Milestone 2 Report

Chengming Xie (cx2234), Hongshan Li (hl3353)

Topic : ***Medical Insurance Fraud Detection*** *(Group 20)*

Date: 3/19/2020

1. Abstract of our project

Machine learning for data-driven diagnosis has been actively studied in medicine to provide better healthcare. Supporting analysis of a patient cohort similar to a patient under treatment will help clinicians to make better decisions and also help the insurance providers to detect the possible fraud insurance claim. However, such analysis is difficult due to the characteristics of medical records: high dimensionality, irregularity in time,and sparsity. To address this challenge, we deal with these medical records as event sequences in natural language applications and calculate the similarity of medical records between patients. We employed event sequence embeddings (***Skip-gram based Word2Vec***) , and sequence alignment (***Dynamic time warp*** algorithm).

2. Overview of Milestone 2 Progress

In this project we used detailed and comprehensive patients' data from MIMIC-III dataset which contains 26 tables and medical records for 61000+ patients. In the progress of milestone 1, we have chosen two disease categories of patients to begin with our project (digestive system with icd-9 code beginning with 500 and injury & poisoning related disease beginning with 800.) We have finished preprocessing our data to a json format dictionary which contains all medical records for each patient ID (i.e. Diagnoses, Procedures and Prescriptions.)

In milestone 2 progress, we started training our skip-gram based Word2Vec model and Dynamic Time Warping Algorithms to our processed events sequence data for patients who fell into one specific disease category. During the implementation of our models, we utilized the techniques of ***Spark Clusters*** to boost our time efficiency by a large amount of times and resolved the question of the expansive memory requirement of our skip-gram neural network (~ 20 million event pairs in input and output layer).

So far, we finished producing vector representations (of length 200) for each unique medical event code and aligned each patient's event sequence data of different length, based on which we computed the cosine similarity (or Euclidean distance) between each patient.

In the future of milestone 3, based on the distance metrics we have already created, we will be further able to detect the ***anomaly*** among patients in the specific disease category. The results may be used to investigate whether the medical records in this category involved in the medical fraud.

Also, if we still have time left, we will train a ***LSTM recurrent neural network*** to predict the next medical event which would possibly happen given the historical health records. Such research direction could be another way to help insurers to prevent insurance fraud in advance.

3. Implementation and progress of project

In our milestone 2 implementation, we leveraged the power of cloud computing platform (GCP). We aimed to train our models (specifically ***Skip-gram based Word2Vec Model***, ***Dynamic Time Warping***) on the preprocessed data we have already cleaned in milestone 1. We preprocessed the medical events sequence data for patients whose disease category falls into digestive systems and injury & poisoning (i.e. ICD-9 code in 520-579, 800 - 999). And we trained our models separately in these two categories of patients' event sequence data (example of our cleaned data is shown below.)

```
pd.DataFrame(result_dict[68])
```

	ADMITTIME	ICD_9	SEQ_NUM
0	2173-12-15 16:16:00	042	1.0
1	2173-12-15 16:16:00	486	2.0
2	2173-12-15 16:16:00	4254	3.0
3	2173-12-15 16:16:00	42820	4.0
4	2173-12-15 16:16:00	4280	5.0
5	2173-12-15 16:16:00	5849	6.0
6	2173-12-15 16:16:00	5859	7.0
7	2173-12-15 16:16:00	2639	8.0
8	2173-12-15 16:16:00	2848	9.0
9	2173-12-15 16:16:00	2761	10.0
10	2173-12-15 16:16:00	2859	11.0
11	2173-12-15 16:16:00	V141	12.0
12	2173-12-15 16:16:00	V071	13.0
13	2174-01-04 22:21:00	042	1.0

```
input_list = []
for patient_id in respira_dict:
    a = pd.DataFrame.from_dict(respira_dict[patient_id])
    a['Time'] = pd.to_datetime(a['ADMITTIME'],format = '%Y-%m-%d %H:%M:%S')
    patient = a.iloc[:,1:3]
    patient = patient.sort_values(by='Time')
    array = patient.values
    for i in range(len(array)):
        event = array[i]
        time = event[1]
        for j in np.arange(i+1,len(array)):
            related_event = array[j]
            compare_time = related_event[1]
            if compare_time - time <= Timedelta('7 days 00:00:00'):
                input_list.append([event[0],related_event[0]])
            else:
                break
```

```
input_layer
```

```
array([['ATEN25', 'GABA100'],
       ['ATEN25', 'ATOR20'],
       ['ATEN25', 'HCTZ25'],
       ...,
       ['V427', 'V1046'],
       ['V427', 'V1007'],
       ['V1046', 'V1007']],
      dtype='<U17')
```

```
len(input_layer)
```

```
18847875
```

