COMP 204 - Assignment 3

Yue Li

Important instructions:

- Release date: February 22, 2019 at 12:00 AM
- Due date: **March 10, 2019 at 23:59**
- Download the Python file `mining_electronic_health_records.py`
- Complete the 7 programming questions (plus bonus question) within the Python file
- Submit the completed `mining_electronic_health_records.py` on MyCourses
- Write your name and student ID at the top of the file
- Do not use any modules or any pieces of code you did not write yourself
- For each question, your program will be *automatically* tested on a variety of test cases, which will account for 75% of the mark. To be considered correct, your program should print out *exactly and only* what the question specifies. Do not add extra text, extra spaces or extra punctuation.
- For each question, 25% of the mark will be assigned by the TA based on (i) your appropriate naming of variables; (ii) commenting of your program; (iii) simplicity of your program (simpler = better).

1 International Classification of Diseases (ICD)

International Classification of Diseases (ICD) is an international standard classification of diseases published by the World Health Organization (WHO) [1]. It contains codes for diseases, signs and symptoms, abnormal findings, complaints, social circumstances, and external causes of injury or diseases. ICD is widely used as part of the electronic health records (EHR) that characterize patient medical conditions by a comprehensive list of electronic codes, which allows researchers to develop machine learning algorithms for prediction of early diagnosis and early interventions.

ICD is periodically updated. In this assignment, we will be using ICD 9th edition (ICD-9), a slightly out of date version, nonetheless the most used versions in healthcare system at the time. In particular, we are using the ICD-9 codes obtained from `https://icdlist.com/icd-9/index`.

Take a look at the text file `icd9_info.txt` that is provided to you. The ICD-9 codes are structured data and grouped by common diseases. For example, the ICD-9 codes from 001 to 009 are grouped under “Intestinal infectious diseases (001-009)" and the codes from 010 to 018 are under “Tuberculosis (010-018)". In the icd9_info.txt file, the disease group name appears first followed by the specific ICD-9 codes in that group one per line. For example, the first 14 lines are (where the group names are highlighted in bold):

Intestinal infectious diseases (001-009)
001 Cholera
002 Typhoid and paratyphoid fevers
003 Other salmonella infections
004 Shigellosis
005 Other food poisoning (bacterial)
006 Amebiasis
007 Other protozoal intestinal diseases
008 Intestinal infections due to other organisms
009 Ill-defined intestinal infections
Tuberculosis (010-018)
010 Primary tuberculous infection
011 Pulmonary tuberculosis
012 Other respiratory tuberculosis

Question 1: Create an ICD-9 encyclopedia (15 points)

Implement the following function:

```python
def process_icd9_file(filename):
    """
    args:
        filename: Name of file containing ICD-9 definition
    returns:
        dictionary of dictionaries
        level 1 dictionary: icd9 group name as key and icd9 info
↪   under the group as the value (level 2 dictionary)
```

```
        level 2 dictionary: icd9 code as key icd9 name as values
    """
```

The function will read icd9_info.txt line by line while keeping track whether the current line in icd9_info.txt is an ICD-9 group or an ICD-9 code. In total, there are 157 ICD-9 groups and 1234 ICD-9 codes. For instance, the first ICD-9 group is “Intestinal infectious diseases **(001-009)**", and the second group is “Tuberculosis **(010-018)**" and so on. In these cases, the group names start with English descriptions of the disease groups followed by a range, which indicates the first ICD-9 code and the last ICD-9 code in the group separated by a dash “-".

However, not all of the group names contain a dash separating two numerical values. For instance, the group name “Human immunodeficiency virus (042)" contains only one ICD-9 code 042, which is a disease group on its own because it significantly departs from the other diseases.

