2017 Big Data Course for Computational Medicine Datathon

Fei Wang. Assistant Professor. few2001@med.cornell.edu
Xi Zhang. Postdoc Research Associate. xiz2016@med.cornell.edu
Yongjun Zhu. Postdoc Research Associate. yoz2008@med.cornell.edu

Division of Health Informatics
Department of Healthcare Policy and Research
Weill Cornell Medicine

Sessions

- 8am-9:45am
- 10am-12pm
- 1pm-2pm

- 2-3pm Presentation
  - Each team will give a 10min spotlight presentation about what you have achieved

Goal

- Practice the computational tools for analyzing healthcare data
  - Morning – Analyzing structured data
    - Data preprocessing (concatenation, imputation)
    - Prediction and feature selection
    - Clustering
  - Afternoon – Analyzing unstructured data
    - Pubmed
    - Semantic analysis
    - Automatic verification of the features selected from the morning session

ANACONDA
Powered by Continuum Analytics®

https://www.continuum.io/downloads

Datathon (Part I)

- Required Python libraries
  - LIBLINEAR
    - **LIBLINEAR** is a **linear** classifier for samples
    - It supports:
      a) L2-regularized classifiers
      b) L2-loss linear SVM, L1-loss linear SVM, and logistic regression (LR)
      c) L1-regularized classifiers (after version 1.4)
      d) L2-loss linear SVM and logistic regression (LR)
      e) L2-regularized support vector regression (after version 1.9)
      f) L2-loss linear SVR and L1-loss linear SVR.
    - we use Python interface
  - Numpy, Pandas, and Sci-kit learn
    - A set of data science libraries used for data analysis and statistics computation.

These libraries should be installed before tomorrow's datathon.

Datathon (Part I)

- Data files

32 tables

Demographics

patient_demo.csv

MDS_UPDRS_Part_I.csv
MDS_UPDRS_Part_I__Patient_Questionnaire.csv
MDS_UPDRS_Part_II__Patient_Questionnaire.csv
MDS_UPDRS_Part_III__Post_Dose_.csv Motor
MDS_UPDRS_Part_IV.csv

Benton_Judgment_of_Line_Orientation.csv
Epworth_Sleepiness_Scale.csv
Geriatric_Depression_Scale__Short_.csv
Hopkins_Verbal_Learning_Test.csv
Letter_-_Number_Sequencing__PD_.csv
Montreal_Cognitive_Assessment__MoCA_.csv
University_of_Pennsylvania_Smell_ID_Test.csv
QUIP_Current_Short.csv
REM_Sleep_Disorder_Questionnaire.csv
SCOPA-AUT.csv
Semantic_Fluency.csv
State-Trait_Anxiety_Inventory.csv
Symbol_Digit_Modalities.csv
Cognitive_Categorization.csv
Non-motor

Biospecimen
Lumbar_Puncture_Sample_Collection.csv
Laboratory_Procedures.csv
Blood_Chemistry___Hematology.csv
Biospecimen_Analysis_Results.csv
Genetic_Testing_Results.csv
Clinical_Labs.csv
DNA_Sample_Collection.csv
Whole_Blood_Sample_Collection.csv

DaTscan_Imaging.csv
DaTscan_Striatal_Binding_Ratio_Results.csv
Magnetic_Resonance_Imaging.csv
Use_of_PD_Medication.csv
Imaging

Datathon (Part I)

- Overview of tasks
  - Data Manipulation
    a) Record Concatenation
    b) Feature Combination
  - Case/Control Classification: L1-regularized logistic regression. (2-class classification to distinguish patients and health controls).
  - MoCA Score Prediction: linear regression (range of continuous scores is 1-31).
  - H&Y scale prediction: L1-regularized support vector classification. (multiple-class classification, discrete scale 0-5).
  - Patient Clustering: k-means; PCA

Datathon (Part I)

