Predictive Toxicology Framework

Introduction

The area of computational predictive toxicology has been increasingly expanding over the last few years due to recent toxicity assay requirements by regulatory agencies [1], especially in the European Union [2]. The toxicity assessment for a chemical is performed for different “endpoints”. When we talk about endpoints we refer to the body part presenting an adverse physiological response (skin irritation, eye irritation, inhalation, systemic toxicity, etc) after exposure to the chemical in question. The large volume of chemicals requiring a toxicity assessment and the possible different chemical endpoints to test for presents a number of combinations that doesn’t scale utilizing traditional assays, particularly animal models. This, in combination with a push towards replacing animal models due to humane reasons [3,4], makes computational predictions increasingly more useful as a pre-screening tool to narrow the scope of experimental validation. Regulatory agencies are in a constant search for different computational models to parse the vast amounts of data available, and come up with classifiers to evaluate which untested chemicals are at a higher risk of posing a threat of eliciting a toxic response for a particular biological endpoint (or, which have a high likelihood of being completely safe at normal exposure levels).

This problem is a good target area for datacenter-scale computing. Thanks to the huge array of big data computational tools like Spark, Hadoop which parallelizes data processing by leveraging the system optimizations thus enabling the user to only worry about the business logic compared to the traditional algorithm based approaches in which the user has to spend considerable time by thinking about the feasibility of the solution. NoSQL stores like Cassandra, MongoDB help with easy data storage and retrieval without having to stress much about the ACID properties, and python frameworks help with building Machine Learning Models without having to write them all from the scratch.

This project consisted of two main components: data aggregation using datacenter-scale computing tools we’ve learned during the course of this class, and a supervised machine

learning framework to attempt prediction of untested chemicals. The toxicity predictions obtained from the machine learning classification suggest that using known biological relations to a chemical has significant promise as a way to infer its toxic potential.

Related Work

The initial and most common approach to attempt in silico chemical toxicity classification has been to use the chemical structure and various physical characteristics, an approach known as quantitative structure-activity relationships (QSAR [5]). This problem has also been approached from various angles employing machine learning, to the point that it outperformed the prediction accuracy from animal models [6]. A common way to aggregate the known information about chemicals, their biological interactions, and their toxicity potential has been to use biomedical domain ontologies [7]. For a good review on the high level subject of this project, see Yang et al [8]. Also, given the huge amount of labeled training set, it would be impossible to not leverage the benefits of Big Data processing frameworks which are little used in the area of Computational Biology. Hence the proposed Toxicology Framework was built to scale enormously for the future as the training data size increases with the passing of each day in this field.

System Design

Dataset description:

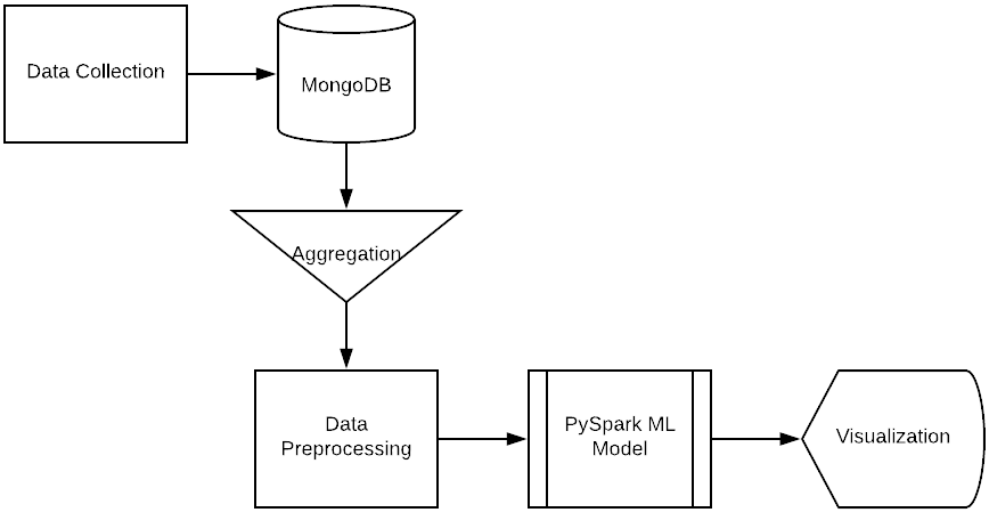
We employed the full dataset of acute toxicity (dermal, oral, and inhalation) from the National Toxicology Program's (NTP) Integrated Chemical Environment (ICE) chemical mixtures database [9] which contains a list of compounds, their active chemical components, and various other metadata. The Global Harmonized System (GHS) toxicity classification score was used as the "label" to be predicted by our supervised learning framework. This is an integer scoring scheme developed by the Organization for Economic and Commercial Development (OECD) that ranges from 1 (seemingly safe) to 5 (highly toxic). Some of these chemicals were tested by different labs, in different concentrations, and resulted in different toxicity scores. In these cases we've decided to use the highest reported toxicity score as our label for that chemical, as a conservative approach. Three different ICE datasets of chemicals were evaluated separately,