Moreover, although majority of the ICD-9 codes contain only 3 integer values (e.g., , “001 Cholera" or “038 Septicemia"), the ICD-9 codes under external causes of injury and supplementary classification start with letter ‘E’ and ‘V’, respectively. For example, take a look at the lines starting at line 1045 and line 1286 in file icd9_info.txt,

External Cause Status (E000)
E000 External cause status
Activity (E001-E030)
E001 Activities involving walking and running
E002 Activities involving water and water craft
E003 Activities involving ice and snow
E004 Activities involving climbing, rappelling, and jumping off

Persons with potential health hazards related to communicable diseases (V01-V09)
V01 Contact with or exposure to communicable diseases
V02 Carrier or suspected carrier of infectious diseases
V03 Need for prophylactic vaccination and inoculation against bacterial diseases
V04 Need for prophylactic vaccination and inoculation against certain viral diseases
V05 Need for other prophylactic vaccination and inoculation against single diseases
V06 Need for prophylactic vaccination and inoculation against combinations of diseases

The above are the caution we should take when implementing the `process_icd9_file` function. After implementing the function, it is useful to validate its correctness by printing it out the content of the encyclopedia as follows:

```
1 icd9_encyclopedia = process_icd9_file("icd9_info.txt")
2 print("\n\n**** output from q1 ****")
3 for k,x in icd9_encyclopedia.items():
4     print('group name', k, sep='\t')
5     for k1,x1 in x.items():
6         print(k1, x1, sep='\t')
```

The `print` in outer for loop (line 4) prints out the group name, where the `print` in the inner for loop (line 6) prints out the individual ICD-9 code and its name. Below are the first and last few lines printed to the screen.

```
group code Intestinal infectious diseases (001-009)
001 Cholera
002 Typhoid and paratyphoid fevers
003 Other salmonella infections
004 Shigellosis
```

```
group name Genetics (V83-V84)
V83 Genetic carrier status
V84 Genetic susceptibility to malignant neoplasm
group name Body mass index (V85)
V85 Body mass index
group name Estrogen receptor status (V86)
V86 Estrogen receptor status
group name Other specified personal exposures and history presenting hazards to health (V87)
V87 Other specified personal exposures and history presenting hazards to health
group name Acquired absence of other organs and tissue (V88)
V88 Acquired absence of other organs and tissue
group name Other suspected conditions not found (V89)
V89 Other suspected conditions not found
group name Retained foreign body (V90)
V90 Retained foreign body
group name Multiple gestation placenta status (V91)
V91 Multiple gestation placenta statu
```

To help you debug your program, the correct output file named **`icd9_encyclopedia.json`** file is included in this assignment.

2 Patient ICD-9 data from intensive critical units

MIMIC-III ('Medical Information Mart for Intensive Care') is a large, single-centre database comprising information relating to patients admitted to critical care units at a large tertiary care hospital namely Beth Israel Deaconess Medical Center (BIDMC) at Boston Massachusetts. Data includes vital signs, medications, laboratory measurements, observations and notes charted by care providers, fluid balance, procedure codes, diagnostic codes, imaging reports, hospital length of stay, survival data [2]. In this assignment, we will be using only the diagnostic code or more specifically the ICD-9 codes.

The gzip compressed file `DIAGNOSES_ICD.csv.gz` contains 651,046 ICD-9 records from the MIMIC-III database for 46,521 de-identified distinct patients. The data is in comma-separated value (csv) format. That is, the columns are separated by comma. The first few lines of the file read as follows:

```
"ROW_ID","SUBJECT_ID","HADM_ID","SEQ_NUM","ICD9_CODE"
243,34,115799,8,"E8790"
244,34,144319,1,"42789"
245,34,144319,2,"42822"
246,34,144319,3,"4263"
247,34,144319,4,"41401"
248,34,144319,5,"V5861"
249,34,144319,6,"4280"
250,34,144319,7,"2449"
251,34,144319,8,"3659"
252,35,166707,1,"3962"
253,35,166707,2,"4260"
254,35,166707,3,"2875"
```