Figure I. Event Sequence for patient id = 68 & input layer of three layer neural network skip-gram model (sample output and code)

- To compute the co-occurrence likelihood (i.e similarity) for two medical events (i.e. Diagnoses, Procedures, prescriptions), we implemented skip-gram model and built three layer neural network to compute distributed vector representation of medical events which are presented in string formats (i.e. ICD-9 code, formulated drug code.) The training objective of our skip-gram model is to learn event vector representations (of size 200) that aimed to predict its context in our defined time window (Results contained in next section.)
- Mathematically, given a sequence of training events, $w_1, w_2, ..., w_n$ (e.g. all medical events of patients who have digestive system disease,) the objective is to maximize the average log-likelihood $1/T \{\sum_{t=1}^{n} \sum_{j=-k}^{k} log\, p(w_{t+j} \mid w_t)\}$,where k is the size of the time window. In our model implementation, we chose 7 days as time window (i.e. w_{t+j} is any medical event happening in seven days before or after w_t.)
- The probability of correctly predicting event w_t given event w_j is determined by the softmax model, which is $p(w_{t+j} \mid w_t) = exp(u^T{}_{w_i} v_{w_j}) / \sum_{l=1}^{V} exp(u^T{}_{\hat{l}} v_{w_j})$, where u_w and v_w is are vector representations of w as event and its context event respectively, and V is the vocabulary containing all events code.
- After fetching all vector representations for all medical events of all patients in a specific disease category, we then implemented a Dynamic programming algorithm (Dynamic Time Warping) to align events sequence data with different length for each patient based on the cosine similarity between two medical events (results of event embedding). We can then compute the dtw distance (accumulate cost of cosine similarity score along the optimized warping path) between two patients' events sequence data (i.e. achieved by DTW package.)

Figure II. DTW for two timestamped sequence data

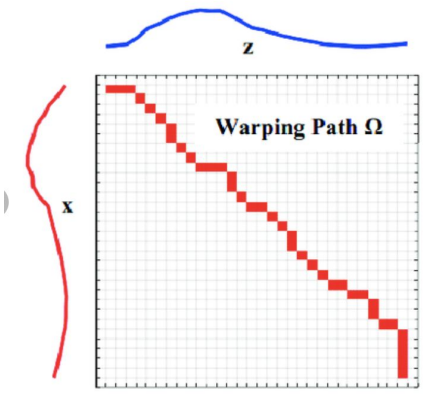

Figure III: similarity score list for a unique patient (id = 4096)

```
patient_similarity
{4096: [-30.447746135219138,
  -69.522483917898001,
  -230.98754365802151,
  -48.652960730896595,
  -20.152156988404869,
  -97.470908904545468,
  -48.702188512724184,
  -14.791020178854907,
  -103.87787434936551,
  -51.584655492004956,
  -88.553040075237362,
  -17.493200179485921,
  -86.796576393784449,
  -84.283417617631429,
  -33.257395144759009,
  -53.267140489526525,
  -59.584418142005404,
  -62.110173014373643,
  -72.164532756471388,
```

4. Challenges and State-of-art solutions

Our ***first challenge*** is data cleaning. Due to the characteristics of medical records: high dimensionality, irregularity in time, and sparsity, we spend a lot of time dealing with these raw data. To solve this problem, it took time to understand the ER between tables and figure out a way to transform these 26 tables into a format in which selected features can be used for our models. We set the time window to deal with the temporal relationship between medical records.

