine Similarity of the vectors is then calculated using the Cartesian product and the function dot on numpy arrays to produce a similarity array between each pair of papers in the whole dataset.

```
# get the similarities for each pair of papers
def get_similarities(paper_rdd, paper_tfidf):
    """ Function that returns the array of similarities between each two papers
        Args:
            paper_rdd (rdd): idd of all papers abstract and titles
            paper_tfidf (pyspark.sql.dataframe.DataFrame): tf-idf vectors for a given paper
        Return:
            similarity_array:  array of cosine similarities for each pair of papers
    """

    print("... Computing L2 norm ...")
    labels = paper_rdd.map(lambda x: x[0])
    features = paper_tfidf

    normalizer = Normalizer(inputCol="features", outputCol="normFeatures")
    data = labels.zip(normalizer.transform(features).rdd.map(lambda r: r.normFeatures))

    #Using a Cartesian product and the function dot on numpy arrays:
    similarity_array = data.cartesian(data)\
    .map(lambda l: ((l[0][0], l[1][0]), l[0][1].dot(l[1][1])))\
    .sortByKey()

    return similarity_array
```

Using this similarity array, based on the preferred method of similarity (title and abstract or tag-based), for a given paper, the n most similar papers can be sorted and outputted:

```
# get the n top similar papers for given paper info
def get_neighbors(paper_PMID, similarity_array, n):
    """ Function that returns the 50 most similar papers for given paper
        Args:
            paper_PMID (int): PMID of the paper we want to find similar papers to
            similarity_array (array): cosine similarity array for all papers
            n (int): number of similar papers we are looking for, for a specified paper
        Return:
            list: the list of papers relevant to the given paper based on cosine similarity
    """
    candidates = similarity_array.filter(lambda x: x[0][0]==paper_PMID).sortBy(lambda a: -a[1])
    neighbors = candidates.map(lambda x: x[0][1])

    return neighbors.take(n)
```

Results:

In this section, we demonstrate the results of 10 recommended papers for a randomly selected paper. These results are first outputted by the PMID for each paper and then based on the PMID, we extract the title and tags of each paper for the title/abstract-based and tags-based similarities, respectively.

Title/Abstract-based Similarity:

Neighbors based on title and abstract = ['27999001', '27999914', '27999962', '27999678', '27999812', '27999842', '27999039', '27999105', '27999104', '27999311']

Main paper title: 'Glucose Metabolism After Gastric Banding and Gastric Bypass in Individuals With Type 2 Diabetes: Weight Loss Effect.'
Titles of recommended papers:

1- 'Glucose Metabolism After Gastric Banding and Gastric Bypass in Individuals With Type 2 Diabetes: Weight Loss Effect.'
2- 'Laparoscopic sentinel node navigation surgery for early gastric cancer: a prospective multicenter trial.'
3- 'Can lymphovascular invasion be predicted by preoperative multiphasic dynamic CT in patients with advanced gastric cancer?'
4- 'POU4F3 Gene Causes Autosomal Dominant Hearing Loss.'
5- 'Glucose >200\u2009mg/dL during Continuous Glucose Monitoring Identifies Adult Patients at Risk for Development of Cystic Fibrosis Related Diabetes.'
6- 'Effect of a High-Protein Energy-Restricted Diet Combined with Resistance Training on Metabolic Profile in Older Individuals with Metabolic Impairments.'
7- 'Population pharmacokinetics of cefazolin before, during and after cardiopulmonary bypass to optimize dosing regimens for children undergoing cardiac surgery.'
8- 'Insulin Regulates Astrocytic Glucose Handling Through Cooperation With IGF-I.'
9- 'Heparanase Overexpression Induces Glucagon Resistance and Protects Animals From Chemically Induced Diabetes.'
10- 'Exercise and Beta-Glucan Consumption (Saccharomyces cerevisiae) Improve the Metabolic Profile and Reduce the Atherogenic Index in Type 2 Diabetic Rats (HFD/STZ).'

It should be noted that this similarity is calculated based on both title and abstracts of the papers. Therefore, just by looking at the titles one may not be able to observe the full similarity. Moreover, the method used here is based on bags of words. So the method ignores context and order of the words and also the synonyms are not counted in the similarity evaluations.

