DRUG EFFECT DISCOVERY USING THE SPONTANEOUS REPORTING SYSTEMS

RAMI S VANGURI
Department of Biomedical Informatics, Columbia University, New York, NY 10032 USA
E-mail: r.vanguri@columbia.edu

JOSEPH D ROMANO
Department of Biomedical Informatics, Columbia University, New York, NY 10032 USA
E-mail: jdr2160@columbia.edu

TAL LORBERBAUM
Department of Biomedical Informatics, Columbia University, New York, NY 10032 USA
E-mail: tal.lorberbaum@columbia.edu

VICTOR NWANKWO
Department of Biomedical Informatics, Columbia University, New York, NY 10032 USA
E-mail: vtn2106@columbia.edu

CHOONHAN YOUN
San Diego Supercomputer Center, University of California, San Diego, La Jolla, CA 92093 USA
E-mail: cyoun@sdsc.edu

NICHOLAS P TATONETTI
Department of Biomedical Informatics, Columbia University, New York, NY 10032 USA
E-mail: nick.tatonetti@columbia.edu

Adverse drug events are a leading cause of morbidity and mortality around the world. Regulatory agencies, such as the Food and Drug Administration (FDA), maintain large collections of adverse event reports, providing an opportunity to retrospectively study drug and drug combination effects. We mined the FDA Adverse Event Reporting System (FAERS) for significant adverse reactions and developed a database of drug effects, known as nSides. FAERS contains millions of reports covering thousands of drugs and thousands of effects, requiring the computing of approximately X billion models. We present a scalable, distributed, on-demand computational infrastructure which can be used with spontaneous reporting systems to calculate side effect significances from drug combinations. Even though nSides is based on FAERS data, the presented infrastructure is portable can be applied to any spontaneous reporting system.

1. Introduction

Spontaneous reporting systems such as the FDA Adverse Event Reporting System (FAERS) are important resources for detecting drug adverse events after a drug is approved (pharmacovigilance). However, pharmacovigilance algorithms often lead to many false positive and false negative findings due to issues of confounding, and detection of drug-drug interactions is an even greater challenge. We previously developed databases for off-label drug effects (OFFSIDES) and drug interactions (TWOSIDES) using FAERS that account for these limitations using a novel Statistical Correction for Uncharacterized Bias (SCRUB).[1] We re-mined FAERS with an updated algorithm to populate a new version of the databases, known as nSides. nSides also contains a public web gateway (http://nsides.io/) accessible to researchers, clinicians and patients alike.

We present the computational infrastructure used to populate the nSides database. Additionally, we present an on-demand interface where users can request drug combination side effect significances. The same infrastructure and interface can be applied to any spontaneous reporting system in order to compare drug effects and bias.

2. Data Sources

There are several data sources which are involved in nSides. We use a curated version of the FDA Adverse Event Reporting System (FAERS) known as Adverse Event Open Learning through Universal Standardization (AEOLUS).[2] AEOLUS aims to clean and normalize the data by removing duplicate cases. This is done by applying standardized vocabularies in the form of RxNorm to map drug names and SNOMED-CT to map outcomes. The AEOLUS dataset is publicly available.

3. Methods

3.1. *Algorithm*

The algorithm used to develop the databases used for nSides is an updated version of the one used to populate the OFFSIDES and TWOSIDES databases. These databases contain side effect significances calculated using raw FAERS data.[1] Generally, a standard signal detection algorithm involves conducting a disproportionality analysis by comparing the observed reporting frequency of a drug and outcome to the expected reporting frequency of all other drugs and the outcome. The metric is known as a Proportional Reporting Ratio (PRR). If the outcome occurred by chance, the frequencies will be equal and the PRR will be one. If the PRR is much larger than one, the null hypothesis is rejected. To reduce sampling variance and selection bias, propensity score matching is implemented to form the groups used in the disproportionality analysis. This procedure, known as SCRUB, matches cases and controls between patients exposed and not exposed to a particular drug (OFFSIDES) or two drugs (TWOSIDES) to mitigate confounding biases. Once cases and controls are matched, the PRR for various side effects are calculated.

There are several key differences between the OFFSIDES and TWOSIDES databases and nSides. The updated algorithm uses a deep learning model instead of logistic regression to

calculate propensity scores to match cases and controls. As a result, the computational power required to populate nSides is much greater. Additionally, nSides is not designed to be limited to effects of single and interactions of 2 drugs.

tensorflow-gpu

3.2. *Computational Challenge*

Populating the nSides database requires the SCRUB procedure to be run for each drug or combination of drugs individually to identify appropriate cases and controls. Once cases and controls are identified, PRR values need to be calculated over a range of side effects. Since the AEOLUS dataset contains ≈4,000 drugs, ≈5,000,000 reports, and ≈8,000 effects to analyze, a computational challenge emerges. To deal with this challenge we employ resources made available by the Open Science Grid (OSG). The OSG provides access to computing resources for research in the United States, free of charge. The computing facilities are located at over 100 sites spanning the United States, primarily at universities and national labs. The computing infrastructure presented is optimized to work on the OSG, but can be adapted to work with other grid computing resources.