Moreover, although with the help of the GCP virtual machine, we faced a big challenge for model training during the skip-gram implementation. For instance, when creating one-hot encoding vectors for input layer in the skip-gram model, we are suffering from memory leak since we had over ***10,000,000+*** events pairs which were too many to generate one hot vectors for each of them; Also, when building neural network model, the model required a large amount of memory in our virtual machine. More specifically, we have tried 100+ GB extended memory VM but it still failed in the model training process and the kernel still consistently broke.

Thus, we leveraged the resilient distributed dataset (***RDD***) and ***Spark*** programming technique in GCP Dataproc ***Clusters*** by creating one master node and two worker nodes. Using Spark in Scala kernel allowed us to better scale the Word2Vec algorithms while enhancing the time efficiency by dozens of times. We successfully implemented the Word2Vec model and could query vector representations for each medical event code. We can also order the most relevant events for each medical event by the cosine similarity of their vectors representations (see Figure IV:IX)

```
val ds = model.findSynonyms("aten25", 20).select("word")
```

```
ds = [word: string]
```

```
[word: string]
```

```
ds.show
```

```
+-------+
|   word|
+-------+
| topr50|
| hctz25|
|  42769|
|  49320|
| 4823.0|
| 4881.0|
|  71590|
|   4589|
|   4439|
|   2720|
|   2765|
|   7048|
|   5180|
|  v4582|
| cil150|
| ator20|
|    566|
|  53081|
| lev500|
|gaba100|
+-------+
```

Figure IV: top 20 similar medical events to Aten 25

word	similarity
topr50	0.6053675413131714
hctz25	0.5762994289398193
42769	0.5510246157646179
49320	0.5212512016296387
4823.0	0.5142900347709656
4881.0	0.5130040049552917
71590	0.49900346994400024
4589	0.49728673696517944
4439	0.47791099548339844
2720	0.47099921107292175
2765	0.46940693259239197
7048	0.4664931297302246
5180	0.46129298210144043
v4582	0.46122118830680847
cil150	0.4498251974582672
ator20	0.44093233346939087
566	0.4358844459056854
53081	0.4232785105705261
lev500	0.39566564559936523
gaba100	0.38527387380599976

Figure V: Json format of event embedding results (Vector Representation)

```
In [12]: model.getVectors.show
```

word	vector
57410	[0.08606849610805...
3782.0	[0.44093295931816...
mege401	[-0.1348615884780...
macr100	[-0.1039232611656...
29644	[-0.1229314729571...
8876.0	[0.445857435464485...
35781	[-0.1755153983831...
45385	[0.02785602398216...
2760	[0.02956386841833...
63.0	[0.71321177482604...
44023	[0.27470186352729...
515	[0.29041996598243...
28803	[-0.3964917659759...
8052	[-0.1912718117237...
nacl23.4i	[0.25673702359199...
28249	[0.67226463556289...
29284	[0.63775449991226...
4233.0	[0.27437391877174...
metosusp101	[1.10645031929016...
28984	[0.06335902959108...

only showing top 20 rows

Figure VI. Spark rdd dataframe: Most relevant medical event to Aten25. (left)

Figure VII: Event Embeddings for all event codes. (right)

