[1] ما ناهنجاری های موجود در تراکنش های پزشکی را از طریق یک روش تحلیل نمودار که با PGX انجام می شود ، تشخیص می دهیم. نتیجه می تواند بررسی بیشتر در آینده برای شناسایی تقلب احتمالی در سیستم مراقبت های بهداشتی باشد. ما از مجموعه داده های عمومی تراکنش های پزشکی CMS (مرکز خدمات پزشکی و دارویی ایالات متحده) سال 2012 استفاده می کنیم. این تراکنش ها بین بیش از 880هزار ارائه دهنده پزشکی و CMS است که در کل مبلغ بیش از 77 بیلیون دلار در سال است. مجموعه داده ها را می توان از وب سایت CMS دانلود کرد. در اینجا ، ما ابتدا نحوه تبدیل داده ها به نمای گراف را نشان می دهیم. سپس ما در مورد نحوه استفاده از PGX برای تجزیه و تحلیل داده های گراف بحث می کنیم.

* دانلود دیتاست

دانلود مجموعه داده Medicare Physician and Other Supplier PUF, CY2012 dataset و سپس اگسترکت کردن  Medicare\_Provider\_Util\_Payment\_PUF\_CY2012.txt

* تبدیل فرمت داده ها به EDGE\_LIST گراف

اسکریپت زیر را در فایلی به نام generateGraph.groovy کپی می کنیم. این فایل متنی بارگیری شده را به عنوان ورودی دریافت می کند و دو فایل را به عنوان خروجی تولید می کند:

* 1. خروجی outputFile،فایلی است که در آن قرار است گرافبه فرمت EDGE\_LIST ایجاد شود
  2. . خروجی specialtiesOutFile، فایلی است که در آن قرار است تخصص های مشخصی نوشته شود. بعداً به عنوان ورودی اضافی برای تحلیل خود به آنها نیاز داریم.

"Creating Dictionaries"

readyToProcess = **false**

doctors = **new** HashMap<Integer,List>()

hcpcs = **new** HashSet<Integer>()

specialties = **new** HashSet<String>()

inputFile = "/path/to/file/Medicare\_Provider\_Util\_Payment\_PUF\_CY2012.txt"

outputFile = "/path/to/file/bipartiteMedicGraph.txt"

specialtiesOutFile = "/path/to/file/specialties.txt"

npiPosition = -1

specPosition = -1

hcpcsPosition = -1

**new** File(inputFile).splitEachLine("\t") { fields ->

**if** (readyToProcess) {

**if** (!doctors.containsKey(fields[npiPosition])) {

doctors.put(fields[npiPosition],[**new** HashSet<Integer>(),fields[specPosition]])

}

**if** (!fields[hcpcsPosition].isInteger()) {

newHcpcs = ""

**for** (int i = 0; i < fields[hcpcsPosition].length(); i++) {

**if** (String.valueOf(fields[hcpcsPosition].charAt(i)).isInteger()) {

newHcpcs = newHcpcs + fields[hcpcsPosition].charAt(i)

}

**else**{

newHcpcs = newHcpcs + ((int)(fields[hcpcsPosition].charAt(i)))

}

}

fields[hcpcsPosition] = newHcpcs

}

specialties.add(fields[specPosition])

hcpcsValue = Integer.parseInt(fields[hcpcsPosition])

hcpcs.add(hcpcsValue)

doctors.get(fields[npiPosition])[0].add(hcpcsValue)

} **else** {

*/\**

*\* We are using this line because the first row is the header and in the*

*\* second we see the copyright, therefore we are not going to process that*

*\*/*

readyToProcess = (fields.size() == 2)

**if** (npiPosition < 0 || specPosition < 0 || hcpcsPosition < 0) {

npiPosition = fields.indexOf("NPI")

specPosition = fields.indexOf("PROVIDER\_TYPE")

hcpcsPosition = fields.indexOf("HCPCS\_CODE")

}

}

}

"Creating files"

ln = System.getProperty("line.separator")

graphFile = **new** File(outputFile)

**if** (graphFile.exists()) {

graphFile.delete()

graphFile.createNewFile()

}

append = **true**

fileWriter = **new** FileWriter(graphFile, append)

buffWriter = **new** BufferedWriter(fileWriter)