- Task 1: Data Manipulation

- There are various Parkinson's disease progression markers and assessments including:

a) demographics

b) motor

c) non-motor

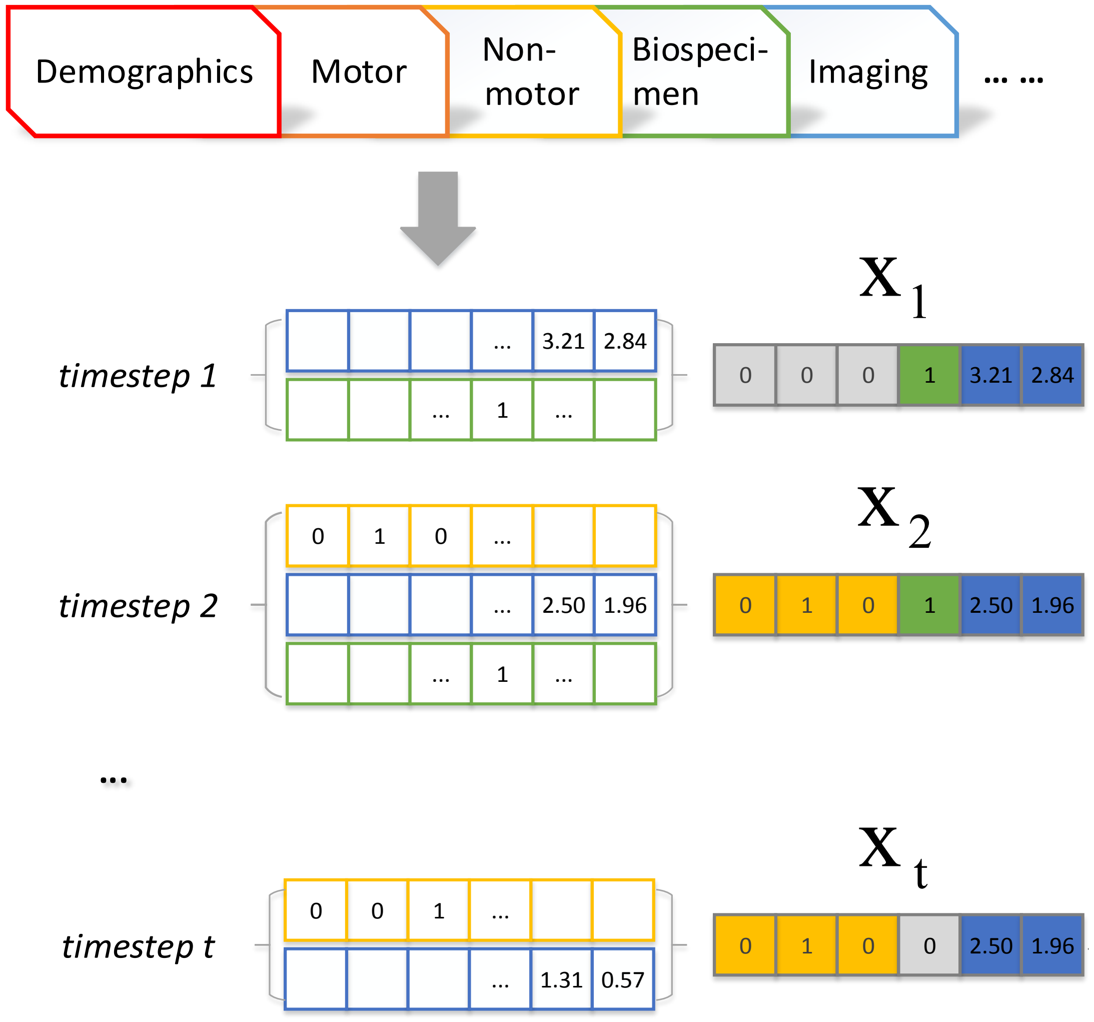
d) biospecimen

e) imaging

Concatenating the multi-source data according timestamps; conducting imputation to fill missing data.

```
patient_array = dict()
feat_array = np.array([n_timestamp,
n_feature], dtype='float32')
patient_array[pid] = feat_array
```

For Each Patient: Store Method



Datathon (Part I)