Tags-based Similarity:

Neighbors based on tags = ['27999001', '27999351', '27999104', '27999135', '27999002', '27999873', '27999311', '27999106', '27999865', '27999950']

Main paper tags: ['Adult', 'Bariatric Surgery', 'Diabetes Mellitus, Type 2', 'Female', 'Gastric Bypass', 'Glucagon-Like Peptide 1', 'Glucose', 'Humans', 'Incretins', 'Insulin Resistance', 'Longitudinal Studies', 'Male', 'Middle Aged', 'Obesity', 'Postoperative Period', 'Prospective Studies', 'Sweetening Agents', 'Weight Loss']

Tags for recommended papers:
1- ['Adult', 'Bariatric Surgery', 'Diabetes Mellitus, Type 2', 'Female', 'Gastric Bypass', 'Glucagon-Like Peptide 1', 'Glucose', 'Humans', 'Incretins', 'Insulin Resistance', 'Longitudinal Studies', 'Male', 'Middle Aged', 'Obesity', 'Postoperative Period', 'Prospective Studies', 'Sweetening Agents', 'Weight Loss']

2- ['Coronary Artery Disease', 'Diabetes Complications', 'Diabetes Mellitus, Type 1', 'Diabetes Mellitus, Type 2', 'Diabetic Cardiomyopathies', 'Glycated Hemoglobin A', 'Heart Failure', 'Humans']

3- ['Animals', 'Diabetes Mellitus, Experimental', 'Fibroblast Growth Factors', 'Glucagon', 'Glucagon-Like Peptide 1', 'Glucuronidase', 'Hyperglycemia', 'Insulin', 'Islets of Langerhans', 'Male', 'Mice', 'Mice, Inbred C57BL', 'Mice, Transgenic', 'Streptozocin']

4- ['Diabetes Mellitus, Type 2', 'Humans', 'Hypoglycemic Agents', 'Insulin Resistance', 'Prediabetic State', 'Randomized Controlled Trials as Topic', 'Secondary Prevention', 'Stroke', 'Thiazolidinediones']

5- ['Adolescent', 'Adult', 'Case-Control Studies', 'Colombia', 'Diabetes Mellitus, Type 2', 'Fatty Acids, Volatile', 'Feces', 'Female', 'Gastrointestinal Microbiome', 'Humans', 'Hypoglycemic Agents', 'Male', 'Metformin', 'Middle Aged', 'Mucins', 'RNA, Ribosomal, 16S', 'Verrucomicrobia']

6- ['Adolescent', 'Adult', 'Area Under Curve', 'Child', 'Diabetes Mellitus, Type 2', 'Dipeptidyl-Peptidase IV Inhibitors', 'Female', 'Half-Life', 'Humans', 'Hypoglycemic Agents', 'Male', 'Piperidines', 'Uracil', 'Young Adult']

7- ['Animals', 'Atherosclerosis', 'Biomarkers', 'Blood Glucose', 'Combined Modality Therapy', 'Diabetes Mellitus, Experimental', 'Diabetes Mellitus, Type 2', 'Diabetic Angiopathies', 'Diabetic Nephropathies', 'Diet, High-Fat', 'Dietary Fiber', 'Dietary Supplements', 'Exercise Therapy', 'Glycated Hemoglobin A', 'Lipids', 'Male', 'Rats, Wistar', 'Saccharomyces cerevisiae', 'Streptozocin', 'beta-Glucans']

8- ['Animals', 'CRISPR-Cas Systems', 'Cells, Cultured', 'Diabetes Mellitus, Type 1', 'Female', 'Immunity, Innate', 'Interferon Type I', 'Male', 'Parvovirus', 'Rats', 'Real-Time Polymerase Chain Reaction', 'Receptor, Interferon alpha-beta', 'Reverse Transcriptase Polymerase Chain Reaction']

9- ['Aged', 'California', 'Cohort Studies', 'Comorbidity', 'Female', 'Florida', 'Humans', 'Kidney Calculi',

'Length of Stay', 'Male', 'Middle Aged', 'Mortality', 'Nephrolithotomy, Percutaneous', 'Paralysis', 'Perioperative Period', 'Postoperative Complications', 'Sepsis', 'Spinal Cord Injuries']

10- ['Aged', 'Aging', 'China', 'Chronic Disease', 'Cross-Sectional Studies', 'Diabetes Mellitus', 'Dyslipidemias', 'Female', 'Humans', 'Hypertension', 'Male', 'Middle Aged', 'Noncommunicable Diseases', 'Obesity', 'Osteoporosis', 'Prevalence', 'Risk Factors', 'Surveys and Questionnaires']

References:

https://spark.apache.org/documentation.html

https://docs.databricks.com