**for** (element **in** hcpcs.iterator()) {

buffWriter.write(element + " \* \"HCPCS\"" + ln)

}

**for** (key **in** doctors.keySet()) {

buffWriter.write(key + " \* \"" + doctors.get(key)[1] + "\"" + ln)

}

**for** (key **in** doctors.keySet()) {

**for** (key2 **in** doctors.get(key)[0].iterator()) {

buffWriter.write(key + " " + key2 + ln)

}

}

buffWriter.flush()

buffWriter.close()

specialtiesFile = **new** File(specialtiesOutFile)

**if** (specialtiesFile.exists()) {

specialtiesFile.delete()

specialtiesFile.createNewFile()

}

fileWriter = **new** FileWriter(specialtiesFile, append)

buffWriter = **new** BufferedWriter(fileWriter)

"Found " + specialties.size().toString() + " specialties, writing them to file"

**for** (spec **in** specialties.iterator()) {

buffWriter.write(spec + ln)

}

buffWriter.flush()

buffWriter.close()

* تولید گراف

برای اجرای اسکریپت بالا، متغییر inputFile و outputFile و specialtiesOutFileرا به ترتیب تغییر می دهیم و سپس مسیر را به اسکریپت بعنوان آرگومانی برای پوسته(shell) ی PGX منتقل می کنیم.

cd $PGX\_HOME

./bin/pgx generateGraph.groovy

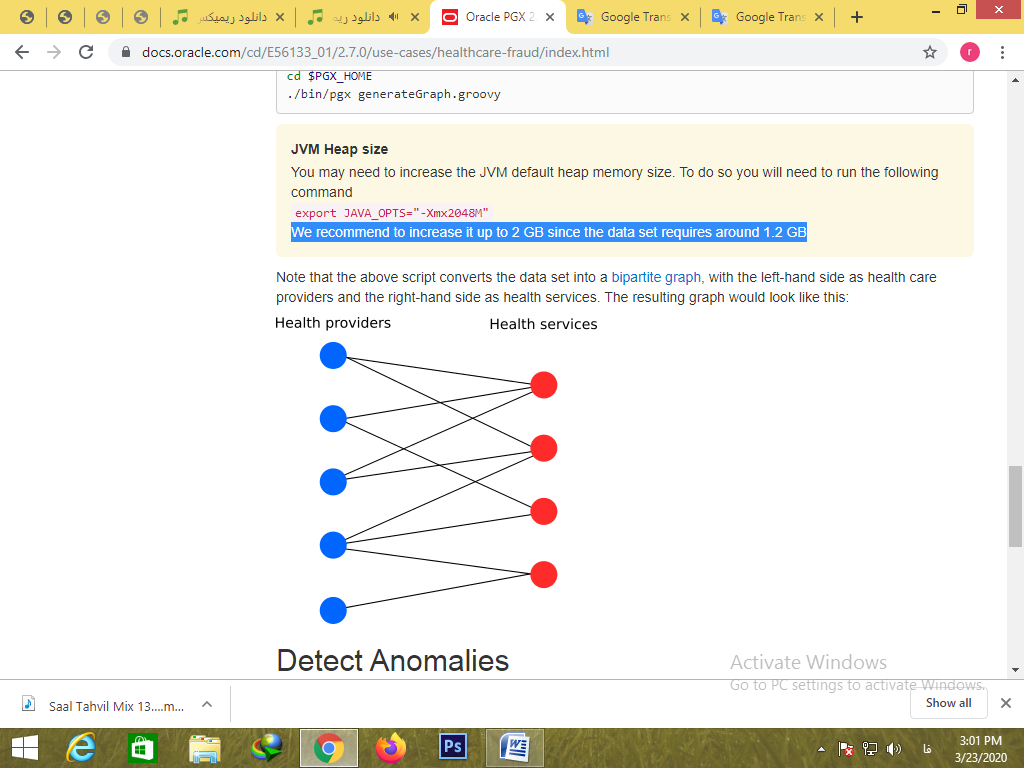
**JVM Heap size:**

شاید لازم باشد اندازه حافظه پیش فرض JVM را افزایش دهید. برای این کار باید دستور زیر را اجرا کنید

توصیه می کنیم آن را تا 2 گیگابایت افزایش دهید زیرا مجموعه داده ها به حدود 1.2 گیگابایت نیاز دارد

export JAVA\_OPTS="-Xmx2048M"

توجه داشته باشید که اسکریپت فوق داده های تبدیل شده را به یک نمودار دو طرفه تبدیل می کند ، در سمت چپ به عنوان ارائه دهنده خدمات درمانی و سمت راست به عنوان خدمات درمانی است. نمودار بدست آمده از این قرار است:



تشخیص ناهنجاری:

ایده تشخیص ناهنجاری به شرح زیر است:

یک لبه در نمودار بالا مربوط به یک ارائه دهنده پزشکی خاص (راس سمت چپ) است که یک سرویس بهداشتی خاص (راس سمت راست) ارائه داده است.