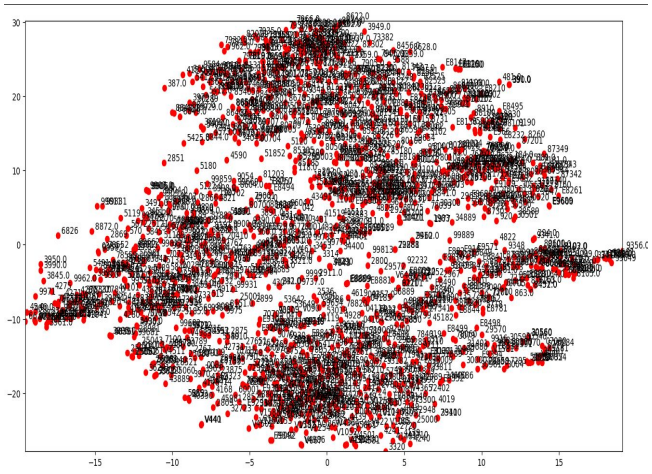

Figure VIII. mapping of 5000+ event code in latent space

```
import org.apache.spark.ml.feature.{Word2Vec, Word2VecModel}

val rdd = sc.textFile("gs://stephxie-6895/listfile4.txt")
val rdd2 = rdd.map(line => line.toLowerCase).
            map(_.split(" ")).
            map(Tuple1.apply).
            toDF("text")

Waiting for a Spark session to start...

rdd = gs://stephxie-6895/listfile4.txt MapPartitionsRDD[1] at textFile
rdd2 = [text: array<string>]

[text: array<string>]

rdd2.take(1)

[aten25, gaba100]
```

Figure IX. screenshot of data input in scala kernel (GCP clusters)

The third challenge we meet is the wide differences of length in the event sequence data. After word embedding, we find our sequence records vary dramatically in length and exact event orders that nonetheless represent similar disease progressions. Multiple patients suffering from the same disease may have doctor visits at different times, they may have symptoms appear or recede at different speeds, and they may have distinct sets of comorbidities. It is necessary to align event sequences based on their implicit semantics.

To solve this problem, we got back to the python module, fetched our results in the storage bucket and implemented dynamic time warping algorithms to align each patient's event sequence data. We retrieved the similar patients based on their DTW distance (accumulated cost of the Path between two patients' event sequences). Sample results of DTW are shown below.

```
distance, path = fastdtw(icd_2080,icd_4579,dist=euclidean)
print(distance)
print(path)

index_a,index_b=zip(*path)

plt.plot(index_a,index_b, color='k', linewidth=2)
plt.savefig('demo_dtw', bbox_inches='tight')
```

```
26.71888577368587
[(0, 0), (1, 1), (2, 1), (3, 2), (4, 3), (5, 4), (6, 5), (7, 6), (8, 7), (9, 8), (10, 9), (11, 10), (12, 11), (13, 11), (14, 1
1), (15, 11), (16, 11), (17, 12), (18, 13), (19, 14), (20, 15)]
```

Figure IX. ***26.71888*** is Euclidean Distance between Patient ID 2000, and Patient ID 4579

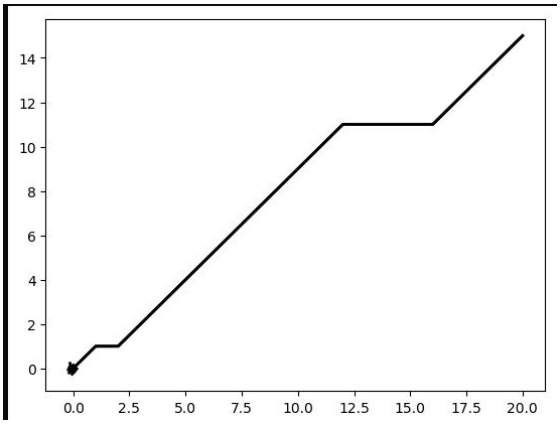

Figure X. Optimal Warping path between two patients above

5. Discussion of Milestone 2 and Exploration of Next Step

Until milestone 2, we have finished event sequence embedding using word2vec, and similar patient retrieval using DTW. We tried several ***new ideas*** here. For example, we compared the effectiveness of two different word2vec algorithms (Glove and Skip-gram) and finally chose implementing Skip-gram in Spark. Unlike traditional word embedding, we set our window size based on the time interval between event data in our sequence embedding.