- SUBJECT_ID: the unique patient identifier. The same patient may have several entries recorded in multiple lines. For instance, in the above lines, patient '34' has 9 entries recorded;
- HADM_ID: admission identifier. The same patient may have multiple admissions;
- ICD9_CODE: ICD-9 code mostly consistent with the ICD-9 encyclopedia we built in question 1 except that the ICD-9 code in the patient data are one or two character longer with the trailing one or two integers indicating more defined disease or symptoms. For instance, E8790 codes for abnormal reaction to cardiac catheterization whereas E879 codes for a more general cause of abnormal reaction of patient, or of later complication.
- ROW_ID and SEQ_NUM are for book keeping purpose only

Question 2. Building patient ICD-9 database (15 points)

For this assignment, we will use columns **SUBJECT_ID** and **ICD9_CODE**. As we have learn from Lecture 20 in class, a very efficient way to import a gzip compressed csv file is by the function called `read_csv(filename, compression="gzip")` from the Python library `pandas`, which will return a `DataFrame` object with 651,047 rows and 5 columns as shown above. We can iterate the `patient_data` row by row (or entry by entry the same way as we are reading a file line by line). In particular, the follow code snippet will create a dictionary with the patient ID (SUBJECT_ID) as key and a `set` of ICD-9 codes as values.

```python
import pandas as pd
filename = "DIAGNOSES_ICD.csv.gz"
patient_data = pd.read_csv(filename, compression="gzip")
patient_records = {} # save patient ICD-9 code into dictionary
for index,row in patient_data.iterrows(): # iterate row by row
    patId = row['SUBJECT_ID'] # access column "SUBJECT_ID"
    icd9_code = row['ICD9_CODE'] # access column "ICD9_CODE"
    if patId not in patient_records:
        patient_records[patId] = set([])
    patient_records[patId].add(icd9_code)
```

Complete the following function specified by the ‘docstring’ below

```python
def process_patient_data(filename, max_patients=5000):
    """
    args:
        filename: "DIAGNOSES_ICD.csv.gz"
        max_patients: optional argument for maximum number of
    ↪   patients to store
        in the dictionary
    returns:
        a dictionary with patient ID as key and a list of ICD-9 code
    ↪   as values
    """
```

NOTE

The ICD-9 code from DIAGNOSES_ICD.csv.gz is not directly compatible with `icd9_encyclopedia` returned from `process_icd9_file()` in question 1. To use `icd9_encyclopedia`, we will need to *parse* the ICD-9 code as follows:

- If the ICD-9 code is numeric or starts with “V", we will keep only the first 3 characters of the ICD-9 code from patient data and stored them as the values in the ICD-9 set; For instance, for ICD-9 code `42789` (i.e., second entry for patient 34), we only keep `427` whereas for code `V5861`, we only keep `V58`
- If the ICD-9 code starts with “E", the first 4 characters of the ICD-9 code is stored as the values in the ICD-9 set. For example, for ICD-9 code `E8790`, we only keep `E879`.

Unless specified, we will import the first 5000 patients from the DIAGNOSES_ICD.csv.gz file (i.e., the default value for the function). A correct function should return `patient_records` as a dictionary. To verify, we can print out the dictionary as follows.

```python
patient_records = process_patient_data("DIAGNOSES_ICD.csv.gz")
print("\n\n**** output from q2 ****")
for k,x in patient_records.items():
    print(k, x ,sep='\t')
```

After printing, the first 10 out of 5000 patient records should display as follows, where the first number is the patient ID (key) and rest are the `set` of ICD-9 code as values:

```
34 {'426', '244', '414', 'E879', '427', '997', '410', '425', '365', '428', 'V58'}
35 {'426', '250', '244', '414', '427', '997', '287', '396', '715', '401'}
36 {'453', '411', 'V10', '401', '996', '415', '486', '997', '596', '496', '414', '998', 'V45', '300', '553', '518', '305', '530', '600'}
37 {'285', '496', '250', '414', '427', '486', '410', '535', '428'}
38 {'995', '584', '414', '427', '997', '998', 'E870', '608', '560', 'V45', '428', '038'}
39 {'V05', 'V29', '765', 'V30', '770', '769', '774', '276'}
41 {'348', '518', '285', '507', '482', '513', '496', '512', '191', '401', '305', '709', '707', 'V10', '112', '788'}
42 {'414', '427', '438', '412', '428', '401'}
43 {'E821', '518', '805'}
44 {'780', '414', '427', '272', '596', '443', '396', '413', '401', 'V17'}
45 {'911', '305', 'E812', '873', '913', '998', '824'}
```