- Task 2: Parkinson's Disease Prediction

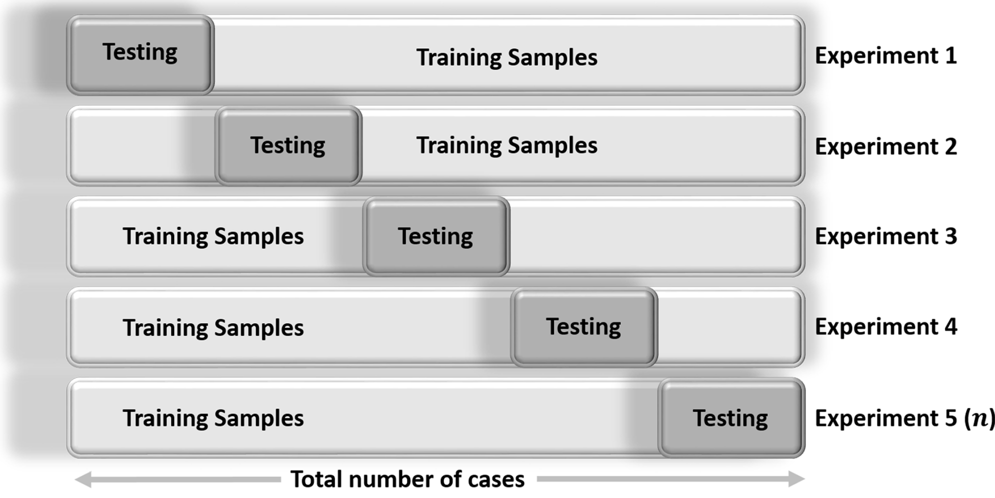
L1-regularized LR solves the following unconstrained optimization problem:

$$\min_{\boldsymbol{w}} \quad \|\boldsymbol{w}\|_1 + C \sum_{i=1}^{l} \log(1 + e^{-y_i \boldsymbol{w}^T \boldsymbol{x}_i}).$$

where $\|\cdot\|_1$ denotes the 1-norm.

```
from liblinear import *
from liblinearutil import *
### model training: X stores data; y stores label
model = train(y, X, options) # options = '-s 6'
### prediction: pred_lb is predicted label
pred_lb, _, _ = predict(y, X, model)
```

5-fold cross validation



Datathon (Part I)

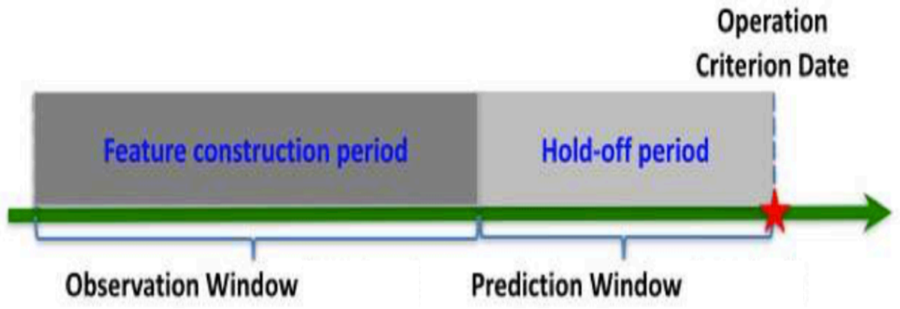
- Task 3: H&Y scale prediction

L1 regularization generates a sparse solution $\boldsymbol{w}$. L1-regularized L2-loss SVC solves the following primal problem:

$$\min_{\boldsymbol{w}} \quad \|\boldsymbol{w}\|_1 + C\sum_{i=1}^{l}(\max(0, 1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i))^2.$$

where $\|\cdot\|_1$ denotes the 1-norm.

```
from liblinear import *
from liblinearutil import *
### model training: X stores data; y stores label
model = train(y, X, options) # options = '-s 5'
### prediction: pred_lb is predicted label
pred_lb, _, _ = predict(y, X, model)
```

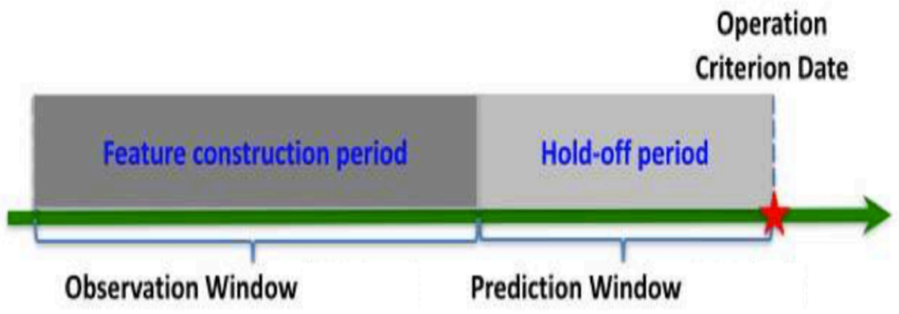


Datathon (Part I)