for different toxicity endpoints: dermal toxicity (compounds that can cause skin irritation upon contact), inhalation toxicity (compounds that can be breathed in and cause lung irritation) and oral toxicity (compounds that cause systemic adverse outcome when ingested).

For each of these chemical mixture compounds and their individual components, we attempted to merge the data from the Comparative Toxicogenomics Database (CTD) [10], matching chemicals by their CAS number which is a unique identifier for all compounds. Specifically, the chemical-gene and chemical-pathway relations from CTD. Only those chemicals from the three ICE datasets from NTP that had a CAS number, had a toxicity classification, and had a corresponding entry in CTD were taken into consideration. This resulted in slightly less than 100 compounds for each of the three toxicity endpoints. We used the information extracted from CTD as features (which genes the chemical is known to interact with, and which biochemical pathway steps is known to participate in). This was equivalent to a large "right join". Each "gene/interaction" pair observed was used as a binary feature (e.g. `"P53 / upregulation" = 0`, or `"NFKB / indirect downregulation" = 1`), and each biochemical pathway was a feature weighted by the p-value associated to that interaction (e.g. `"xenobiotic metabolization = 1.87e-11"`).

After merging the data (and separated the rest of chemicals in CTD that were not included in the ICE datasets, to use as testing) we trained various classifiers to determine whether the rest of CTD chemicals would pose a high or low risk of toxicity, for each of the three endpoints (dermal, oral, and inhalation). In order to avoid complications with long training times, we limited this task to simple yet appropriate Python implementation (SciKit Learn [11]) of the following machine learning classification algorithms: random forests, naive Bayes, and AdaBoost. The performance evaluation was be based on 5-fold cross-validation for each endpoint dataset separately, since we don't have a "ground truth" for the rest of chemicals. We employed linear discriminant analysis (LDA) before training the classifiers with these features, which resulted in an effective dimensionality reduction.

Initial Architecture:



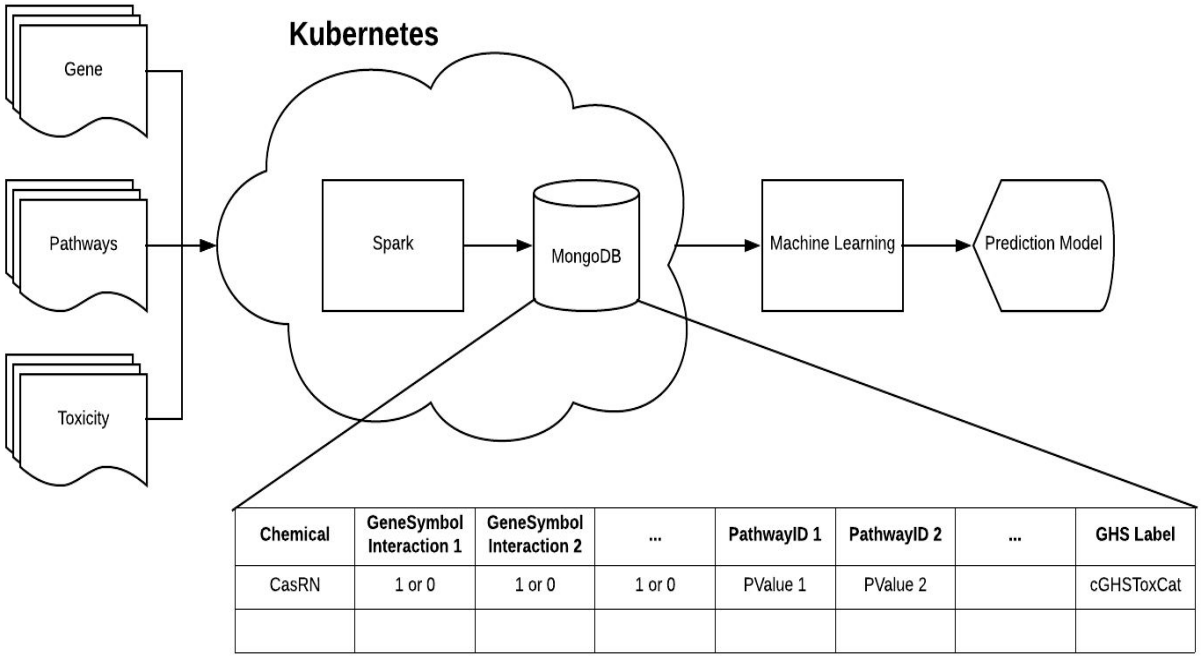
- Data collection: Download the dataset from NTP and CTD and store them to S3
- MongoDB: Use Python code to extract the data and save them to the database
- Aggregation: Use Spark Mongo connector to get the multiple tables from the database. Refer to Dataset portion of the proposal
- Data Preprocessing: Collect only data required for the classifier and clean it
- Machine learning model: Build a machine learning model on Spark. Refer to description below for details.
- Visualization: Visualize the analysis result to explain the useful knowledge found