To help you debug your program, the correct output file named **`patient_records.json`** file is included in this assignment. Note that the JSON somehow does not work proper with `set` object. The sets in the `patient_records` has been converted to lists before saving to JSON file. After loading the `patient_records.json` file, you will need to convert each `list` back to `set` as follows:

```
outfile = "patient_records.json"
f = open(outfile, "r")
patient_records = json.load(f)
for k,x in patient_records.items():
    patient_records[k] = set(x)
f.close()
```

Question 3. Calculate the average number of patients for any ICD-9 code (10 points)

The effective sample size for each specific ICD-9 code is small. In other words, for any given ICD-9 code, only a small number of patients were recorded with that code. To test this, write a function that calculate the average number of patients for any given ICD-9 code.

```
def average_patient_icd9code(patient_records):
    """
    args:
        patient_records obtained from
 ↪  process_patient_data("DIAGNOSES_ICD.csv.gz")
    returns:
        average number of patient observed per ICD-9 code
    """
```

Executing the function should print out only the average patient number:

```
>>> print(f"Average patient count
 ↪  {average_patient_icd9code(patient_records):.2f}")
# Average patient count 57.54
```

3 Diagnosing patients by calculating the ICD-9 codes frequency among closely matched patients in similar disease groups

Our main goal in this assignment is to predict the missing ICD-9 code for any given test patient. We will implement two simple tasks:

1. For a test patient, find k closely matched patients based on the similarity of their ICD-9 codes

2. Calculate the frequency of the specific ICD-9 codes among the closely matched patients. The unrecorded ICD-9 codes with high frequencies for the test patient are considered as the highly plausible codes that might have been missed during the diagnosis process.

In machine learning, this simple procedure is generally known as the k-*nearest neighbour* (kNN) prediction, where the k “nearest neighbours" in our case are those k closely matched patients to the test patient.

However, as we can see, the average number of patients observed per ICD-9 code is only 57.5 out of 5000 patients (i.e., only 1.15%). Such low representation of specific ICD-9 code among patient cohort makes it difficult to directly apply kNN because it is difficult to draw convincing conclusion based on the small sample size.

Question 4. Group the ICD-9 codes for each patient into disease groups (15 points)

To remedy the problem, we will represent patients’ ICD-9 codes from `patient_records` (Section 2) using the *157 disease groups* that we have stored in the `icd9_encyclopedia` (Section 1). For example, we want to represent the ICD-9 codes for patient 34 and 35:

```
34 {’426’, ’244’, ’414’, ’E879’, ’427’, ’997’, ’410’, ’425’, ’365’, ’428’, ’V58’}
35 {’426’, ’250’, ’244’, ’414’, ’427’, ’997’, ’287’, ’396’, ’715’, ’401’}
```

as their corresponding disease groups:

```
34 {’Surgical and medical procedures as the cause of abnormal reaction of patient or later complication, without mention of misadventure at the time of procedure (E878-E879)’, ’Persons encountering health services for specific procedures and aftercare (V50-V59)’, ’Complications of surgical and medical care, not elsewhere classified (996-999)’, ’Other forms of heart disease (420-429)’, ’Disorders of the eye and adnexa (360-379)’, ’Disorders of thyroid gland (240-246)’, ’Ischemic heart disease (410-414)’}
35 {’Diseases of other endocrine glands (249-259)’, ’Hypertensive disease (401-405)’, ’Complications of surgical and medical care, not elsewhere classified (996-999)’, ’Arthropathies and related disorders (710-719)’, ’Diseases of the blood and blood-forming organs (280-289)’, ’Other forms of heart disease (420-429)’, ’Disorders of thyroid gland (240-246)’, ’Ischemic heart disease (410-414)’, ’Chronic rheumatic heart disease (393-398)’}
```

