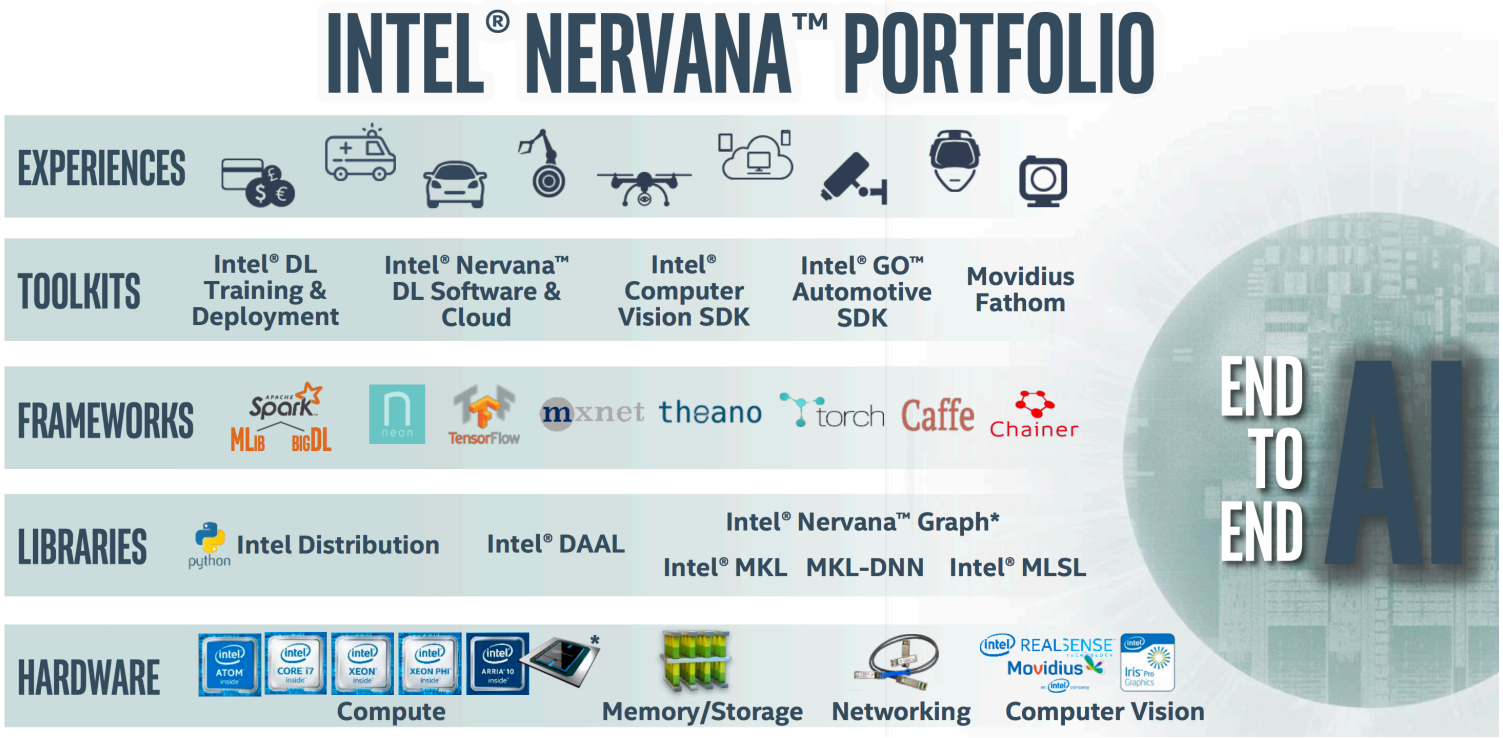
Distributed end-to-end drug similarity analytics and visualization workflow

Anahita Bhiwandiwalla – Intel Corporation
Dina Suehiro – Intel Corporation

Who we are?


INTEL® NERVANA™ PORTFOLIO
EXPERIENCES
TOOLKITS
Intel® DL Training & Deployment
Intel® Nervana™ DL Software & Cloud
Intel® Computer Vision SDK
Intel® GO™ Automotive SDK
Movidius Fathom
FRAMEWORKS
Spark
MLlib
BIGDL
neon
TensorFlow
mxnet
theano
torch
Caffe
Chainer
LIBRARIES
python
Intel Distribution
Intel® DAAL
Intel® Nervana™ Graph*
Intel® MKL
MKL-DNN
Intel® MLSL
HARDWARE
Compute
Memory/Storage
Networking
Computer Vision
END TO END AI

