DREAMing of big data and scalable machine learning: Predicting kinase binding with matrix factorization

Jonathan King, Paul Francoeur, Amanda Kowalczyk, Shashank Rajashekar, Chakra Chennubhotla, David Ryan Koes
Departments of Computational and Systems Biology and Information Science, University of Pittsburgh

Assign4: Predicting Drug-Target Interactions with Large-Scale Matrix Factorization

Checkpoint Wednesday, October 31, 11:59pm
Due ~~Friday, Nov 9th~~ Monday, Nov 12, 11:59pm
Submit

In this assignment, your goal is to predict drug-target interactions using matrix factorization. We will consider the effects of chemical compounds ("drugs") on cell lines and biochemical assays ("targets") as a matrix. Only some of the elements of this matrix are available to us, and our goal is to predict the missing values.

Input

```
./assign4.py train.csv missing.csv output.csv
```

The first argument of your program (`train.csv`) is a CSV file (possibly stored on Google Cloud) where each line consists of `row,column,value` as shown below.

```
122839,0,4.72
49129,2,4.2
53115,2,4.36
2402,3,5.7
23327,3,4.07
45087,3,5.38
....
```

The rows are indices of compounds and the columns indices of targets.

The second argument (`missing.csv`) specifies the location in the matrix of the missing values you should predict. Each line consists of `row,column` indices.

You should write out the predictions to the specified output file using the same `row,column,value` format as the training data. Since each line includes the row and column indices, you may write these in any order.

Data

For training we are using a kinase-focused subset of ChEMBL. We are also including all the targets and compounds from stage 1 of the DREAM Drug-Kinase Binding Prediction Challenge. There are 811 kinase targets and 123,756 compounds, but not all targets or compounds have associated affinity values. The full dataset consists of 214,132 pChEMBL affinity values (meaning 99.8% of the drug-target matrix is missing). Your code will be evaluated on two train/test splits of the full dataset. A **Random** split where 10% of dataset is randomly selected for the test set, and a **Hard** set where 5% of the data is randomly selected and 5% is selected by removing entire compounds or targets.

In addition to the files passed on the commandline, you may use any of the following files (**not including the complete set**, which is provided for training your dream predictions on), all of which are available from the Google File System. The sim files provide precomputed similarites between proteins and between compounds. The remaining files are necessary for submitting to the DREAM challenge (provide names) or if you want to compute your own similarity metrics.

all.csv	**Complete dataset**
correct_randomtrain.csv	**Random training set**
correct_randommissing.csv	**Random test set (input)**
correct_randomtest.csv	**Random test set (output)**
correct_hardtrain.csv	**Hard training set**
correct_hardmissing.csv	**Hard test set (input)**
correct_hardtest.csv	**Hard test set (output)**
targ_sim.txt	Target-target similarities, indicated using numerical indices
correct_cmpd_sim.txt	Compound-compound simlarities, indicated using numerical indices
targ_uniprot.txt	Mapping from ChEMBL target identifier to UniProt name and numerical index
targ_seq.txt	Mapping from ChEMBL target identifier to sequence (probably won't use this)
correct_cmpd_index.txt	Mapping from ChEMBL compound id to numerical index
correct_cmpd_fingerprints.txt	Mapping from ChEMBL compound id to bit string of chemical fingerprint (probably won't use this)

DREAM Files

makedream.py	Script to generate submission file
round_1_template.csv	**DREAM provided template**
dream_all.csv	**Training set**
dream_missing.csv	**Values to predict**
dream_targ_sim.txt	Target-target similarities, indicated using numerical indices
dream_cmpd_sim.txt	Compound-compound simlarities, indicated using numerical indices
dream_targ_uniprot.txt	Mapping from ChEMBL target identifier to UniProt name and numerical index
dream_targ_seq.txt	Mapping from ChEMBL target identifier to sequence (probably won't use this)
dream_cmpd_index.txt	Mapping from ChEMBL compound id to numerical index
dream_cmpd_fingerprints.txt	Mapping from ChEMBL compound id to bit string of chemical fingerprint (probably won't use this)
cmpd_dream.txt	Mapping from ChEMBL compound id to DREAM compound names

Implementation

You may use any of the built-in Spark routines, including SVD and ALS, but our expectation is that you will implement some form of stochastic gradient descent (SGD) in order to get the best possible performance. As a simple starting point, consider this (non-stochastic and non-distributed) implementation which clearly shows how to translate the math into code.