In the next milestone, we are going to perform the anomaly detection among patients falling in one specific disease category using DTW similarity score and medical events similarity we have computed in milestone 2.

We will be able to start with some machine learning algorithms with the help of the principles of these algorithms (such as Kmeans, Isolation forest, Local Outlier Factor, etc.) while using our custom distance metrics (DTW distance using either Euclidean or Cosine).

Also, if we still have time left, we will begin to implement another research direction on Medical Insurance anomaly detection by predicting the medical event which would happen in the future based on the historical medical events records . We aim to help the insurance issuers to prevent the medical fraud which might happen in the future. We can achieve the task by using Keras or Tensorflow to build LSTM Recurrent Neural Network to forecast the timestamped patients' event sequence data based on our event embedding results in Milestone 2.

Appendix: More screenshots for our codes

If you need detailed code, feel free to contact us.

In [1]:
```
import pandas as pd
import numpy as np
import datetime
from pandas import Timestamp
from pandas import Timedelta
```

In [2]:
```
df = pd.read_csv("/CarePre/ADMISSIONS.csv")
```

In [7]:
```
from os import listdir
from os.path import isfile, join
mypath = "/CarePre"
onlyfiles = [f for f in listdir(mypath) if isfile(join(mypath, f))]
```

In [4]:
```
onlyfiles
```

Out[4]:
```
['SERVICES.csv',
 'CALLOUT.csv',
 'TRANSFERS.csv',
 'CPTEVENTS.csv',
 'D_CPT.csv',
 'D_ICD_PROCEDURES.csv',
 'DATETIMEEVENTS.csv',
 'D_ITEMS.csv',
 'PRESCRIPTIONS.csv',
 'OUTPUTEVENTS.csv',
 'DIAGNOSES_ICD.csv',
 'INPUTEVENTS_CV.csv',
 'PROCEDUREEVENTS_MV.csv',
 'NOTEEVENTS.csv',
 'ADMISSIONS.csv',
 'PATIENTS.csv',
 'CHARTEVENTS.csv',
 'CAREGIVERS.csv',
 'PROCEDURES_ICD.csv',
 'INPUTEVENTS_MV.csv',
 'D_ICD_DIAGNOSES.csv',
 'D_LABITEMS.csv',
 'ICUSTAYS.csv',
```

In [3]:
```
prescriptions = pd.read_csv("/CarePre/PRESCRIPTIONS.csv",low_memory=False)
prescriptions.head()
```

Out[3]:

	ROW_ID	SUBJECT_ID	HADM_ID	ICUSTAY_ID	STARTDATE	ENDDATE	DRUG_TYPE	DRUG	DRUG_NAME_POE	DRUG_NAME_GENERIC
0	2214776	6	107064	NaN	2175-06-11 00:00:00	2175-06-12 00:00:00	MAIN	Tacrolimus	Tacrolimus	Tacrolimus
1	2214775	6	107064	NaN	2175-06-11 00:00:00	2175-06-12 00:00:00	MAIN	Warfarin	Warfarin	Warfarin
2	2215524	6	107064	NaN	2175-06-11 00:00:00	2175-06-12 00:00:00	MAIN	Heparin Sodium	NaN	NaN
3	2216265	6	107064	NaN	2175-06-11 00:00:00	2175-06-12 00:00:00	BASE	D5W	NaN	NaN
4	2214773	6	107064	NaN	2175-06-11 00:00:00	2175-06-12 00:00:00	MAIN	Furosemide	Furosemide	Furosemide

In [4]:
```
patients = pd.read_csv("/CarePre/PATIENTS.csv")
```

In [5]:
```
admissions = pd.read_csv('/CarePre/ADMISSIONS.csv')
admissions
```

Out[5]:

	ROW_ID	SUBJECT_ID	HADM_ID	ADMITTIME	DISCHTIME	DEATHTIME	ADMISSION_TYPE	ADMISSION_LOCATION	DISCHARGE_LOCAT
0	21	22	165315	2196-04-09 12:26:00	2196-04-10 15:54:00	NaN	EMERGENCY	EMERGENCY ROOM ADMIT	DISC-TRAN CANCER/CHLDRN H
1	22	23	152223	2153-09-03 07:15:00	2153-09-08 19:10:00	NaN	ELECTIVE	PHYS REFERRAL/NORMAL DELI	HOME HEALTH CARI

In [13]:
```
diag_icd = pd.read_csv('/CarePre/DIAGNOSES_ICD.csv')
```

In [14]:
```
merge1 = pd.merge(admissions[['SUBJECT_ID','HADM_ID','ADMITTIME','ADMISSION_TYPE']],diag_icd, how = 'left',on = 'HADM_ID')
merge2 = pd.merge(merge1,D_diag_icd[['ICD9_CODE','SHORT_TITLE']],how = 'left',on = 'ICD9_CODE')
```

In [15]:
```
diag_df = merge2.rename(columns = {'SUBJECT_ID_x':'SUBJECT_ID','ICD9_CODE':'ICD_9'}).drop(['SUBJECT_ID_y'],axis = 1)
```

In [16]:
```
merge3 = pd.merge(admissions[['SUBJECT_ID','HADM_ID','ADMITTIME','ADMISSION_TYPE']],procedures,
                  how = 'left',on = 'HADM_ID')
merge4 = pd.merge(merge3,D_procedures_icd[['ICD9_CODE','SHORT_TITLE']],how = 'left',on = 'ICD9_CODE')
```

In [17]:
```
procedure_df = merge4.rename(columns = {'SUBJECT_ID_x':'SUBJECT_ID','ICD9_CODE':'ICD_9'}).drop(['SUBJECT_ID_y'],axis = 1)
```

In [18]:
```
prescriptions_df = prescriptions[['SUBJECT_ID','HADM_ID',
                                  'STARTDATE','FORMULARY_DRUG_CD']].rename(columns = {'STARTDATE':'ADMITTIME'})
```

In [19]:
```
prescriptions_df.head(1)
```

Out[19]:

	SUBJECT_ID	HADM_ID	ADMITTIME	FORMULARY_DRUG_CD
0	6	107064	2175-06-11 00:00:00	TACR1

In [53]:
```
diag_df.head(2)
```

Out[53]:

	SUBJECT_ID	HADM_ID	ADMITTIME	ADMISSION_TYPE	ROW_ID	SEQ_NUM	ICD_9	SHORT_TITLE
0	22	165315	2196-04-09 12:26:00	EMERGENCY	151	1.0	9678	Pois-sedative/hypnot NEC
1	22	165315	2196-04-09 12:26:00	EMERGENCY	152	2.0	9693	Poison-antipsychotic NEC

In [20]:
```
patient_dict = dict()
for index in admissions.SUBJECT_ID.unique():
    patient = diag_df[diag_df['SUBJECT_ID'] == index][['ADMITTIME','ICD_9','SHORT_TITLE','SEQ_NUM']]
    patient_dict[index] = []
    for i in range(len(patient)):
        stay = patient.iloc[i,:].to_dict()
        stay['event_type'] = 'Diagnose'
        patient_dict[index].append(stay)
```

In [27]:
```
patient_dict[2]
```

Out[27]:
```
[{'ADMITTIME': '2138-07-17 19:04:00',
  'ICD_9': 'V3001',
  'SEQ_NUM': 1.0,
  'SHORT_TITLE': 'Single lb in-hosp w cs',
  'event_type': 'Diagnose'},
 {'ADMITTIME': '2138-07-17 19:04:00',
  'ICD_9': 'V053',
  'SEQ_NUM': 2.0,
  'SHORT_TITLE': 'Need prphyl vc vrl hepat',
  'event_type': 'Diagnose'},
 {'ADMITTIME': '2138-07-17 19:04:00',
  'ICD_9': 'V290',
  'SEQ_NUM': 3.0,
  'SHORT_TITLE': 'NB observ suspct infect',
  'event_type': 'Diagnose'}]
```