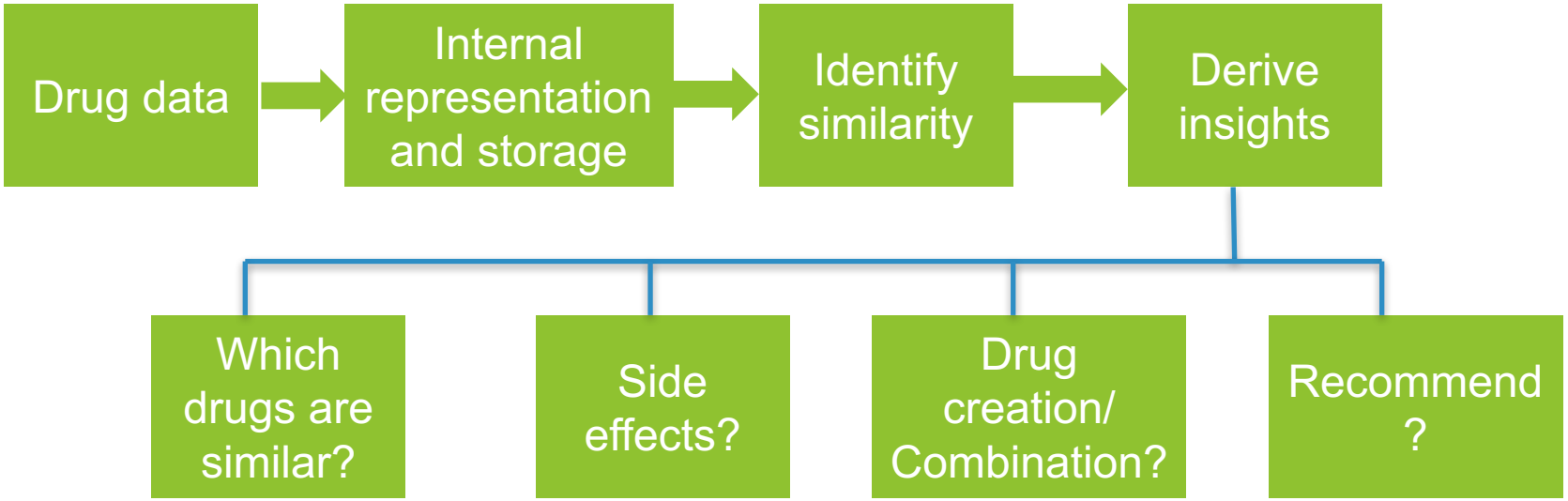
Agenda

- Chemical Similarity
- Use-case overview
- Drug data & SMILES
- Tanimoto Co-efficient
- Graph Analytics
- Apache Spark
- Spark-tk library
- Superset visualization
- Demo
- Next steps

Chemical Similarity

- Similarity of chemical elements, molecules or chemical compounds
- Structural or functional
- **Usages:**
  - Predicting properties of chemical compounds
  - Designing chemicals with predefined set of properties
  - Drug design

Use-case overview



SMILES

Simplified **M**olecular **I**nput **L**ine **E**ntry **S**ystem

– Typographical method using printable characters for entering and representing molecules and reactions

SMILES	Name	SMILES	Name
CC	ethane	[OH3+]	hydronium ion
O=C=O	carbon dioxide	[2H]O[2H]	deuterium oxide
C#N	hydrogen cyanide	[235U]	uranium-235
CCN(CC)CC	triethylamine	F/C=C/F	E-difluoroethene
CC(=O)O	acetic acid	F/C=C\F	Z-difluoroethene
C1CCCCC1	cyclohexane	N[C@@H](C)C(=O)O	L-alanine
c1ccccc1	benzene	N[C@H](C)C(=O)O	D-alanine

Drug data

Raw data format

id	name	SMILES	chembl_id	chebi_id	DrugBank	PubChem	MeSH_ID	KEGG_drug
0	levodopa	C1=CC(=C(C=C1C[C@@H](C(=O)O)N)O)O	CHEMBL1009	15765	DB01235	6047	D007980	D00059
1	nipecotic acid	C1CC(CNC1)C(=O)O	CHEMBL277498	116931	NA	4498	C030278	NA
2	betaxolol	CC(C)NCC(COC1=CC=C(C=C1)CCOCC2CC2)O	CHEMBL423	3082	DB00195	2369	D015784	NA
3	oxybutynin	CCN(CC)CC#CCOC(=O)C(C1CCCCC1)(C2=CC=CC=C2)O	CHEMBL1231	7856	DB01062	4634	C005419	D00465
4	indapamide	CC1CC2=CC=CC=C2N1NC(=O)C3=CC(=C(C=C3)Cl)S(=O)(=O)N	CHEMBL406	100619	DB00808	3702	D007190	D00345
5	anabasine	C1CCNC(C1)C2=CN=CC=C2	CHEMBL280963	28986	NA	2181	D000691	NA
6	tamoxifen	CC/C(=C(\C1=CC=CC=C1)/C2=CC=C(C=C2)OCCN(C)C)/C3=CC=CC=C3	CHEMBL83	41774	DB00675	2733526	D013629	D00966
7	eucatropine	CC1CC(CC(N1C)(C)C)OC(=O)C(C2=CC=CC=C2)O	CHEMBL1618660	NA	NA	7534	NA	NA
8	flucytosine	C1=NC(=O)NC(=C1F)N	CHEMBL1463	5100	DB01099	3366	D005437	D00323
9	terazosin	COC1=C(C=C2C(=C1)C(=NC(=N2)N3CCN(CC3)C(=O)C4CCCO4)N)OC	CHEMBL611	9445	DB01162	5401	C041226	NA
10	citiolone	CC(=O)NC1CCSC1=O	CHEMBL2104457	NA	NA	14520	C000486	NA
11	nitrofurantoin	C1C(=O)NC(=O)N1/N=C\C2=CC=C(O2)[N+](=O)[O-]	CHEMBL572	116592	DB00698	5353830	D009582	D00439
12	lomustine	C1CCC(CC1)NC(=O)N(CCCl)N=O	CHEMBL514	110898	DB01206	3950	D008130	D00363
13	meptazinol	CCC1(CCCCN(C1)C)C2=CC(=CC=C2)O.Cl	CHEMBL314437	239237	NA	65483	D008621	NA
14	rilmenidine	C1CC1C(C2CC2)NC3=NCCO3	CHEMBL289480	160422	NA	68712	C032302	NA
15	bufexamac	CCCCOC1=CC=C(C=C1)CC(=O)NO	CHEMBL94394	253408	NA	2466	D002019	NA
16	mebhydrolin	CN1CCC2=C(C1)C3=CC=CC=C3N2CC4=CC=CC=C4	CHEMBL1625607	NA	NA	22530	C005139	NA
18	piperidolate	CCN1CCCC(C1)OC(=O)C(C2=CC=CC=C2)C3=CC=CC=C3.Cl	CHEMBL1474977	998861	NA	8520	C010293	NA
20	metamizole sodium	CC1=C(C(=O)N(N1C)C2=CC=CC=C2)N(C)CS(=O)(=O)[O-].[Na+]	CHEMBL487894	59033	DB04817	522325	D004177	NA
21	sulfamethoxazole	CC1=CC(=NO1)NS(=O)(=O)C2=CC=C(C=C2)N	CHEMBL443	9332	DB01015	5329	D013420	D00447