- Task 4: MoCA Score Prediction

  Linear regression

$$\min_W \frac{1}{2}\sum_i \|y_i - Wx_i\|_2^2 + \lambda||W||_1$$



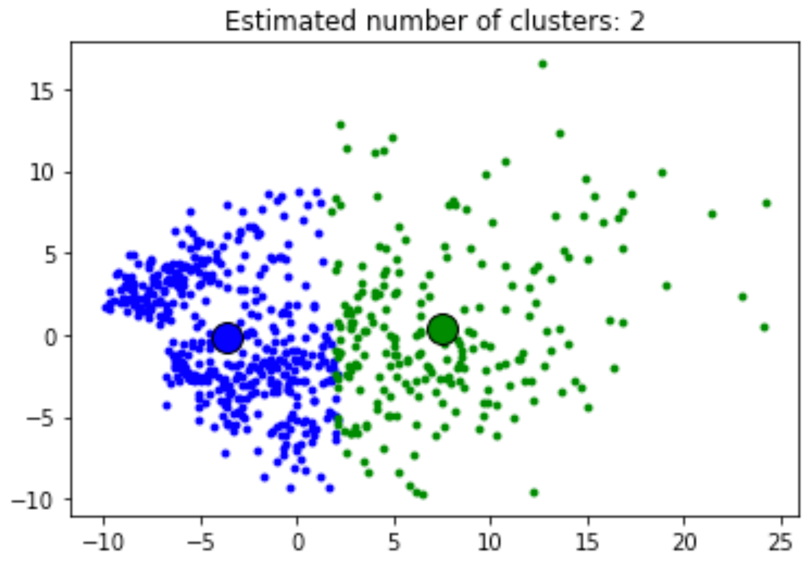
```
from sklearn import linear_model

regr = linear_model.LinearRegression()
### model training: X stores data; y stores label; pred_lb is predicted label
model = self.regr.fit(X, y)
pred_lb = regr.predict(self.X)
```

Datathon (Part I)

- Task 5: Patient Clustering

  k-means:

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2$$



```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
### model training: X stores data; y stores label;
reduced_data = PCA(n_components=32).fit_transform(X) # dimension reduction
k_means = KMeans(init='k-means++', n_clusters=2, n_init=10) # k-means
k_means.fit(reduced_data)
### pred_lb is predicted label
pred_lb = k_means.labels_
```

Datathon (Part II)

- Required Python libraries
  - Biopython
    - Biopython is a set of freely available tools for biological computation written in Python.
    - Biopython will be used for downloading bibliographic records from PubMed.
  - Natural Language Toolkit (NLTK)
    - A suite of libraries for natural language processing (NLP) for English written in Python.
    - NLTK will be used for tokenizing abstracts of bibliographic records.
  - gensim
    - An open-source vector space modeling and topic modeling toolkit written in Python.
    - gensim will be used for generating various models of document representation.
  - Matplotlib, Numpy, Pandas, and Sci-kit learn
    - A set of data science libraries used for data analysis and visualization.

These libraries should be installed before tomorrow's datathon.

Datathon (Part II)

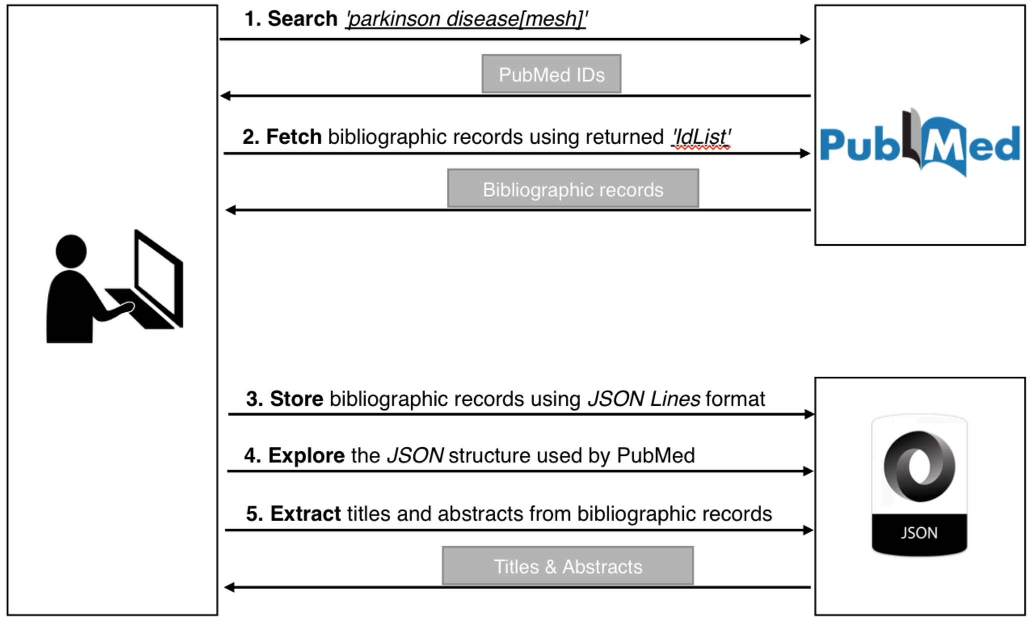
- Data files
    - Bibliographic records
        - Obtaining bibliographic records is a part of task 1.
    - SNIP2015.txt
        - A list of journal names and their SNIP (source normalized impact factor) of 2015.
        - Will be used in task 2.
    - pd_feature_by_predict_model.csv
        - A list of important feature terms and their importance scores identified in the morning session.