As we proceeded with the implementation of our proposed architecture, we stumbled upon a few things. Firstly, the results we obtain after the completion of the predictive aspect of the problem were not intuitive enough to visualize. When we cleaned the data, we had an enormously huge test set for which we did not have a truth value. Hence we decided to evaluate the model based on accuracy instead of visualization since the best thing we could come up with was a scatter plot.

Secondly, the architecture was too simple to hold together as a complete entity. We had inconsistencies, like mismatch between the components developed by the individual members of our team. Hence we started focussing more on the architectural aspects of solving the problem. For example, one of us was developing the python/ pymongo application to work with

the data obtained from NTP and CTD and the other was setting up a MongoDB replica set as a server which the python application will talk to. Initially, we wrote a script to automate the mongodb server setup but this involved incompatibility with the local machines we were developing. That's when we decided to containerize the applications since it would be lightweight and was extremely easy to build new images and can also be easily shipped, downloaded and worked with.

Overall Architecture:



Chemical	GeneSymbol Interaction 1	GeneSymbol Interaction 2	...	PathwayID 1	PathwayID 2	...	GHS Label
CasRN	1 or 0	1 or 0	1 or 0	PValue 1	PValue 2		cGHSToxCat

Data cleaning:

We use Pandas in Python to implement the data cleaning part. Pandas is a toolkit in Python, which can manipulate the dataframe easily. In the beginning, we have three table from different database. To integrate all of them, we use 'CasRN' as ID of each chemical element, and we combined columns like 'PathwayID', 'GeneSymbol Interactions' and 'GHS label' as attributes in our processed data. For some datas with missing 'CasRN', we'll just ignore them. Besides, a chemical might have multiple GHS labels, we'll only get the highest one in this case.

Computational Framework:

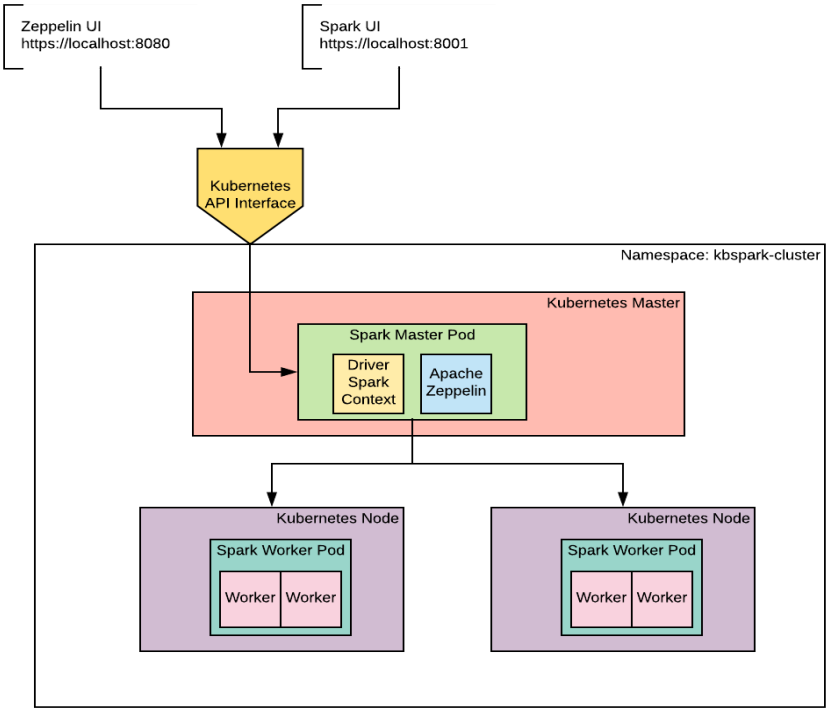
Spark was a great cluster-computing framework for us to process multiple datasets since we could store the structured data frame to the database easily by integrating with MongoDB. PySpark was used to parse the useful columns, such as Chemicals, GeneSymbol Interactions, PathwayID and GHS toxicity categories. “GeneSymbol Interactions” was a combined column with different symbols and interactions. We splitted it to multiple pairs by the delimiter ‘|’. There were over ten thousands of combinations for these gene and interaction pairs. Therefore, the constructed table would be a huge table with over ten thousands of columns. Then we matched each element by ‘CasRN’, an unique chemical, to generate a sparse matrix for machine learning model training. We had three different kinds of toxic dataset, dermal, inhalation and oral to process, which meant there would be three huge tables stored in our MongoDB deployed for analysis.

Containerization:

Docker was an easy choice since it has many readily available images that are prebuilt for different purposes and also has an easy integration with common applications like AWS. We started with containerizing the database with a linux base and built set up the server. Once this was ready, it was built into an image and shipped to work with the python application. Next challenge was to communicate between the two images. We had to create a common network and also attach a volume for the database. Although it was possible to do them separately, it would be tedious to keep track when combining the other applications to come in future. So the solution was to use docker compose where the configurations were pushed into a yaml file and appropriate services were set up. This easy fix helped us run the python application to store the documents into a mongodb collection with one command.

Orchestration:

Although docker compose helped with the initial orchestration, it did not help much when we had to set up a spark cluster since our computation would not work well on a standalone cluster. So we moved to the next step which was Kubernetes which is becoming a standard orchestration tool.



We spun up three EC2 instances on AWS to set up our Kubernetes/ spark cluster. We needed docker images, for both the worker and master nodes of Spark. The spark worker was set up from an ubuntu base image on top of which dependencies like java jdk, python, pip and spark were installed and configured. The necessary python library requirements were then installed with pip. For the spark master, all the steps to set up a spark worker were followed in addition to which Zeppelin was installed. Amazon Elastic IP was used to make the configurations easier.

Spark Master deployment was defined using the Kubernetes deployment abstraction. The image build above was used as the base image and port 7077 was exposed and the container was started. The spark master needs to communicate to the workers for which a service was defined with an additional port to access the Spark UI. The workers were started in a similar fashion with a replication factor of 2. The spark shell was now accessible with support for pyspark. A Zeppelin deployment was configured to access Apache Zeppelin by which jobs can be submitted through Zeppelin Notebooks which was very helpful to avoid reprocessing the data. Since Zeppelin had support for S3, it was easy to access and store the data into S3 buckets after which a simple pymongo script pushed the training data into the MongoDb database.

Evaluation/ Findings

Spark resulted in a highly performant way to parse the datasets used. Architecture refinement at every step proved to be very fruitful. Initially starting with a bare computation framework like spark and then containerizing it was extremely useful for collaboration between the team members. The orchestration step following that was helpful in building automatically scalable deployments.

The 5-fold cross-validation scores were surprisingly positive for the three toxicity endpoints. We used weighted F1-score as the evaluation metric (the harmonic mean between precision and recall, weighted by the number of samples available for each possible classification label) for a fair, balanced representation of the classification performance. The following weighted F1-scores were obtained for the different datasets:

ICE Dataset	**Mean Weighted F1-score**	**Standard Deviation**
Inhalation toxicity	0.923843	0.064043
Oral Toxicity	0.973101	0.026140
Dermal Toxicity	0.949165	0.056096

Review of Team Member Work

Ignacio Tripodi: Studied the various datasets available and selected the most appropriate source for this project, as well as the project concept. Designed the data aggregation/sanitation details and biological feature representation for the machine learning task. Designed and implemented the machine learning framework.