Tanimoto Co-efficient

- Range from 0 to 1; 1 being most similar
- Similarity ratio over bitmaps
- Each bit of fixed-size array represents presence or absence of a characteristic

Presented in mathematical terms, if samples *X* and *Y* are bitmaps, X_i is the *i*th bit of *X*, and $\wedge$, $\vee$ are bitwise *and*, *or* operators respectively, then the similarity ratio T_s is

$$T_s(X, Y) = \frac{\sum_i (X_i \wedge Y_i)}{\sum_i (X_i \vee Y_i)}$$

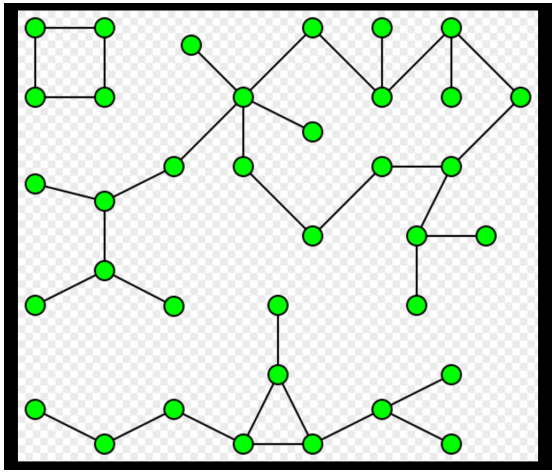
https://en.wikipedia.org/wiki/Jaccard_index#Tanimoto_similarity_and_distance

Graph Analytics

- Mathematical structures used to model pairwise relations between objects
- Vertices/nodes/points connected by edges/arcs/lines
- Undirected edges: No distinction between the two vertices associated with each edge
- Weighted edges: Each edge has a value/weight associated with it

Connected components

- Form subgraphs in which any two vertices are connected by edges



Clusters of similar drugs?

https://en.wikipedia.org/wiki/Connected_component_(graph_theory)

Clustering co-efficient

Numerical metrics for drug graph?

- Measure of the degree to which nodes in a graph tend to cluster together
- **Global** clustering co-efficient – overall indication of the clustering in the network
- **Local** clustering co-efficient – indication of the embeddedness of single nodes

Label propagation

- Finding communities in data
  - Communities structure: if the nodes of the graph can be easily grouped into densely connected sets of nodes(potentially overlapping)
- **Principle:** Pairs of nodes are more likely to be connected if they are both members of the same community(ies)
- Efficient run time
- Requires no apriori information to run

Closeness Centrality

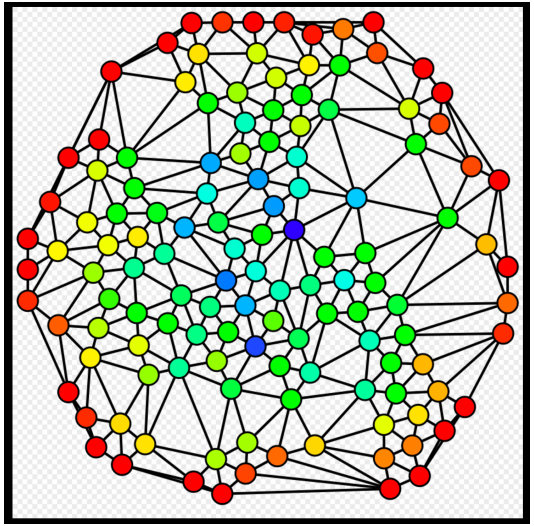
What drugs are most similar to others?

- Average length of the shortest path between the node and all other nodes
- More central a node, the closer it is to all other nodes

Betweenness centrality

Which drugs stand between each other?

- Quantify the **number of times a node acts as bridge** along the shortest path between two nodes
- Vertices that have a high probability to occur on a randomly chosen shortest path have high betweennness



https://en.wikipedia.org/wiki/Betweenness_centrality

Degree Centrality

- Identify most important vertices in a graph
- Number of edges incident upon a node
- Immediate risk of a node for catching network flow
- Directed graphs: indegree and outdegree

Why Spark?