A key challenge with this dataset is dealing with compounds/targets that have no or nearly no affinity data associated with them (empty rows and columns). One possible approach to deal with this issue is to first downsample the data to only those targets and compounds that have a reasonable amount of data, perform matrix factorization on this smaller dataset, and then use the provided target and compound similarity matrices to perform k-nn regression when predicting affinities for compound/target pairs that aren't in this reduced matrix.

Alternatively, you may want to evaluate a more integrated approach where the similarity matrices are integral to the factorization of the matrix.

Evaluation

Your code will be evaluated for its accuracy in predicting the missing values, as measured by RMSE (although keep in mind that, in practice, it is a good idea to consider additional metrics, such as correlation) and for performance. For grading, we will weight the accuracy of your predictions more heavily than the performance (although your code must finish within the time limit).

Submission

We will run your code similarily to the last assignment, only this time with 32 quad-core workers.

```
  gcloud dataproc clusters create grader --zone us-east1-b --region us-east1 --num-workers 32 --properties spark:spark.executor.heartbeatInterval=120,s
  gcloud dataproc jobs submit pyspark --region us-east1 --cluster grader a4.py -- gs://mscbio2065-data/randomtrain.csv gs://mscbio2065-data/randommissi
```

Note I needed to increase the default stack size to get ALS to run with our data. Your program must finish processing a file within 45 minutes or it will be killed. You should submit your code as a single python script. Your results, including all output produced by your program, will be emailed to you and your score and time displayed on the webpage using your Avatar name.