Datathon (Part II)

- Overview of the tasks
  - The goal is to explore the research landscape of Parkinson’s disease and the experimental results obtained in the morning session by leveraging knowledge in scientific literature.
  - Metadata of PubMed articles will be used to explore the research landscape
  - Abstracts of the articles will be used as the reference to explore important feature terms identified in the morning session.

Datathon (Part II)

- Task 1: Downloading articles on "Parkinson's disease" and extracting titles and abstracts.
  - Task 1 has two sub-tasks. The first one is to search and download bibliographic records from PubMed. Parkinson's disease is represented as '*parkinson disease[mesh]',* which will be used as the query to be sent to PubMed.
  - The second task is to implement functions to extract titles and abstracts from the downloaded bibliographic records, which will be used in the subsequent tasks.


1. Search *'parkinson disease[mesh]'*
PubMed IDs
2. Fetch bibliographic records using returned *'IdList'*
Bibliographic records
PubMed
3. Store bibliographic records using *JSON Lines* format
4. Explore the *JSON* structure used by PubMed
5. Extract titles and abstracts from bibliographic records
Titles & Abstracts
JSON

Datathon (Part II)

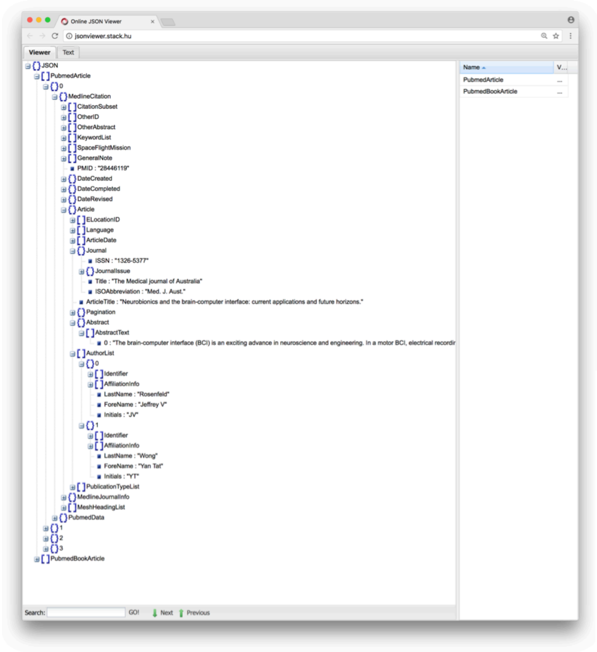
- Task 1: sample codes

```
from Bio import Entrez
from nltk.tokenzie import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
# retrieving pubmed ids
def search(query_term):
    handle = Entrez.esearch(db=‘pubmed’, term=query_term)
    id_list = Entrez.read(handle)[‘IdList’]
# downloading bib records
def fetch(id_list):
    ids = ‘,’.join(id_list)
    handle = Entrez.efetch(db=‘pubmed’, retmode=‘xml’, id=ids)
    results = Entrez.read(handle)
# extracting abstracts and tokenize abstracts
abstract = bib_record['MedlineCitation']['Article']['Abstract']['AbstractText'][0].strip()
sentences = [sent_tokenize(abstract)]
tokens = [word_tokenize(sentence) for sentence in sentences]
```

Datathon (Part II)

- Task 2. Understanding the research landscape of the field (i.e., Parkinson's disease)
  - Task 2 comprises various visualization subtasks using metadata of downloaded bibliographic records.
  - JSON structures need to be understood first in order to extract metadata and use them to make visualizations.