- Spark Core
- Spark SQL
- GraphX
- Mllib
- Spark Streaming

Spark-tk Introduction

- Open source library enhancing the Spark experience
- APIs for feature engineering, ETL, graph construction, machine learning, scoring
- Abstraction level familiar to data scientists; removes complexity of parallel processing
- Lower level Spark APIs seamlessly exposed
- Easy to build apps plugging in other tools/libraries

https://github.com/trustedanalytics/spark-tk

http://trustedanalytics.github.io/spark-tk/

Spark-tk components

- **Frames:**
  - Scalable data frame representation
  - More intuitive than low level HDFS file and Spark RDD/DataFrame/DataSet formats; schema inference
  - APIs to manipulate the data frames for feature engineering and exploration, such as joins and aggregations
  - Run user-defined transformations and filters using distributed processing
  - Input to our models
  - Easy to go from Spark-tk Frames to Spark representations

Spark-tk components

- **Graphs**
  - Scalable graph representation based on Frames for vertices and edges
  - In house distributed algorithms for graph analytics using GraphX
  - Supports importing/exporting to OrientDB's scalable graph databases – visualize, real-time graph querying
  - Store massive graphs

Spark-tk components

- **Machine Learning & Streaming**
  - Time series analysis
  - Recommender systems
  - Topic Modeling
  - Clustering
  - Classification/Regression
  - Image processing
  - Scikit learn Models
  - Streaming via Scoring engine

RDKit

- Open source Cheminformatics and Machine Learning Software written in C++ and Python
- SMILES String of a drug → RDKit's Mol object
- Compute Similarity on two Mol objects:

```
DataStructs.FingerprintSimilarity(FingerprintMols.FingerprintMol(mol1),
                                  FingerprintMols.FingerprintMol(mol2))
```

http://www.rdkit.org/

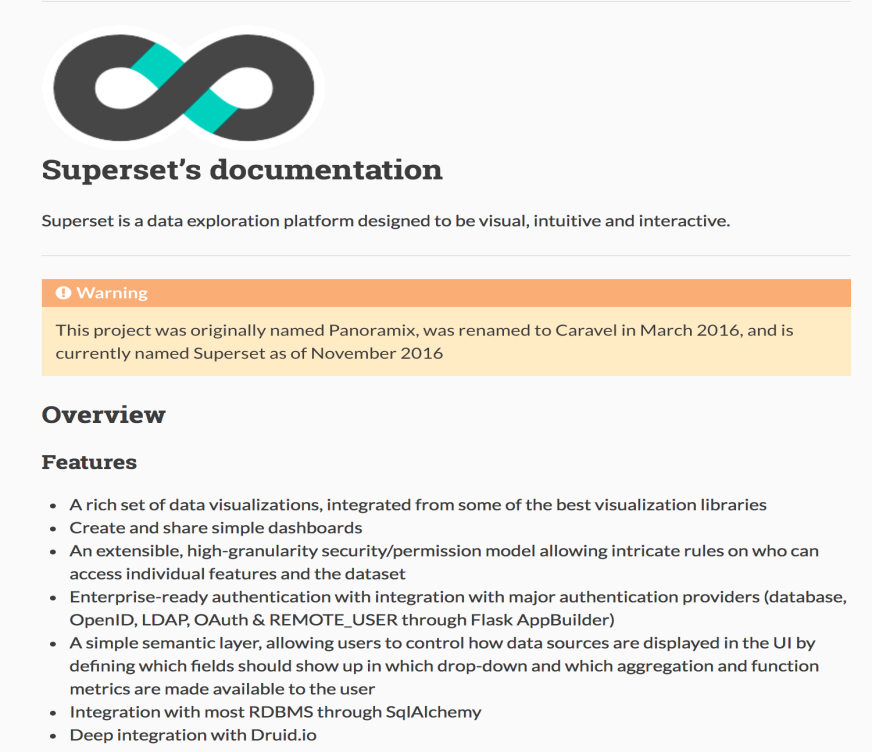
http://www.rdkit.org/docs/api/index.html

Spark-tk backend

Import/export from:

- CSV
- Hbase
- JDBC
- Hive
- JSON
- Pandas
- OrientDB
- Frame to/from Spark objects

Visualization with Superset



Superset's documentation

Superset is a data exploration platform designed to be visual, intuitive and interactive.

Warning

This project was originally named Panoramix, was renamed to Caravel in March 2016, and is currently named Superset as of November 2016