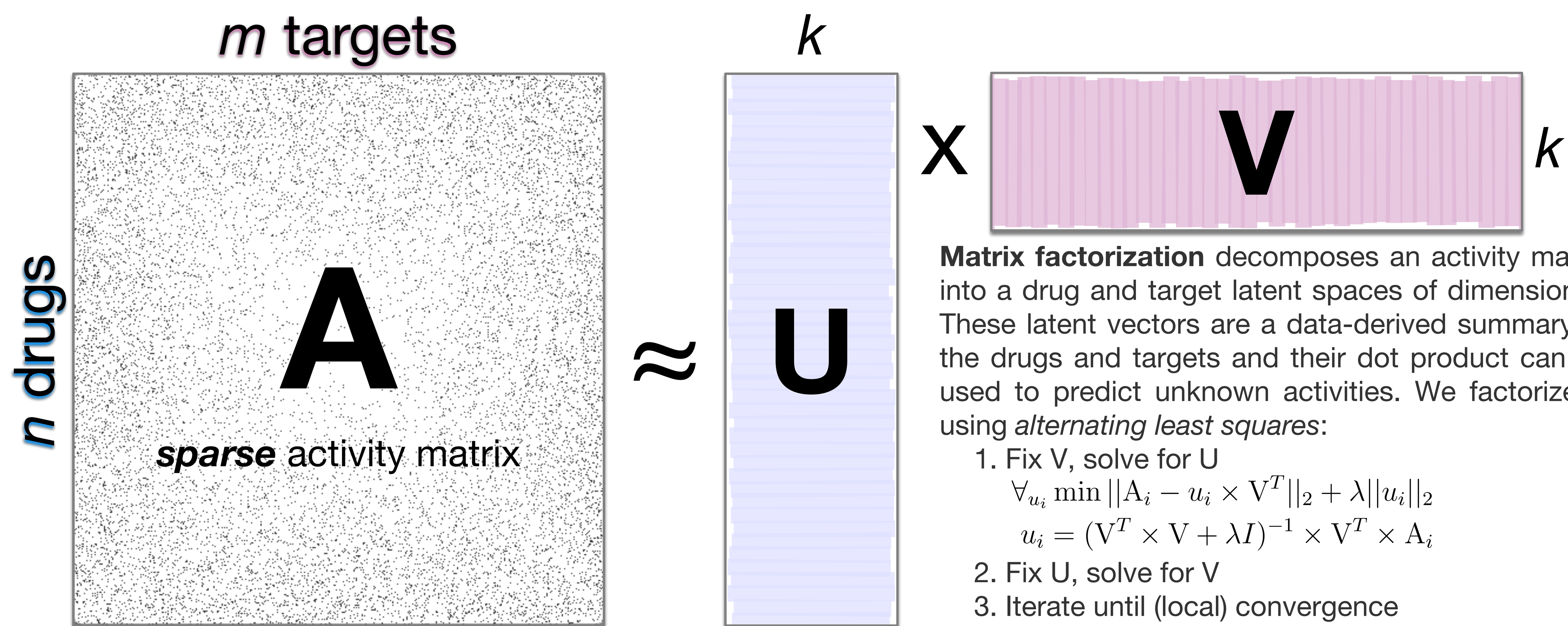
Grading

10%	Submit a non-trivial solution by the checkpoint date (e.g., ALS).
40%	Accuracy on Random.
40%	Accuracy on Hard.
10%	Performance (Speed).
10pt Bonus	Submit your best solution to the DREAM challenge

User ID (private):
Avatar Name (public):
Code: Choose File No file chosen
Submit

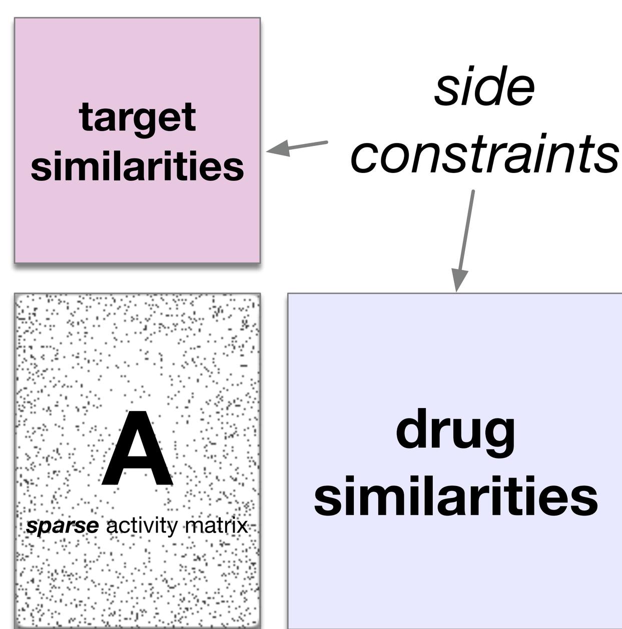
The IDG-DREAM Drug-Kinase Binding Prediction Challenge is an open exercise in predicting pan-assay kinase binding affinities. We used the 2018 DREAM challenge as the basis for an **assignment** in our course, *Scalable Machine Learning for Big Data Biology* (https://mscbio2065.wordpress.com).

Students used **Google Cloud** and **PySpark** to process a dataset of more than 125,000 kinase inhibitors spanning more than 800 targets extracted from ChEMBL. Inspired by recommender systems, like those that won the Netflix prize, students were tasked with applying **matrix factorization** to the problem and implementing a solution to the **cold start problem** to compensate for the sparsity of the data.



Matrix factorization decomposes an activity matrix into a drug and target latent spaces of dimension *k*. These latent vectors are a data-derived summary of the drugs and targets and their dot product can be used to predict unknown activities. We factorize A using *alternating least squares*:

1. Fix V, solve for U
   $$\forall_{u_i} \min ||\mathrm{A}_i - u_i \times \mathrm{V}^T||_2 + \lambda||u_i||_2$$
   $$u_i = (\mathrm{V}^T \times \mathrm{V} + \lambda I)^{-1} \times \mathrm{V}^T \times \mathrm{A}_i$$
2. Fix U, solve for V
3. Iterate until (local) convergence



The **cold start** problem arises when a drug or target has no activity information and so a meaningful latent vector cannot be constructed. **Side constraints** provide additional information that links related drugs and targets together. In our case, we used the chemical fingerprint similarity of the drugs and the sequence similarity of the targets. Various constraints can be added to the problem to enforce that similar drugs/targets have similar latent vectors. However, all student submissions used a simple **nearest neighbor initialization** approach to resolve missing data.