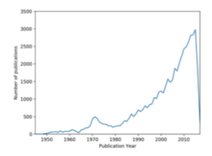
http://jsonviewer.stack.hu/

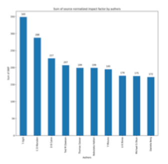


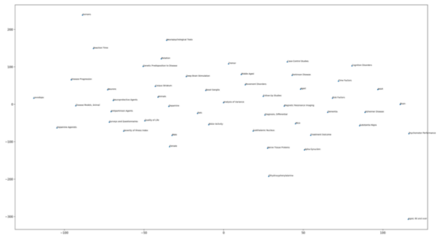
```
{} JSON
  [] PubmedArticle
    {} 0
      {} MedlineCitation
        [] CitationSubset
        [] OtherID
        [] OtherAbstract
        [] KeywordList
        [] SpaceFlightMission
        [] GeneralNote
          PMID : "28446119"
        {} DateCreated
        {} DateCompleted
        {} DateRevised
        {} Article
          [] ELocationID
          [] Language
          [] ArticleDate
          {} Journal
              ISSN : "1326-5377"
            {} JournalIssue
              Title : "The Medical journal of Australia"
              ISOAbbreviation : "Med. J. Aust."
            ArticleTitle : "Neurobionics and the brain-computer interface: current applications and future horizons."
          {} Pagination
          {} Abstract
            [] AbstractText
                0 : "The brain-computer interface (BCI) is an exciting advance in neuroscience and engineering. In a motor BCI, electrical recordir
          [] AuthorList
            {} 0
              [] Identifier
              [] AffiliationInfo
                LastName : "Rosenfeld"
                ForeName : "Jeffrey V"
                Initials : "JV"
            {} 1
              [] Identifier
              [] AffiliationInfo
                LastName : "Wong"
                ForeName : "Yan Tat"
                Initials : "YT"
          [] PublicationTypeList
        {} MedlineJournalInfo
        [] MeshHeadingList
      {} PubmedData
    {} 1
    {} 2
    {} 3
  [] PubmedBookArticle
```

Exploration & Visualization

1. Explore the JSON structure and extract important metadata to create visualizations.
2. Key metadata need to be extracted include publication year, author info, journal info, affiliation info, and MeSH terms.
3. Several visualizations are created to identify the landscape of the filed such as active researchers, journals, institutions, etc.
4. MeSH terms assigned to the articles of Parkinson's disease are visualized.







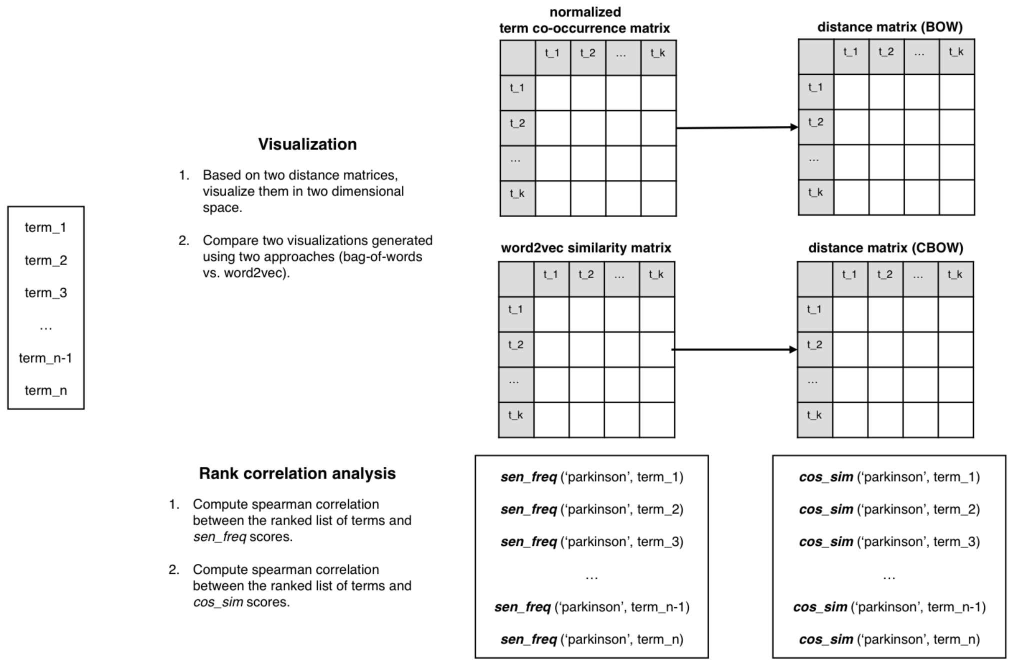
Datathon (Part II)