بنابراین با بیشتر شدن خدمات ارائه شده مشترک، ارائه دهندگان پزشکی در نمودار نزدیک تر می شوند ، به این معنا که از طریق مسیرهای کوتاه (دو هاپ) بسیاری به هم متصل می شوند.

از سوی دیگر ، مجموعه داده ها همچنین "تخصص" ارائه دهندگان پزشکی را مشخص می کند. از آنجا که ارائه دهندگان پزشکی با همان تخصص احتمالاً بسیاری از خدمات مشترک را انجام می دهند ، انتظار می رود که این راسها نسبت به سایرین به یکدیگر نزدیک باشند.

اگر یک ارائه دهنده پزشکی وجود دارد که استثنایی با سایر ارائه دهندگان با تخصص های مختلف نزدیک است ، آن را غیر عادی تلقی می شود - ارائه دهنده خدمات درمانی را انجام می دهد که به طور معمول توسط ارائه دهندگان با تخصص متفاوت انجام می شود.

برای محاسبه این نزدیکی در میان رئوس ، از Personalran Pagerank (PPR) استفاده می کنیم. به طور خاص ، برای هر تخصص ، ما امتیاز PPR را برای کلیه ارائه دهندگان، با تعیین راس های تخصص فعلی ، به عنوان رأس های شروع، محاسبه می کنیم. نمره PPR از یک راس داده شده این احتمال را نشان می دهد که پیاده روی های تصادفی از رئوس شروع در این راس به پایان برسد. در نتیجه ، نمره PPR بالا برای یک راس نشان می دهد که راس نزدیک به راسهای شروع است.

پس از محاسبه نمره PPR در هر تخصص ، مقادیر را بررسی می کنیم تا ناهنجاری ها را شناسایی کنیم. یعنی ، ما به بالاترین مقادیر رئوس نگاه می کنیم و می بینیم که آیا آنها متعلق به تخصص فعلی هستند یا خیر. اگر اینگونه نباشد ، ارائه دهنده آنومالی تلقی می شود.

ما دو پیاده سازی از این تجزیه و تحلیل را در زیربخش های زیر ارائه می دهیم. اولین رویکرد از الگوریتم Personalized PageRank built-in استفاده می کند. مورد دوم با نوشتن و کامپایل یک الگوریتم گراف دلخواه با Green-Marl باعث بهبود مصرف حافظه و عملکرد آنالیز می شود.

* اسکریپت زیر را درون پوسته PGX اجرا کنید:
* specialties = []
* "Reading specialties"
* specialtiesPath = "/path/where/this/was/created/specialties.txt"
* graphFilePath = "/path/where/this/was/created/bipartiteMedicGraph.txt"
* **new** File(specialtiesPath).eachLine { line ->
* specialties.push(line)
* }
* "Creating the graph config"
* builder = GraphConfigBuilder.forFileFormat(Format.EDGE\_LIST)
* builder.setUri(graphFilePath)
* builder.addVertexProperty("specialty", PropertyType.STRING, "")
* builder.setVertexIdType(IdType.INTEGER)
* cfg = builder.build()
* "Load the graph from a config into PGX"
* directedG = session.readGraphWithProperties(cfg)
* g = directedG.undirect()
* directedG.destroy()
* "Iterating over all specialties"
* **for** (item **in** specialties) {
* println item *// Printing the specialty that was filtered out*
* v = g.getVertices(**new** VertexFilter("vertex.specialty == '"+item+"'"))
* pgRank = analyst.personalizedPagerank(g, v, 0.001, 0.85, 1000)
* subgraph = g.filter(**new** VertexFilter("vertex.specialty != '"+item+"' AND vertex.specialty != 'HCPCS'"))
* *// We get the top pagerank values from the subgraph*
* resultSet = subgraph.queryPgql("SELECT suspect, suspect.${pgRank.getName()} AS pagerank MATCH (suspect) ORDER BY suspect.${pgRank.getName()} DESC LIMIT 10")
* resultSet.print();
* *// We clean up the subgraph since we don't need it anymore*
* subgraph.destroy()
* }