	Random			Hard		
	Time	RMSE	Pearson R	Time	RMSE	Pearson R
bmd	1325.234	0.757	0.837	1312.371	0.818	0.789
sparky	1557.055	0.810	0.813	1501.945	0.829	0.788
nottrivial	1505.468	0.819	0.809	1474.940	0.835	0.784
ALS-1nn (A)	1038.047	0.874	0.788	1022.554	0.890	0.764
Amylith	1326.458	0.862	0.798	1271.306	0.912	0.761
w-als-1nn	919.918	0.873	0.775	716.087	0.945	0.726
yo!	2058.818	1.473	0.754	2026.042	1.479	0.716
°_°	1403.050	0.975	0.720	1365.627	1.088	0.622
sub31	143.425	3.210	0.652	116.954	3.203	0.616
P	171.111	3.208	0.651	151.905	3.201	0.615
AP	174.418	2.313	0.618	151.113	2.304	0.583
SGD	69.101	1.277	0.428	61.831	1.350	0.334
ALS	288.333	1.312	0.428	282.319	1.291	0.306
S	187.145	4.666	0.141	157.259	5.813	0.085
Mean	27.239	1.373	0.000	27.340	1.328	0.000
S	X	X	X	X	X	X
ignore	X	X	X	X	X	X
Sparky	X	X	X	X	X	X

Results

Students did well on the held out test set constructed from ChEMBL, as shown by the correlations greater than 0.78 and root mean squared errors under 0.9 on the class leaderboard. However, **performance on the DREAM Challenge was abysmal.** The ChEMBL trained models failed to generalize. Our best correlation was 0.20 while the best overall in the competition was 0.54.