- Task 2: sample codes

```
from matplotlib.pyplot as plt
# retrieving metadata from the JSON file
year_articles = dict()
try:
    pub_year = bib_record['MedlineCitation']['Article']['Journal']['JournalIssue']['PubDate']['Year']
    if pub_year in year_articles:
        year_articles[pub_year] = year_articles[pub_year] + 1
    else:
        year_articles[pub_year] = 1
except:
    pass
# sorting
sorted_year_articles = sorted(year_articles.items(), key=operator.itemgetter(0), reverse=False)
# plot number of articles by year
x = list()
y = list()
for item in sorted_year_articles:
    x.append(item[0])
    y.append(item[1])
plt.plot(x, y)
plt.xlabel('Publication Year')
plt.ylabel('Number of publications')
plt.axis(( 1945, 2017, 0, 3500))
plt.show()
```

Datathon (Part II)

- Task 3. Exploring relative importance of terms.
    - Task 3 has two sub-tasks. The first one is to visualize feature terms in 2D space using two different similarity measures: term co-occurrence and word2vec similarity.
    - The second task is to compare the ranking obtained in the morning session with rankings derived from term co-occurrence and word2vec similarity. One ranking is derived from co-occurrence values between the feature terms and the term ‘parkinson’. The other ranking is derived from cosine similarities between the feature terms and the term ‘parkinson’.



Visualization

1. Based on two distance matrices, visualize them in two dimensional space.
2. Compare two visualizations generated using two approaches (bag-of-words vs. word2vec).

Rank correlation analysis

1. Compute spearman correlation between the ranked list of terms and *sen_freq* scores.
2. Compute spearman correlation between the ranked list of terms and *cos_sim* scores.

term_1
term_2
term_3
…
term_n-1
term_n
normalized term co-occurrence matrix
distance matrix (BOW)
word2vec similarity matrix
distance matrix (CBOW)
t_1
t_2
…
t_k
sen_freq (‘parkinson’, term_1)
sen_freq (‘parkinson’, term_2)
sen_freq (‘parkinson’, term_3)
…
sen_freq (‘parkinson’, term_n-1)
sen_freq (‘parkinson’, term_n)
cos_sim (‘parkinson’, term_1)
cos_sim (‘parkinson’, term_2)
cos_sim (‘parkinson’, term_3)
…
cos_sim (‘parkinson’, term_n-1)
cos_sim (‘parkinson’, term_n)

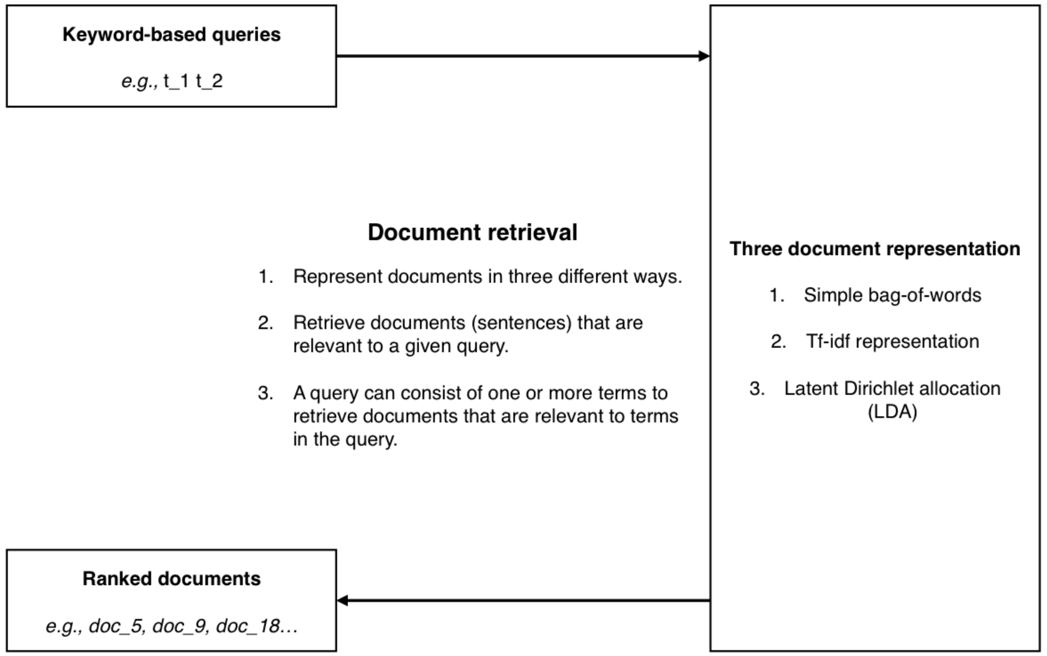
Datathon (Part II)