The nSides database is initially populated with single drug effects. To do this, a deep neural network model is generated for each unique ≈4,500 drugs in the AEOLUS dataset. The models are generated using the TensorFlow machine learning library. Because the computation involved in deep neural network model generation is more intensive than the logistic regression models used for the creation of Offsides and Twosides, a more complicated computational infrastructure is used.

3.2.1. *Distributed Computing Strategy*

The combinatorial complexity of running all jobs scales on the order of $\Theta(N^2)$ in the number of drugs. Given that a single job running on a server with [SPECS HERE] takes 4-10 hours to complete, a distributing computing strategy is necessary to make the total runtime practical. We utilized two distributed computing systems to determine the PRR between all pairs of coreported drugs, as described below.

The first of these, as mentioned above, was provided by the OSG. The OSG uses the HTCondor job submission software,[3] which handles allocation of computing resources to jobs submitted by the user. The other distributed computing system we utilized was Columbia University's scientific computing cluster, named Habanero. Habanero, like the OSG, uses a job submission management system, but unlike the OSG, Habanero uses the Slurm workload manager[4] for user-submitted jobs. We used both HTCondor and Slurm to submit jobs in a directed acyclic graph (DAG) configuration, supported using native extensions to HTCondor and Slurm.[5] In short, the DAG strategy allows users to take advantage of the fact that many distributed computing workflows consist of jobs containing common elements, such as initial data preparation. In a simplified example, a workflow may consist of one invariant data preparation stage followed by two sequential machine learningg models, where each step relies on the previous step, and the machine learning models accept variable parameters. Here, the DAG capabilities of HTCondor and Slurm will only run the first stage one time, and

sequentially pass the results of each stage to the next stage with the appropriate parameters as the results of the previous stage are available. This approach allows us to substantially reduce the combinatorial complexity of running all jobs to a manageable level.

3.3. *Computational Structure*

Generating a model for an individual drug involves the following:

(1) Dimensional reduction of the complete AEOLUS dataset: only consider co-reported drugs that appear in at least 1 report
(2) Separate exposed and non-exposed reports
(3) Generate deep learning models
(4) Use scores generated by models to perform propensity score matching
(5) Evaluate side effect PRR values
(6) Populate nSides database

3.4. *On-demand Interface*

The original testing of our approach was performed manually on the distributed computing servers using a command line interface and shell scripts. In order to improve the ease of job submission in the future, we have developed a robust searchable shared-usage gateway to the OSG resources we have described previously. The gateway uses a three-tiered architecture consisting of browser-based user interfaces on the frontend, the OSG job submission system on the backend, and middleware to facilitate communication between the other two components. The user interfaces are deployed using the Python Flask framework, and they access a variety of web services that constitute the middle tier of the gateway. These web services are arranged in a way that allows a heterogeneous collection of resources to be accessed remotely in a uniform fashion.

Additionally, the user interface frontend is bundled alongside a RESTful Web API (Application Programming Interface) that allows authenticated users to submit jobs programmatically. This API is implemented using the Agave tenant service[6]—a cloud-based API system designed for developing APIs to be used for scientific computing. The Agave job API manages all aspects of job execution and management, including data staging, job submission, job monitoring, output archiving, event logging, sharing, and notifications.

4. Discussion

References

1. N. P. Tatonetti, P. P. Ye, R. Daneshjou and R. B. Altman, *Science Translational Medicine* **4**, 125ra31 (2012).
2. J. M. Banda, L. Evans, R. S. Vanguri, N. P. Tatonetti, P. B. Ryan and N. H. Shah, *Scientific data* **3** (2016).
3. T. Tannenbaum, D. Wright, K. Miller and M. Livny, Condor – a distributed job scheduler, in *Beowulf Cluster Computing with Linux*, ed. T. Sterling (MIT Press, October 2001)
4. M. Jette and M. Grondona, Slurm: Simple linux utility for resource management, in *ClusterWorld Conference and Expo*, June 2003.

5. P. Couvares, T. Kosar, A. Roy, J. Weber and K. Wenger, Workflow in condor, in *Workflows for e-Science*, eds. I. Taylor, E. Deelman, D. Gannon and M. Shields (Springer Press, January 2007)
6. R. Dooley, M. Vaughn, D. Stanzione, S. Terry and E. Skidmore, Software-as-a-service: The iplant foundation api, in *5th IEEE Workshop on Many-Task Computing on Grids and Supercomputers*, November 2012.