objectId	userId	rmse	pearson
9682320	David Koes (dkoes)	1.3187	0.2018
9682783	Amanda Kowalczyk (Amylith)	1.398	0.1584
9682826	Shashank Badavanahalli Rajashekar (shashanksira)	2.1258	0.1619
9682808	Jonathan K (jok120)	1.3754	0.0789

```python
import pyspark
import numpy as np
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql.functions import col, avg
import csv
from pyspark.sql import Row
import sys

if len(sys.argv) != 4:
    print("Incorrect arguments.\ntrain.csv, missing.csv, output.csv")
    sys.exit(1)
trainfile, testfile, outfile = sys.argv[1], sys.argv[2], sys.argv[3]

def tokenize_DF(r, reverse=False):
    items = r
    if reverse: # protein, drug
        return int(items[1]), (int(items[0]), float(items[2]))
    return int(items[0]), (int(items[1]), float(items[2]))

def tokenize_text(r, reverse=False, as_list=False):
    items = r.split()
    if reverse and not as_list: # protein, drug
        return int(items[1]), (int(items[0]), float(items[2]))
    if reverse and as_list:
        return int(items[1]), [(int(items[0]), float(items[2]))]
    if not reverse and not as_list:
        return int(items[0]), (int(items[1]), float(items[2]))
    if not reverse and as_list:
        return int(items[0]), [(int(items[1]), float(items[2]))]

def reduce_best_sim_knn(x, y, k=6):
    """x and y are both lists of type [(idx, similarity)...]. Returns z of same
        type. """
    if len(x) + len(y) <= k:
        return x + y # concat
    else:
        both = list(sorted(filter(lambda tup: tup[1] != 1, x + y), key=lambda tup: -tup[1]))
        return both[:k]

def get_sum_counts(rdd):
    """ Starts out as (drug, (protein, val))
        Returns (drug, (sum, count)) """
    rdd = rdd.map(lambda r: (r[0], (r[1][1], 1))).reduceByKey(
        lambda t0, t1: (t0[0] + t1[0], t0[1] + t1[1]))
    return rdd

def impute_knn(drug, target):
    """ For drug i and target j, use precomputed NN lookups and precomputed avgs
        to predict activity. Returns (i, j, activity)."""

    if drug in missing_drgs.value:
        # most_similar_drugs = [tup[0] for tup in drug_sims.value[drug]]
        most_similar_drugs = drug_sims.value[drug]
    else:
        most_similar_drugs = [(drug, 1.0)]

    if target in missing_pros.value:
        # most_similar_targs = [tup[0] for tup in targ_sims.value[target]]
        most_similar_targs = targ_sims.value[target]
    else:
        most_similar_targs = [(target, 1.0)]

    drug_sum, drug_cnt, targ_sum, targ_cnt = 0, 0, 0, 0
    drug_distances = np.sum([tup[1] for tup in most_similar_drugs])
    targ_distances = np.sum([tup[1] for tup in most_similar_targs])
    drg_avg, trg_avg = 0, 0
    for msd, sim in most_similar_drugs:
        if msd not in drug_sum_counts.value:
            msd = next(iter(drug_sum_counts.value))
        d_sum, d_cnt = drug_sum_counts.value[msd]
        ind_avg = d_sum / float(d_cnt)
        drg_avg += ind_avg * (sim / float(drug_distances))
    for mst, sim in most_similar_targs:
        t_sum, t_cnt = targ_sum_counts.value[mst]
        ind_avg = t_sum / float(t_cnt)
        trg_avg += ind_avg * (sim / float(targ_distances))


    average = (drg_avg * .85) + (trg_avg * .15)
    return drug, target, average

# Set up Spark context and dataframes
spark = pyspark.sql.SparkSession.builder.getOrCreate()
sc = spark.sparkContext

drugsim_file = 'gs://mscbio2065-data/correct_cmpd_sim.txt' #files located on the cloud
ptnsim_file = 'gs://mscbio2065-data/targ_sim.txt' #files located on the cloud

df_train = spark.read.csv(trainfile,header=False, inferSchema=True)
df_test = spark.read.csv(testfile,header=False, inferSchema=True)

# Compute Missing drugs and create RDDs for (i,j) and (j,i)
pro_set = set(range(812)) # proteins, c1
drg_set = set(range(123766)) # drugs, c0
existing_drgs = df_train.rdd.map(tokenize_DF)
existing_drgs_set = set(existing_drgs.keys().collect())
existing_pros = df_train.rdd.map(lambda r: tokenize_DF(r, reverse=True))
existing_pros_set = set(existing_pros.keys().collect())
missing_drgs = sc.broadcast(drg_set - existing_drgs_set)
missing_pros = sc.broadcast(pro_set - existing_pros_set)

# Create/broadcast dictionaries that have (drug/targ) -> (sum, cnt)
# To be used when computing drug/targ averages
drug_sum_counts = get_sum_counts(existing_drgs).collectAsMap() # dictionary X -> (sum, cnt)
targ_sum_counts = get_sum_counts(existing_pros).collectAsMap() # dictionary X -> (sum, cnt)
drug_sum_counts = sc.broadcast(drug_sum_counts)
targ_sum_counts = sc.broadcast(targ_sum_counts)

# Create Dictionaries that map (drug/targ) -> (most_similar_drug/targ, sim)
# To be used when imputing missing data
targ_sims = sc.textFile(ptnsim_file, minPartitions=128).map(
  lambda r: tokenize_text(r, as_list=True)).filter(
    lambda r: r[1][0][0] not in missing_pros.value and r[0] in missing_pros.value
   ).reduceByKey(reduce_best_sim_knn).collectAsMap()
drug_sims = sc.textFile(drugsim_file, minPartitions=128).map(
  lambda r: tokenize_text(r, as_list=True)).filter(
    lambda r: r[1][0][0] not in missing_drgs.value and r[0] in missing_drgs.value
   ).reduceByKey(reduce_best_sim_knn).collectAsMap()
targ_sims = sc.broadcast(targ_sims)
drug_sims = sc.broadcast(drug_sims)

# Impute the missing values specified in the test matrix.
df_imputed = df_test.rdd.map(lambda r: impute_knn(r[0],r[1])).toDF()
df_imputed = df_imputed.withColumnRenamed("_1", "_c0"). \
      withColumnRenamed("_2", "_c1").withColumnRenamed("_3", "_c2")
# Combine the imputed items with the training matrix in prep. for ALS
df_train_plus_imputed = df_train.unionAll(df_imputed)

# Make predictions for i,j pairs where (i,j) exists in the ALS factorization
als = ALS(maxIter=100, regParam=0.15, userCol="_c0", itemCol="_c1", ratingCol="_c2")
model = als.fit(df_train_plus_imputed)
predictions = model.transform(df_test) # c0 c1 pred

outfile_ob =  open(outfile, "w")
csv_writer = csv.writer(outfile_ob)
for r in predictions.rdd.collect():
    csv_writer.writerow(r)
outfile_ob.close()
```

This work was supported by R01GM108340 from NIGMS and contributions from Google Cloud and the University of Pittsburgh Center for Research Computing.