In [21]:
```
procedure_dict = dict()
for index in admissions.SUBJECT_ID.unique():
    patient = procedure_df[procedure_df['SUBJECT_ID'] == index][['ADMITTIME','ICD_9','SHORT_TITLE','SEQ_NUM']]
    procedure_dict[index] = []
    for i in range(len(patient)):
        stay = patient.iloc[i,:].to_dict()
        stay['event_type'] = 'Procedure'
        procedure_dict[index].append(stay)
```

In [23]:
```
## All features
result_dict = dict()
for index in admissions.SUBJECT_ID.unique():
    result_dict[index] = patient_dict[index]+procedure_dict[index
```

In [47]:
```
pd.DataFrame(result_dict[68])
```

Out[47]:

	ADMITTIME	ICD_9	SEQ_NUM
0	2173-12-15 16:16:00	042	1.0
1	2173-12-15 16:16:00	486	2.0
2	2173-12-15 16:16:00	4254	3.0
3	2173-12-15 16:16:00	42820	4.0
4	2173-12-15 16:16:00	4280	5.0
5	2173-12-15 16:16:00	5849	6.0
6	2173-12-15 16:16:00	5859	7.0
7	2173-12-15 16:16:00	2639	8.0
8	2173-12-15 16:16:00	2848	9.0
9	2173-12-15 16:16:00	2761	10.0
10	2173-12-15 16:16:00	2859	11.0
11	2173-12-15 16:16:00	V141	12.0
12	2173-12-15 16:16:00	V071	13.0

In [24]:
```
respiratory = list(diag_df[diag_df['ICD_9'].isin([str(x) for x in np.arange(520,580)])]['SUBJECT_ID'])
```

In [25]:
```
unique_patient = []

for i in respiratory:
    if i not in unique_patient:
        unique_patient.append(i)
```

In [26]:
```
len(unique_patient)
```

Out[26]:
```
1109
```

In [27]:
```
respira_dict = dict()
for index in unique_patient:
    respira_dict[index] = result_dict[index]
```

In [28]:
```
for patient in respira_dict:
    try:
        for event in respira_dict[patient]:
            event.pop('event_type')
            if event.get('SHORT_TITLE'):
                event.pop('SHORT_TITLE')
            if event.get('FORMULARY_DRUG_CD'):
                event['ICD_9'] = event.pop('FORMULARY_DRUG_CD')
    except:pass
```

In [29]:
```
patient_event = dict()
for patient in respira_dict:
    current = respira_dict[patient]
    current_df = pd.DataFrame.from_dict(current)
    current_df['Time'] = pd.to_datetime(current_df['ADMITTIME'],format = '%Y-%m-%d %H:%M:%S')
    patient_event[patient] = np.array(current_df.sort_values(by='Time')['ICD_9'])
```

In [78]:
```
input_list = []
for patient_id in respira_dict:
    a = pd.DataFrame.from_dict(respira_dict[patient_id])
    a['Time'] = pd.to_datetime(a['ADMITTIME'],format = '%Y-%m-%d %H:%M:%S')
    patient = a.iloc[:,1:3]
    patient = patient.sort_values(by='Time')
    array = patient.values
    for i in range(len(array)):
        event = array[i]
        time = event[1]
        for j in np.arange(i+1,len(array)):
            related_event = array[j]
            compare_time = related_event[1]
            if compare_time - time <= Timedelta('5 days 00:00:00'):
                input_list.append([str(event[0]),str(related_event[0])])
            else:
                break
```

```
import org.apache.spark.ml.feature.{Word2Vec, Word2VecModel}
```

```
val rdd = sc.textFile("gs://stephxie-6895/listfile4.txt")
val rdd2 = rdd.map(line => line.toLowerCase).
           map(_.split(" ")).
           map(Tuple1.apply).
           toDF("text")
```

```
Waiting for a Spark session to start...

rdd = gs://stephxie-6895/listfile4.txt MapPartitionsRDD[1] at textFile
rdd2 = [text: array<string>]

[text: array<string>]
```

```
rdd2.take(1)
```

```
[aten25, gaba100]
```

```
val word2vec = new org.apache.spark.ml.feature.Word2Vec().
    setInputCol("text").
    setOutputCol("features").
    setVectorSize(200).
    setMinCount(0).
    setMaxIter(50)

val model = word2vec.fit(rdd2)

val result = model.transform(rdd2)

val df = model.getVectors.filter($"word" === "aten25")

model.getVectors.show

val ds = model.findSynonyms("aten25", 20).select("word")

ds.show
```