[5 out of the 15 points] To this end, It is helpful to first implement a function that process the `icd9_encyclopedia` dictionary object and return another dictionary that with key as icd-9 code and value as the corresponding disease group names:

```python
def process_icd9_encyclopedia(icd9_encyclopedia):
    """
    args:
        icd9_encyclopedia obtained from
↪   process_icd9_info("icd9_info.txt")
    returns:
        icd9_encyclopedia_reverseIndex: dictionary with key as icd-9
↪   code and value
        as the corresponding disease group names
    """
```

[10 out of the 15 points] Next, implement the `summarize_patient_records` to group ICD-9 codes into groups for each patient. **Please make sure to use `set` to store the disease groups per patient.**

```python
def summarize_patient_records(patient_records, icd9_encyclopedia_reverseIndex):
    """
    args:
        patient_records obtained from process_patient_data("DIAGNOSES_ICD.csv.gz")
        icd9_encyclopedia_reverseIndex from process_icd9_encyclopedia(icd9_encyclopedia)
    returns:
        patient_records_summary: a dictionary with patient ID as key and a set of
        disease group names as value
    """
```

Question 5. Find similar patients (15 points)

Suppose $\mathcal{A}$ is the set of disease groups for patient A and $\mathcal{B}$ is the set of disease groups for patient B. The similarity score between patient A and patient B is defined as:

$$Similarity(\mathcal{A}, \mathcal{B}) = |\mathcal{A} \cap \mathcal{B}| - |\mathcal{A} - \mathcal{B}| - |\mathcal{B} - \mathcal{A}| \tag{1}$$

In English, this means that the similarity between patient A and patient B is equal to the number disease groups they have in common subtracted by the number of disease groups unique to patient A and the number of disease groups unique to patient B.

Complete the following function to calculate the similarity between each of the query patients and the rest of the patients:

```python
def get_patients_similarity(query_patient_records, patient_records_summary, icd9_encyclopedia_reverseIndex):
    """
    args:
        query_patient_records: same compound type as patient_records but for a test set of patients
        patient_records_summary: dictionary obtained from
        summarize_patient_records(patient_records, icd9_encyclopedia_reverseIndex)
        icd9_encyclopedia_reverseIndex from process_icd9_encyclopedia(icd9_encyclopedia)
    returns:
        patient_similarity: a dictionary with key as test patient ID and value as a list of 2-value tuples
        The first value in the tuple is the neighbor patient ID and the second value in the tuple is the
        similarity score between the neighbor patient and the test patient
    NOTE: the list must *not* contain the query patient ID and the similarity with the query patient themselves
    """
    query_patient_records_summary =
     ↪  summarize_patient_records(query_patient_records,
     ↪  icd9_encyclopedia_reverseIndex)

    patient_similarity = {}

    # YOUR CODE HERE

    return patient_similarity
```

To sort a list of tuples, we will need to define a helper function:

```
def getKey1(item):
    return item[1]
```

This will tell the `sort` function to sort on the second value in the tuple. For example,

```
>>> x=[('100',-1), ('3',0), ('39',5)]
>>> x.sort(key=getKey1) # sort in increasing order (default)
>>> print(x) # [('100', -1), ('3', 0), ('39', 5)]
>>> x.sort(key=getKey1, reverse=True) # sort in decreasing order
>>> print(x) # [('39', 5), ('3', 0), ('100', -1)]
```

To help debug your program, 3 query patients were intentionally chosen from the 5000-patient `patient_records` by the following snippet. These 3 patients are:

- patient '71': Schizophrenia patient with ICD-9 code '295' among many other codes
- patient '85': Parkinson's disease patient with ICD-9 code '332' among other codes
- patient '124': Chronic bronchitis patient with ICD-9 code '491'

The 3 patients do not share the target disease codes (i.e., 295, 332, 491). Nonetheless, there are many disease complications among these patients because they often not only have the target diseases but also many other diseases, some of which are related to the target diseases whereas many were caused by other factors such as aging (Parkinson) or smoking (Chronic brochitis).

```
icd9group_examples = [[295], [332], [491]]
example_disorders_all = set([])
for i in icd9group_examples:
    for j in i:
        example_disorders_all.add(str(j))

query_patient_records = {}
for i in icd9group_examples:
    example_disorders = set([])
    for j in i:
        example_disorders.add(str(j))
    for patId,icd9list in patient_records.items():
        if len(icd9list & example_disorders) > 0 and \
        len(icd9list & (example_disorders_all-example_disorders)) ==
        ↪  0:
            query_patient_records[patId] = icd9list
            break
print(query_patient_records.keys()) # dict_keys(['71', '85', '124'])
```

Once you have implemented `get_patients_similarity` function, it helps to verify that the function correctly detects the most similar patients for the 3 query patients as follows:

```
patient_similarity = get_patients_similarity(query_patient_records,
    patient_records_summary, icd9_encyclopedia_reverseIndex)

for k,x in patient_similarity.items():
    print(k, x[0:5])

#71 [('6693', 1), ('1438', 0), ('2183', 0), ('4596', 0), ('750', -1)]
#85 [('5166', 0), ('2061', -1), ('5107', -1), ('2598', -2), ('4676',
 ↪  -2)]
#124 [('3868', 6), ('2061', 3), ('323', 2), ('3426', 2), ('1006', 1)]
```

The full patient similarity dictionary file saved as **patient_similarity.json** is also included with this assignment for debugging purpose.

Question 6. Patient ICD-9 code prediction (15 points)

Finally, we can implement the `make_diagnosis` function for any one or more query patient records. The general guideline is as follows:

1. First get the `patient_similarity` dictionary that records for each query patient (key) the sorted list of patients from the most similar to the least similar patients
2. To predict ICD-9 code for each query patient,
   (a) Find the `top_k` most similar patients not including the query patient. Here you will need to sort the list of tuples);
   (b) Calculate the frequencies of ICD-9 codes observed among the `top_k` most similar patients;
   (c) Save as a list of tuples, where the first value of the tuple is ICD-9 code, the second value is the ICD-9 frequency estimated from the `top_k` most similar patients (not including the query patient him/herself)

Complete the diagnosis function as specified below.

```python
def make_diagnosis(query_patient_records, patient_records_summary, patient_records, top_k):
    """
    args:
        query_patient_records: same compound type as patient_records but for a test set of patients
        patient_records_summary: obtained from summarize_patient_records(patient_records)
        patient_records: obtained from process_patient_data("DIAGNOSTIC_CODE.csv.gz")
        top_k: integer value specifies the number of most closely matched patients to each query patient
    returns:
        query_patient_diagnosis: a dictionary with key as the query patient ID and value as the
        sorted list of tuples. The first value in the tuple is the icd-9 code and the second value in the
        tuple is the frequency of icd-9 observed among the top_k matched patient. The list is sorted in _decreasing_
        order by the frequency of the ICD-9 code
    """
    patient_similarity = get_patients_similarity(query_patient_records,
        patient_records_summary, icd9_encyclopedia_reverseIndex)

    query_patient_diagnosis = {}

    # YOUR CODE HERE

    return query_patient_diagnosis
```