- Task 3: sample codes

```
import gensim
import task1
file_path = Your JSON file path
# train word2vec model by using
def train_word2vec_unigram():
    # word2vec’s input is a list of list of tokens. Use method implemented in task 1
    docs = task1.get_articles_as_lists_of_tokens(file_path)
    model = gensim.models.Word2Vec(docs)
    word2vec_model_path = 'data/models/word2vec_model'
    output_file = open(word2vec_model_path, 'wb')
    model.save(output_file)
# visualize word2vec
# distance_word2vec is a matrix of feature terms filled with distances among feature terms
# feature_terms is a list of feature terms provided as a data file
tsne = TSNE(n_components=2, random_state=1, metric='precomputed')
X_tsne = tsne.fit_transform(distance_word2vec)
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], s=10)
for label, x, y in zip(feature_terms, X_tsne[:, 0], X_tsne[:, 1]):
    plt.annotate(label, xy=(x, y), xytext=(0, 0), fontsize=7, textcoords='offset points')
plt.show()
```

Datathon (Part II)

- Task 4: Indexing and retrieving the most relevant documents.
  - In task 4, two models of document representation TF-IDF and LDA will be generated. In the TF-IDF model, rows are documents and columns are words included in the documents. In the LDA model, rows are documents and columns are topics that are automatically generated by the algorithm.
  - Keyword-based queries will be sent to the models and ranked documents will be returned. By manually reviewing the returned documents, relationships between keywords in the query can be identified.



Keyword-based queries
e.g., t_1 t_2
Document retrieval
1. Represent documents in three different ways.
2. Retrieve documents (sentences) that are relevant to a given query.
3. A query can consist of one or more terms to retrieve documents that are relevant to terms in the query.
Three document representation
1. Simple bag-of-words
2. Tf-idf representation
3. Latent Dirichlet allocation (LDA)
Ranked documents
e.g., doc_5, doc_9, doc_18…

Datathon (Part II)

- Task 4: sample codes

```
from gensim import corpora, models, similariteis
import task1
file_path = Your JSON file path
# get documents (use a function implemented in task 1)
docs = task1.get_sentences_as_lists_of_tokens(file_path)
# construct a dictionary
dictionary = corpora.Dictionary(docs)
output_file = open('data/models/dictionary.dict', 'wb')
dictionary.save(output_file)
# construct a tf-idf model
corpus = [dictionary.doc2bow(doc) for doc in docs]
tfidf_model = models.TfidfModel(corpus, normalize=True)
tfidf_corpus = tfidf_model[corpus]
tfidf_model.save('data/models/docs_tfidf.model')
corpora.MmCorpus.serialize('data/models/docs_tfidf.mm', tfidf_corpus)
# create indexes
tfidf_index = similarities.Similarity('data/models/docs_tfidf_index', tfidf_corpus, num_features=tfidf_corpus.num_terms)
# query
def get_sentences_tfidf(query, topn=10):
    # load the dictionary
    dictionary = corpora.Dictionary.load('data/models/dictionary.dict')
    return_list = list()
    tfidf_model = models.TfidfModel.load('data/models/docs_tfidf.model')
    tfidf_index = similarities.Similarity.load('data/models/docs_tfidf_index.index')
    query_bow = dictionary.doc2bow(query.split())
    query_tfidf = tfidf_model[query_bow]
    tfidf_index.num_best = topn
    results = tfidf_index[query_tfidf]
```

enjoy!