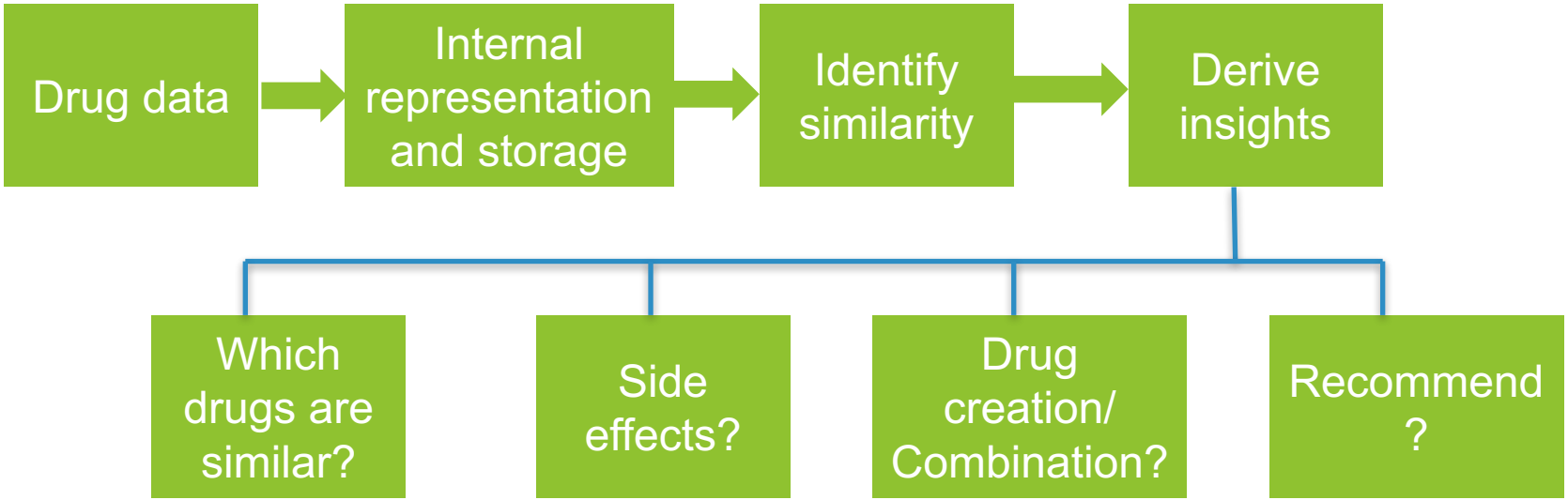
Overview

Features

- A rich set of data visualizations, integrated from some of the best visualization libraries
- Create and share simple dashboards
- An extensible, high-granularity security/permission model allowing intricate rules on who can access individual features and the dataset
- Enterprise-ready authentication with integration with major authentication providers (database, OpenID, LDAP, OAuth & REMOTE_USER through Flask AppBuilder)
- A simple semantic layer, allowing users to control how data sources are displayed in the UI by defining which fields should show up in which drop-down and which aggregation and function metrics are made available to the user
- Integration with most RDBMS through SqlAlchemy
- Deep integration with Druid.io

https://github.com/airbnb/superset; http://airbnb.io/superset/

Use-case overview



Demo

Next steps

- Further investigate Side effects and implications of drug interactions
- Combine with patient Electronic Health Records and train models
- Drug recommendations/suggestions

Thank You.

https://github.com/trustedanalytics/spark-tk

http://trustedanalytics.github.io/spark-tk/

Backup – Jupyter Notebook

Drug Similarity Analytics Demo

This notebook demonstrates the process of importing and cleaning drug data, using rdkit to calculate the tanimoto coefficient to compare each drug, creating a sparktk graph, running degree centrality, and then exporting the data to a table in a postgres database.

We start out by importing the rdkit and sparktk libraries:

In [1]:
```
# rdkit imports for using SMILES index to get molecule fingerprint and tanimoto calculation
from rdkit import Chem
from rdkit import DataStructs
from rdkit.Chem.Fingerprints import FingerprintMols
import sys
sys.path.append("/home/dmsuehir/anaconda2/envs/my-rdkit-env/")

# import sparktk and create a sparktk context, with postgresql paths
import sparktk
submit_args = "--jars /home/dmsuehir/.m2/repository/postgresql/postgresql/8.4-702.jdbc4/postgresql-8.4-702.jdbc4.jar "
\
          "--driver-class-path /home/dmsuehir/.m2/repository/postgresql/postgresql/8.4-702.jdbc4/postgresql-8.4-702.j
dbc4.jar"
spark_conf = "/home/dmsuehir/superset_demo/sparktk.conf"
tc = sparktk.TkContext(pyspark_submit_args=submit_args,extra_conf_file=spark_conf, extra_conf_dict={"spark.dynamicAllo
cation.enabled": "true", "spark.sql.shuffle.partitions": 200})
```

Import data into frames

We start out with raw data in a `drugs.csv` file, which we will import into a `frame`, using sparktk. The frame `inspect()` method gives a preview of a few columns of data.

In [2]:
```
drugs_csv_path = "hdfs://10.7.151.77/user/atkuser/drugs.csv"
nodes_frame = tc.frame.import_csv(drugs_csv_path, delimiter="\t", header=True)
nodes_frame.schema
```

Out[2]:
```
[('id', int),
 ('name', str),
 ('CAS', str),
 ('chembl_id', str),
 ('chebi_id', str),
 ('DrugBank', str),
 ('PubChem', str),
 ('MeSH_ID', str),
 ('KEGG_drug', str),
 ('SMILES', str),
 ('InChI', str)]
```

SPARK SUMMIT 2017

In [3]:
```
# Drop columns that we don't need and then inspect data
nodes_frame.drop_columns(["CAS", "chembl_id", "chebi_id", "DrugBank", "PubChem", "MeSH_ID", "KEGG_drug", "InChI"])
nodes_frame.inspect()
```

Out[3]:
```
[#]  id  name
=======================
[0]   0  vanoxerine
[1]   1  nipecotic acid
[2]   2  betaxolol
[3]   3  oxybutynin
[4]   4  indapamide
[5]   5  5248896
[6]   6  tamoxifen
[7]   7  halcinonide
[8]   8  flucytosine
[9]   9  terazosin

[#]  SMILES
================================================================================
[0]  Cl=CC=C(C=Cl)/C(=N/NC2=C(C=C(C=C2)[N+](=O)[O-])[N+](=O)[O-])/CCl
[1]  ClCC(CNCl)C(=O)O
[2]  CC(C)NCC(COC1=CC=C(C=Cl)CCOCC2CC2)O
[3]  CCN(CC)CC#CCOC(=O)C(C1CCCCC1)(C2=CC=CC=C2)O
[4]  CC1CC2=CC=CC=C2N1NC(=O)C3=CC(=C(C=C3)Cl)S(=O)(=O)N
[5]  Cl(C(C([N-]C(ClCl)Cl)C(=O)O)Cl)N.Cl(C(=C(C(=C([N-]1)Cl)Cl)N)Cl)C(=O)O.[NH2-].O.O.[Mn].[Mn].[Mn+2]
[6]  CC/C(=C(\Cl=CC=CC=Cl)/C2=CC=C(C=C2)OCCN(C)C)/C3=CC=CC=C3
[7]  C[C@]12CCC(=O)C=C1CC[C@@H]3[C@@]2([C@H](C[C@]4([C@H]3C[C@@H]5[C@]4(OC(O5)(C)C)C(=O)CCl)C)O)F
[8]  Cl=NC(=O)NC(=ClF)N
[9]  COC1=C(C=C2C(=Cl)C(=NC(=N2)N3CCN(CC3)C(=O)C4CCCO4)N)OC
```

Calculating the similarity between drugs

Next, we are going to compare the similarity between drugs by calculating the tanimoto coefficient. First, we will create an frame of edges which will connect one drug to another (representing the two drugs that we will compare). For simplicity, we are importing this data from a file that has already been saved off with the top 20 most similar connections for each drug.