Once you have implemented the `make_diagnosis` function, use the following functions to check the correctness of your implementation:

```python
def getCode_icd9code_Info(icd9_encyclopedia):

    icd9_info = {}

    for k,x in icd9_encyclopedia.items():
        for icd9_code, icd9_names in x.items():
            icd9_info[icd9_code] = {'group':k, 'code':icd9_code, 'name':icd9_names}
    return icd9_info

def write_diagnosis_report(query_patient_diagnosis, query_patient_records, icd9_info, outfile):

    f = open(outfile, 'w')

    f.write("patId\tsymptoms_status\ticd9 group\tICD9 code\tICD9
     ↪  name\tFrequency\n")

    for patId, diagnosis in query_patient_diagnosis.items():

        for icd9 in query_patient_records[patId]:

            x = icd9_info[icd9]

             ↪  f.write(f"{patId}\tobserved\t{x['code']}\t{x['name']}\t1\n")

        for icd9_tuple in diagnosis:
            icd9_code = icd9_tuple[0]
            icd9_freq = icd9_tuple[1]
            if icd9_freq > 0.2:
                icd9_code_info = icd9_info[icd9_code]

                 ↪  f.write(f"{patId}\tpredicted\t{icd9_code}\t{icd9_code_inf
    f.close()

icd9_info = getCode_icd9code_Info(icd9_encyclopedia)
outfile = ``diagnosis_report.txt"
write_diagnosis_report(query_patient_diagnosis,
 ↪  query_patient_records, icd9_info, outfile)
```

The function `write_diagnosis_report` will write to the file named `diagnosis_report.txt` the patient observed ICD-9 as well as the top predicted ICD-9 code with the frequency greater than 0.2. For example, below is the diagnosis report for patient 71:

```
patId symptoms_status icd9 group ICD9 code ICD9 name Frequency
71 observed 285 Other and unspecified anemias 1
71 observed 266 Deficiency of B-complex components 1
71 observed E950 Suicide and self-inflicted poisoning by solid or liquid substances 1
71 observed 295 Schizophrenic psychoses 1
71 observed 969 Poisoning by psychotropic agents 1
71 observed 276 Disorders of fluid, electrolyte, and acid-base balance 1
71 predicted 296 Affective psychoses 0.55
71 predicted 965 Poisoning by analgesics, antipyretics, and antirheumatics 0.4
71 predicted 305 Nondependent abuse of drugs 0.35
```

Under the column “symptoms_status", “observed" indicates that the ICD-9 code is already recorded for the patient (hence the value in the “Frequency" column is equal to 1) whereas “predicted" indicate the ICD-9 code is not yet observed but predicted with high frequency equal or greater than 0.2. To help you debug your code, the full content of `diagnosis_report.txt` is included in this assignment.

Question 7. Evaluating the diagnosis accuracy (15 points)

In bioinformatics, it is crucial to quantify the prediction accuracy in an unbiased way. To this end, we are going to split the 5000 patients into halves, we will use the first half (2500 patients) as *the reference patients* or often referred to as the “training data" and the second half as *the test patients* or often referred to as the “testing data" as generated by the following snippet:

```python
all_patient_records = process_patient_data("DIAGNOSES_ICD.csv.gz", max_patients=5000)

train_patient_records = {}
test_patient_records = {}

n_train = 2500

for patId in all_patient_records.keys():
    if len(train_patient_records.keys()) < n_train:
        train_patient_records[patId] = all_patient_records[patId]
    else:
        test_patient_records[patId] = all_patient_records[patId]
```