توجه داشته باشید که باید دو متغیر زیر را برای اشاره به فایل های های خروجی اسکریپت تبدیل گراف مان تغییر دهیم:

* specialtiesPath مسیر فایل های حاوی مقدار تخصص ها است.
* GraphFilePath مسیر رسیدن به گراف دو طرفه تولید شده با فرمت EDGE\_LIST است.

اجازه دهید نگاهی دقیق تر به آنچه اسکریپت انجام می دهد بیندازیم:

1. تخصص ها را در یک لیست بخوانید
2. یک پیکربندی گرافی را ایجاد کنید که به گراف دو بخشی ارجاع دارد. تخصص مرتبط با هر ارائه دهنده به عنوان یک رشته ویژگی راس(a string vertex property) بارگذاری می شود.
3. گراف را در مموری بخوانید.
4. یک کپی غیر مستقیم از گراف ایجاد کنید.
5. روی همه تخصص ها حلقه بزن، به طوری که برای هر حلقه:

5.1) تمام رئوس مرتبط با تخصص فعلی را در گراف پیدا کنید.

5.2) الگوریتم Personalran Pagerank را بر روی گراف بدون جهت متمایل به سمت راسهای یافت شده ما با یک عامل میرایی[[1]](#footnote-2) 0.85 ، حداکثر اختلاف خطای[[2]](#footnote-3) 0.001 و حداکثر تعداد تکرار[[3]](#footnote-4) 1000 اجرا کنید.

5.3) یک زیرگراف ایجاد کنید که شامل همه ارائه دهندگانی که شامل تخصص فعلی نیستند، باشد.

5.4) 10 ارائه دهنده برتر در آن زیرگراف با بالاترین امتیاز PageRank شخصی شده را چاپ کنید.

خروجی

==> Iterating over all specialties

Pain Management

suspect pagerank

============================================

1124020011 0.03378340111411831

1023236478 0.03188995380346009

1356353247 0.03125303489326371

1689644668 0.031087249961419247

1043213259 0.03092537640401708

1952347981 0.03041650032484901

1568465763 0.029535881620308922

1982872990 0.028917315945686653

1538130018 0.028529067396474646

1629071527 0.02803827130175676

Pathology

suspect pagerank

============================================

1134277494 1.7097787102012583

1346233277 1.6401878728267945

1194769497 1.6374635275343175

1932145778 1.4326716377223467

1538105366 1.4233972566122781

1497773337 1.4107892152282349

1205896594 1.3572755649336943

1538144910 1.3541746206567151

1366543795 1.3011186472395846

1548280852 1.2980797553943368

...

* Advanced: Detect Anomalies Using a Custom Personalized PageRank

اسکریپت زیرمحاسبات همانند اسکریپت اصلی معرفی شده در بخش قبلی دارد. تفاوت در این است که ما از نسخه سفارشی الگوریتم Personalran Pagerank استفاده می کنیم که بجای ایجاد ابتدا یک مجموعه راس، به طور مستقیم روی خاصیت تخصص گراف تولید شده کار می کند.

ابتدا برنامه Green-Marl زیر را در فایل personalized\_pagerank\_from\_kind.gm کپی کنید:

*/\**

*\* Copyright (C) 2013 - 2018 Oracle and/or its affiliates. All rights reserved.*

*\*/*

**procedure** personalized\_pagerank\_from\_kind(G: graph, filter: string,

specialty: nodeProp<string>, e,d: double, max\_iter\_count: int;

pg\_rank: nodeProp<double>)

{

**if** (G.numNodes() == 0){

**return**;

}

nodeProp<bool> is\_start;

G.is\_start = \_.specialty == filter;

int M = count(n : G.nodes)(n.specialty == filter);

double diff;

int cnt = 0;

G.pg\_rank = (\_.is\_start) ? 1.0/M : 0;

**do** {

diff = 0.0;