In [4]:
```
edges_csv_path = "hdfs://10.7.151.77/user/atkuser/drug-edges-20.csv"
schema = [("source_id", int), ("dest_id", int)]
edges_frame = tc.frame.import_csv(edges_csv_path, schema=schema, header=True, delimiter="\t")
edges_frame.inspect()
```

Out[4]:

[#]	source_id	dest_id
[0]	190	638
[1]	190	1260
[2]	190	100
[3]	190	916
[4]	190	139
[5]	190	228
[6]	190	201
[7]	190	62
[8]	190	935
[9]	190	77

In [5]:
```
print "Nodes: {0}".format(nodes_frame.count())
print "Edges: {0}".format(edges_frame.count())
```

```
Nodes: 1278
Edges: 25560
```

Next, we join the `nodes_frame` with the `edges_frame` so that we have one frame with the drug information (drug name and SMILES index) for the source and destination (the two drugs that we are comparing).

In [6]:
```
# join the edges frame with nodes to get the smiles index for the source/target
joined = nodes_frame.join_right(edges_frame, left_on="id", right_on="source_id")
joined = nodes_frame.join_right(joined, left_on="id", right_on="dest_id")
joined.rename_columns({"name_R":"source_name", "SMILES_R":"source_SMILES","name_L":"target_name","SMILES_L":"target_SM
ILES"})
joined.inspect()
```

Out[6]:
```
[#]  target_name  target_SMILES
=============================================================================
[0]  5224221      Cl=CC(=C(C=C1C2=CSC(=N2)C3=CC4=C(C=CC(=C4)Br)OC3=O)Cl)Cl
[1]  5224221      Cl=CC(=C(C=C1C2=CSC(=N2)C3=CC4=C(C=CC(=C4)Br)OC3=O)Cl)Cl
[2]  5224221      Cl=CC(=C(C=C1C2=CSC(=N2)C3=CC4=C(C=CC(=C4)Br)OC3=O)Cl)Cl
[3]  decitabine   Cl[C@@H]([C@H](O[C@H]1N2C=NC(=NC2=O)N)CO)O
[4]  decitabine   Cl[C@@H]([C@H](O[C@H]1N2C=NC(=NC2=O)N)CO)O
[5]  decitabine   Cl[C@@H]([C@H](O[C@H]1N2C=NC(=NC2=O)N)CO)O
[6]  decitabine   Cl[C@@H]([C@H](O[C@H]1N2C=NC(=NC2=O)N)CO)O
[7]  decitabine   Cl[C@@H]([C@H](O[C@H]1N2C=NC(=NC2=O)N)CO)O
[8]  decitabine   Cl[C@@H]([C@H](O[C@H]1N2C=NC(=NC2=O)N)CO)O
[9]  decitabine   Cl[C@@H]([C@H](O[C@H]1N2C=NC(=NC2=O)N)CO)O

[#]  source_name
================
[0]  merbromin
[1]  oxaprozin
[2]  nizatidine
[3]  azacitidine
[4]  rilmenidine
[5]  minoxidil
[6]  8-azaguanine
[7]  idoxuridine
[8]  trifluridine
[9]  riboflavin

[#]  source_SMILES
=============================================================================
[0]  Cl=CC=C(C(=Cl)C2=C3C=C(C(=O)C=C3OC4=C(C(=C(C=C24)Br)[O-])[Hg])Br)C(=O)[O-].O.O.O.O.[Na+].[Na+]
[1]  Cl=CC=C(C=Cl)C2=C(OC(=N2)CCC(=O)O)C3=CC=CC=C3
[2]  CN/C(=C\[N+](=O)[O-])/NCCSCCl=CSC(=Nl)CN(C)C
[3]  Cl=NC(=NC(=O)N1[C@H]2[C@@H]([C@@H]([C@H](O2)CO)O)O)N
[4]  ClCC1C(C2CC2)NC3=NCCO3
```

Now that we have a frame (called `joined`) that has all the data that we need in order to compare drugs, we are defining a function (`calculate_tanimoto`) that uses the `rdkit` libraries to calculate the tanimoto coefficient from two SMILES strings. We are then adding a new column to the frame using the `calculate_tanimoto()` function. A column with the rounded tanimoto coefficient is also added to the frame, to be used later with generating a histogram.

In [7]:
```
# define function returns the tanimoto coefficient based on the SMILES index for two drugs
def calculate_tanimoto(smiles1, smiles2):
    mol1 = Chem.MolFromSmiles(smiles1)
    mol2 = Chem.MolFromSmiles(smiles2)
    return DataStructs.FingerprintSimilarity(FingerprintMols.FingerprintMol(mol1),FingerprintMols.FingerprintMol(mol2)
)

# add a column with the calculated tanimoto coefficient
joined.add_columns(lambda row: calculate_tanimoto(row.source_SMILES, row.target_SMILES), ("tanimoto", float))
joined.add_columns(lambda row: round(row.tanimoto,1), ("rounded_tanimoto", float))

joined.inspect(20, columns=["source_name", "target_name", "tanimoto", "rounded_tanimoto"])
```

Out[7]:

[##]	source_name	target_name	tanimoto	rounded_tanimoto
[0]	merbromin	5224221	0.442685243825	0.4
[1]	oxaprozin	5224221	0.302759134974	0.3
[2]	nizatidine	5224221	0.364485981308	0.4
[3]	azacitidine	decitabine	0.833558863329	0.8
[4]	rilmenidine	decitabine	0.405829596413	0.4
[5]	minoxidil	decitabine	0.329758713137	0.3
[6]	8-azaguanine	decitabine	0.363150867824	0.4
[7]	idoxuridine	decitabine	0.525720164609	0.5
[8]	trifluridine	decitabine	0.504347826087	0.5
[9]	riboflavin	decitabine	0.304016620499	0.3
[10]	zalcitabine	decitabine	0.544589774078	0.5
[11]	zidovudine	decitabine	0.429609445958	0.4
[12]	promethazine	prochlorperazine	0.75415282392	0.8
[13]	Prestwick-685	prochlorperazine	0.307639836289	0.3
[14]	trazodone	prochlorperazine	0.326105810928	0.3
[15]	perphenazine	prochlorperazine	0.979651162791	1.0
[16]	moracizine	prochlorperazine	0.612798264642	0.6
[17]	clozapine	prochlorperazine	0.304273504274	0.3
[18]	diazoxide	prochlorperazine	0.312984496124	0.3
[19]	mesoridazine	prochlorperazine	0.624309392265	0.6

Graph analytics using degree centrality

In order to run graph algorithms using sparktk, we'll need to create a `graph` object using a frame of nodes and a frame of edges. We will use the original `nodes_frame` to create the graph, and then create a new frame of `graph_edges`. The csv being imported to create the `graph_edges` frame has columns for `src`, `dst`, and `tanimoto` and is filtered to only include connections where the tanimoto coefficient is greater than 0.4.

In [8]:
```
edge_schema = [("src", int),("dst", int),("tanimoto",float)]
graph_edges = tc.frame.import_csv("hdfs://10.7.151.77/user/atkuser/drug-edges-gt-4.csv", schema=edge_schema)
graph_edges.inspect()
```

Out[8]:
```
[#]  src   dst  tanimoto
==============================
[0]   631   31  0.494117647059
[1]    32   31     0.47265625
[2]   234   31   0.40782122905
[3]   634   31           0.44
[4]   834   31  0.447058823529
[5]    35   31  0.476767676768
[6]   236   31  0.443264393516
[7]  1036   31  0.446569178853
[8]    37   31   0.41488162345
[9]  1237   31  0.412100456621
```

In [9]:
```
# create graph
graph = tc.graph.create(nodes_frame, graph_edges)

# run degree centrality algorithm, and then sort frame to find drugs with more high tanimoto connections
result = graph.degree_centrality("out")
result.sort("degree_centrality", False)
result.inspect(20, columns=["name", "degree_centrality"])
```

Out[9]:
```
[##]  name                    degree_centrality
================================================
[0]   isometheptene              0.980422866092
[1]   oxamic acid                0.978073610023
[2]   3-nitropropionic acid      0.977290524667
[3]   pentolonium                0.972592012529
[4]   heptaminol                 0.969459671104
[5]   carbachol                  0.960062646829
[6]   mephentermine              0.946750195771
[7]   mevalolactone              0.942051683634
[8]   nipecotic acid             0.939702427565
[9]   paracetamol                0.935003915427
[10]  bemegride                  0.926389976507
```

Exporting data to postgres and creating tables in superset

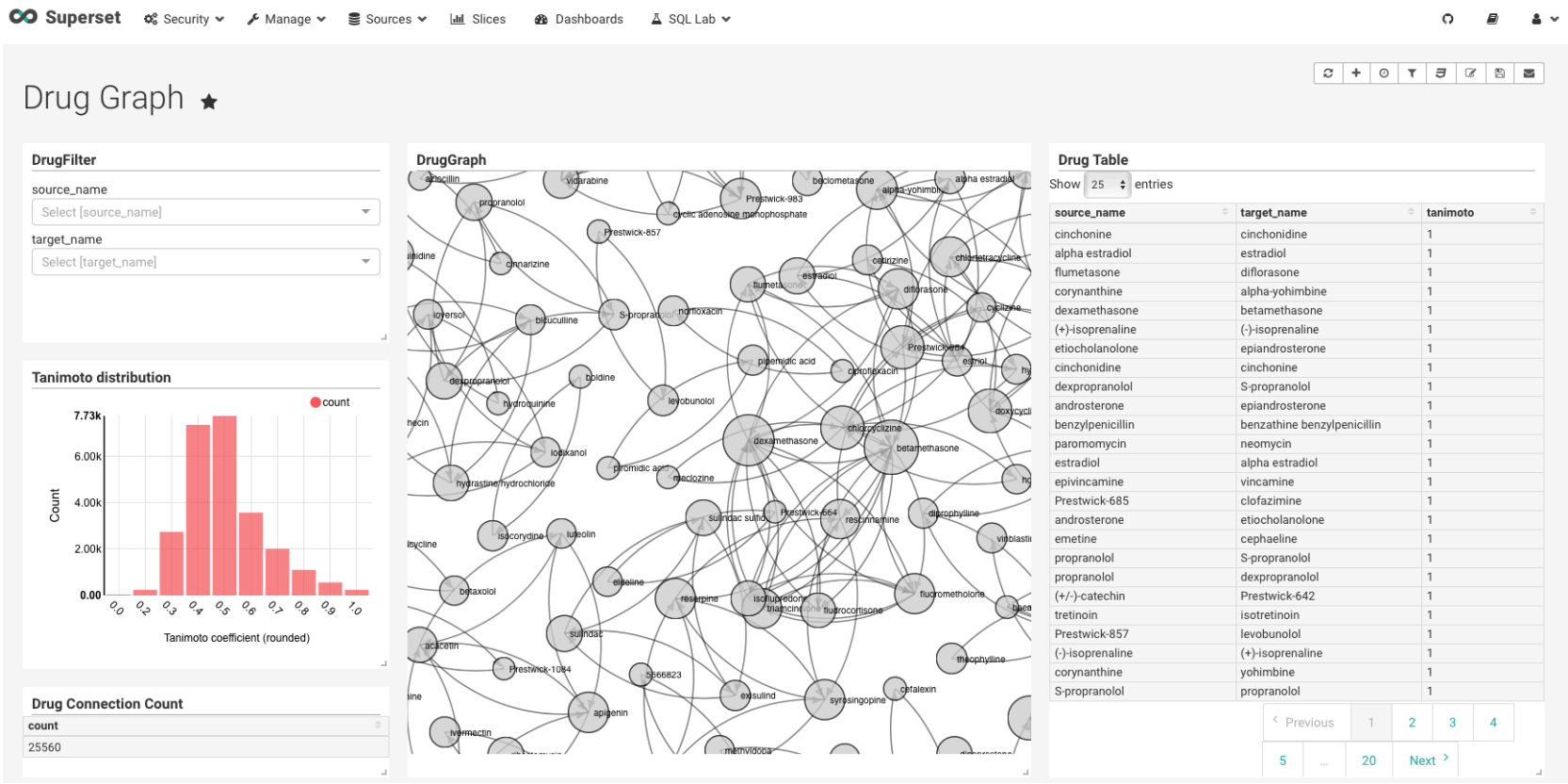
In order to visualize the data in superset, we need to export the data to a database. sparktk has methods that make it easy to export to database such as postgres, which is being used in this demo. Once the frame has been exported to postgres, we then use superset's python library to add the table so that the data can be viewed from the superset dashboard.