Complete the following function:

```python
def icd9_prediction(target_icd9, test_patient_records, train_patient_records,
    icd9_encyclopedia_reverseIndex, top_k = 5):
    """
    args:
        target_icd9: icd9 code that we will be predicting from a subset of the test_patient_records
        test_patient_records: obtained from process_patient_data("DIAGNOSTIC_CODE.csv.gz")
        train_patient_records: obtained from process_patient_data("DIAGNOSTIC_CODE.csv.gz")
        icd9_encyclopedia_reverseIndex: obtained from process_icd9_encyclopedia(icd9_encyclopedia)
        top_k: integer value specifies the number of most closely matched patients to each query patient (default: 20)
    returns:
        accuracy: correct predictions divided by the total correct plus incorrect predictions
    """
    train_patient_records_summary = summarize_patient_records(train_patient_records, icd9_encyclopedia_reverseIndex)

    test_patient_records_new = {}

    for patId, icd9code in test_patient_records.items():
        if target_icd9 in icd9code:
            test_patient_records_new[patId] = icd9code -
             ↪  {target_icd9}

    test_patient_records_controls = {}

    for patId, icd9code in test_patient_records.items():
        if target_icd9 not in icd9code:
            test_patient_records_controls[patId] = icd9code
        if len(test_patient_records_controls) ==
         ↪  len(test_patient_records_new):
            break

    test_patient_records_new.update(test_patient_records_controls)

    test_patient_diagnosis = make_diagnosis(test_patient_records_new,
        train_patient_records_summary, train_patient_records, top_k)

    correct = 0
    incorrect = 0

# YOUR CODE HERE

    accuracy = correct/(correct+incorrect)

    return accuracy
```

Below is the pseudocode of the `icd9_prediction` procedure:

Algorithm 1 ICD9 prediction evaluation

```
 1: procedure ICD9 PREDICTION((target_icd9, test_patient_records, train_patient_records, top_k=5)
 2:     Summarize train_patient_records by disease groups
 3:     Extracts all of the “cases" from the 2500 patients who contain the target ICD-9 code but remove target ICD-9 code from these patient records as if it weren’t recorded for those patients;
 4:     Suppose there are $N/2$ cases. Choose the same number of controls by selecting the first $N/2$ patients that *do not contain* the target ICD-9 code
 5:     Merge the case and control patient datasets into a single test data set
 6:     Make prediction over the test data set using $k$ closely matched training patient records per test patient
 7:     correct ← 0; incorrect ← 0;
 8:     for test patient $j \leftarrow 0$ to $N-1$ do
 9:         if patient $j$ contains the target ICD-9 code (i.e., a case) then
10:             if the target ICD-9 code is in the predicted ICD-9 code list for patient $j$ then
11:                 if if the frequency for the target ICD-9 $>$ 1/top_k then
12:                     correct ← correct + 1
13:                 end if
14:             else
15:                 incorrect ← incorrect + 1
16:             end if
17:         else
18:             if the target ICD-9 code is in the predicted ICD-9 code list for patient $j$ then
19:                 if if the frequency for the target ICD-9 $>$ 1/top_k then
20:                     incorrect ← incorrect + 1
21:                 end if
22:             else
23:                 correct ← correct + 1
24:             end if
25:         end if
26:     end for
27:     accuracy ← correct/(correct+incorrect)
28:     return accuracy
29: end procedure
```

After completing `icd9_predict`, check the accuracy of predicting ICD-9 code 295, 332, and 491 using $K = 20$ closely matched patients with the following code:

```
accuracies = {}

icd9_target_list = ['295','332','491']

for target_icd9 in icd9_target_list:
    accuracies[target_icd9] = icd9_prediction(target_icd9,
                  test_patient_records, train_patient_records,
                    icd9_encyclopedia_reverseIndex, top_k = 20)

for icd9,accuracy in accuracies.items():
    print(f"{icd9}: {100*accuracy:.0f}%")
#295: 65%
#332: 61%
#491: 77%
```

Bonus: choosing the best k (10 points)

In the previous application, we chose $k = 20$ as an arbitrary choice (i.e., we used 20 closely matched reference patients to predict the target ICD-9 code for each test patient). Experiment with different values of k between 1 and 500 and report the k that yields the highest averaged prediction accuracy over the 3 target ICD-9 codes (i.e., ICD-9 295, 332, 491).

References

[1] World Health Organization. International classification of diseases:[9th] ninth revision, basic tabulation list with alphabetic index, 1978.

[2] Alistair E W Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. MIMIC-III, a freely accessible critical care database. *Scientific Data*, 3:160035–9, May 2016.