**foreach** (t: G.nodes) {

double val1 = (t.is\_start) ? (1-d) : 0;

double val2 = d\* sum(w: t.inNbrs) {w.pg\_rank / w.outDegree()} ;

double val = val1 + val2;

diff += | val - t.pg\_rank |;

t.pg\_rank <= val;

}

cnt++;

} **while** ((diff > e) && (cnt < max\_iter\_count));

}

بعد ، اسکریپت زیر را در پوسته PGX اجرا کنید:

specialities = ["Gynecological/Oncology","Geriatric Psychiatry","Ambulance Service Supplier","Clinical Psychologist","Unknown Supplier/Provider","Anesthesiology","Certified Nurse Midwife","Anesthesiologist Assistants","Pain Management","Sleep Medicine","Colorectal Surgery (formerly proctology)","Plastic and Reconstructive Surgery","Otolaryngology","Portable X-ray","Endocrinology","Osteopathic Manipulative Medicine","Preventive Medicine","Nurse Practitioner","Dermatology","Independent Diagnostic Testing Facility","Pathology","Infectious Disease","Allergy/Immunology","Pharmacy","Maxillofacial Surgery","Thoracic Surgery","Pediatric Medicine","Radiation Therapy","Licensed Clinical Social Worker","Physical Medicine and Rehabilitation","Family Practice","Hematology","Surgical Oncology","Nuclear Medicine","Occupational therapist","Cardiac Electrophysiology","Speech Language Pathologist","Audiologist (billing independently)","Multispecialty Clinic/Group Practice","Physician Assistant","Rheumatology","Neurosurgery","Interventional Radiology","Psychologist (billing independently)","Unknown Physician Specialty Code","Urology","Internal Medicine","Peripheral Vascular Disease","Radiation Oncology","Pulmonary Disease","Oral Surgery (dentists only)","Clinical Laboratory","Gastroenterology","Mass Immunization Roster Biller","Hand Surgery","Ophthalmology","Medical Oncology","Chiropractic","Addiction Medicine","Nephrology","Psychiatry","General Practice","Public Health Welfare Agency","Hematology/Oncology","Emergency Medicine","Interventional Pain Management","Neurology","Certified Clinical Nurse Specialist","Neuropsychiatry","General Surgery","Cardiac Surgery","Ambulatory Surgical Center","Sports Medicine","Hospice and Palliative Care","Orthopedic Surgery","Optometry","Mammographic Screening Center","Physical Therapist","All Other Suppliers","Vascular Surgery","Registered Dietician/Nutrition Professional","Critical Care (Intensivists)","CRNA","Obstetrics/Gynecology","Slide Preparation Facility","Geriatric Medicine","Podiatry","Cardiology","Diagnostic Radiology"]

*// Load the graph from a json file into PGX*

directedG = session.readGraphWithProperties("bipartite.ej.json")

g=directedG.undirect()

directedG.destroy()

*// Compile a program that runs a personalized pagerank algorithm taking a set of vertices as the source*

pprFromKind = session.compileProgram("personalized\_pagerank\_from\_kind.gm")

*// We create a VertexProperty that is going to hold the actual pagerank*

pgRank = g.createVertexProperty(PropertyType.DOUBLE, "pgRank")

"Iterating over all specialities"

**for** (item **in** specialities) {

println (item)

*// We set the value that we want to filter out*

*// We run the personlized pagerank algorithm*

pprFromKind.run(g, item, g.getVertexProperty("speciality"),0.001, 0.85, 1000, pgRank)

*// We create a subgraph to filter out the nodes that we don't need to evaluate*

subgraph = g.filter(**new** NodeFilter("node.speciality != '"+item+"' AND node.speciality != 'HCPCS'"))

*// We get the top pagerank values from the subgraph*

resultSet = subgraph.queryPgql("SELECT suspect, suspect.${pgRank.getName()} AS pagerank MATCH (suspect) ORDER BY suspect.${pgRank.getName()} DESC LIMIT 10")

resultSet.print();

*// We clean up the subgraph since we don't need it anymore*

subgraph.destroy()

}

[1] <https://docs.oracle.com/cd/E56133_01/2.7.0/use-cases/healthcare-fraud/index.html>

1. damping factor [↑](#footnote-ref-2)
2. maximum error difference [↑](#footnote-ref-3)
3. maximum number of iterations [↑](#footnote-ref-4)