In [10]:
```
%%capture
import superset
```

In [11]:
```
def create_table(frame, database_name, table_name, table_description=""):
    # export train frame to postgres, and create table in superset
    db_connection_str = "jdbc:postgresql://localhost/{0}?user=*&password=*".format(database_name)
    frame.export_to_jdbc(db_connection_str, table_name)
    sql_table = superset.models.SqlaTable

    tbl = superset.db.session.query(sql_table).filter_by(table_name=table_name).first()

    if not tbl:
        tbl = sql_table(table_name=table_name)

    # copy database info from the existing table - replace this with getting Database
    db = superset.db.session.query(superset.models.Database).filter_by(database_name=database_name).first()

    # set table properties
    tbl.database = db
    tbl.description = table_description
    tbl.is_featured = True

    # merge and commit table
    superset.db.session.merge(tbl)
    superset.db.session.commit()
    tbl.fetch_metadata()
```

In [12]:
```
create_table(joined, database_name="postgres", table_name="drug_graph2")
```

Backup – Superset Dashboard



Drug Graph ★

DrugFilter

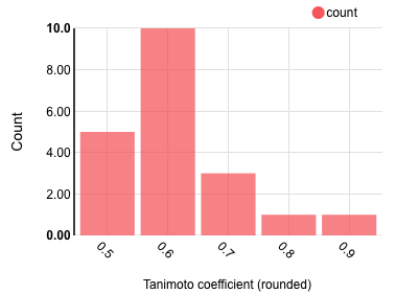
source_name

× levodopa

target_name

Select [target_name]

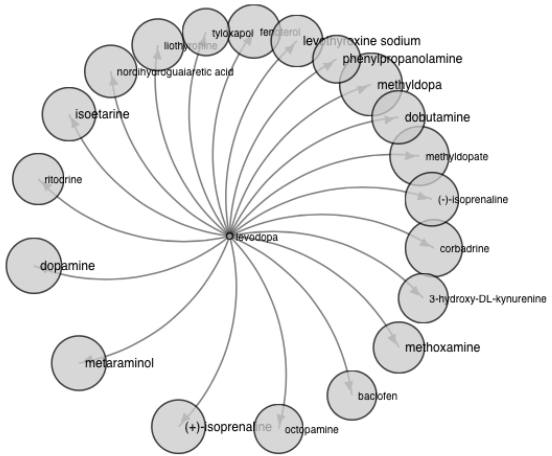
Tanimoto distribution



Drug Connection Count

count
20

DrugGraph



Drug Table

Show 25 entries

source_name	target_name	tanimoto
levodopa	methyldopa	0.898181818181818
levodopa	methyldopate	0.781645569620253
levodopa	corbadrine	0.72318339100346
levodopa	dopamine	0.676113360323887
levodopa	metaraminol	0.653710247349823
levodopa	(-)-isoprenaline	0.628289473684211
levodopa	(+)-isoprenaline	0.628289473684211
levodopa	fenoterol	0.625
levodopa	isoetarine	0.622093023255814
levodopa	dobutamine	0.616393442622951
levodopa	nordihydroguaiaretic acid	0.592982456140351
levodopa	liothyronine	0.581578947368421
levodopa	levothyroxine sodium	0.58005249343832
levodopa	ritodrine	0.565916398713826
levodopa	methoxamine	0.5625
levodopa	3-hydroxy-DL-kynurenine	0.526992287917738
levodopa	baclofen	0.518633540372671
levodopa	octopamine	0.512820512820513
levodopa	phenylpropanolamine	0.48014440433213
levodopa	tyloxapol	0.466887417218543

Previous 1 Next