Dynamic Time Warping

In [30]:
```
import json
from sklearn.metrics.pairwise import cosine_similarity
from dtw import dtw
```

In [31]:
```
file = []
for line in open('embedding json_part-00000-5f166835-f8bb-45c0-8549-083ac77275b1-c000.json', 'r'):
    file.append(json.loads(line))
```

In [32]:
```
embed_dict = dict()
for i in range(len(file)):
    cur_word = file[i]['word']
    embed_dict[cur_word] = file[i]['vector']['values']
```

In [36]:
```
def icd_distance(x,y):
    a = np.asarray(embed_dict[x]).reshape(1,-1)
    b = np.asarray(embed_dict[y]).reshape(1,-1)
    return cosine_similarity(a,b)[0]
```

In [38]:
```
icd_distance('04110','acda500')
```

Out[38]:
```
array([-0.04589768])
```

In [122]:
```
acc_cost_matrix[-1,-1]
```

Out[122]:
```
-30.447746135219138
```

In [39]:
```
patient_id = [i for i in patient_event]
```

In [ ]:
```
patient_similarity = dict()
count = 0
for index in range(len(patient_id)):
    patient = patient_id[index]
    current = patient_event[patient]
    a = [str(i) for i in current]
    a = [i.lower() for i in a]
    patient_similarity[patient] = []
    for i in np.arange(index+1,len(patient_id)):
        compared = patient_event[patient_id[i]]
        b = [str(i) for i in compared]
        b = [i.lower() for i in b]
        d, cost_matrix, acc_cost_matrix, path = dtw(a, b, dist=icd_distance)
        patient_similarity[patient].append(acc_cost_matrix[-1,-1])
```

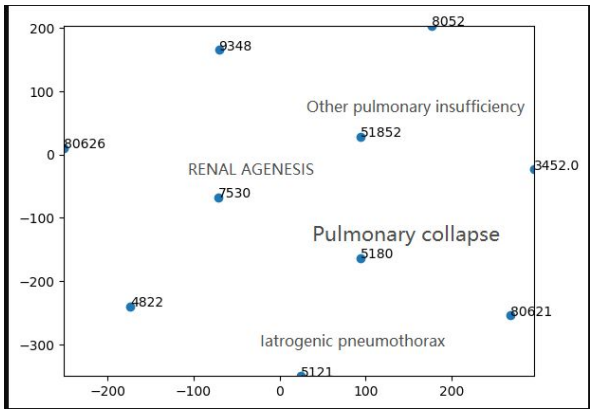

Figure: zoomed in sample space around 5180 (Pulmonary collapse).
(i.e. 10 nearest event code to 5180)

References:

Guo, Rongchen et al. *Comparative Visual Analytics for Assessing Medical Records with Sequence Embedding.* ArXiv abs/2002.08356 (2020): n. pag.

S. Guo, et al., *Visual Progression Analysis of Event Sequence Data in IEEE Transactions on Visualization & Computer Graphics*, vol. 25, no. 01, pp. 417-426, 2019. doi: 10.1109/TVCG.2018.2864885.

Graphen AI.