Chi Chen: Wrote the script for data preprocessing, including removing the useless data, grouping the related data into same table. After that, I wrote the scripts for constructing the training table and test table as we discussed in meeting. This script can be extended to the script parse data into mongodb.

Harshini Muthukrishnan: Wrote the MongoDb server setup script for AWS. Created containers for the pymongo and MongoDB parts and orchestrated them with docker compose to store the cleaned data into the database. Set up a spark Kubernetes cluster and wrote dockerfiles to create the spark base image which will be run as pods on the Kubernetes cluster and wrote deployment scripts.

Chu-Sheng Ku: Imported datasets to MongoDB by Python script to test how we can access the data from database in the first stage. Then, implemented the distributed system Spark to process multiple data sources and stored structured data frames to the database MongoDB and created EMR clusters to verify the whole process worked as expected.

Timeline:

- Data collection - 11/1 -> Ignacio
- Data aggregation architecture (stretch goal: incorporate RDkit features) - 11/10 -> Chi
- Import data into Mongodb - 11/15 -> Chu-Sheng / Harshini
- Build up the docker images - 11/19 -> Harshini
- Hyperparameter optimization & reclassification - 11/22 -> Ignacio / Chi
- Machine learning architecture - 11/27 -> Ignacio / Chi
- Dataset processing by Spark - 11/28 -> Chu-Sheng
- Integration with Spark and MongoDB - 12/1 -> Chu-Sheng
- Spark and MongoDB clusters verification - 12/4 -> Chu-Sheng
- Orchestration using docker compose - 12/4 -> Harshini
- Toxicity prediction of the rest of CTD chemicals - 12/10 -> Ignacio
- Spark cluster set up with Kubernetes - 12/10 -> Harshini
- Deployable framework using microservice architecture - Future Work

Conclusion

The data sanitation and merging task yielding less than 100 chemicals for each endpoint illustrated how much work there is left in curation efforts and experimental inquiry of gene expression after chemical exposure. We hope projects like this one will encourage further in vitro assays to validate the predictions obtained and aid the manual data curation efforts like CTD.

The best performing classifier was determined to be random forests, using the “gini” criterion and 1000 iterators. Using all the samples in each dataset we trained a classifier with this configuration, and learned the toxicity score for the rest of chemicals in CTD. Our predictions can be shared with the scientific community for further evaluation, and validated manually using the scientific literature. An interesting continuation of this work could be a natural language processing project that would take the predicted toxicity scores as input and, for example, automated the process of searching for evidence in the literature for those chemicals scoring high (4 or 5 in GHS score).

References

[1] N. Y. Choksi *et al.*, “United States regulatory requirements for skin and eye irritation testing,” *Cutaneous and Ocular Toxicology*, vol. 0, no. ja, pp. 1–58, Nov. 2018.

[2] I. S. Pratt and T. Barron, “Regulatory recognition of indirect genotoxicity mechanisms in the European Union,” *Toxicology Letters*, vol. 140–141, pp. 53–62, Apr. 2003.

[3] P. O. of the E. Union, “Accelerating progress in the replacement, reduction and refinement of animal testing through better knowledge sharing.,” 10-Feb-2017. [Online]. Available: https://publications.europa.eu/en/publication-detail/-/publication/58919142-f1b8-11e6-8a35-01aa75ed71a1/language-en. [Accessed: 22-Jan-2018].

[4] T. Hartung, “Toxicology for the twenty-first century,” *Nature*, vol. 460, pp. 208–212, Jul. 2009.

[5] G. Gini, “QSAR Methods,” in *In Silico Methods for Predicting Drug Toxicity*, E. Benfenati, Ed. New York, NY: Springer New York, 2016, pp. 1–20.

[6] T. Luechtefeld, D. Marsh, C. Rowlands, and T. Hartung, “Machine learning of toxicological big data enables read-across structure activity relationships (RASAR) outperforming animal test reproducibility,” *Toxicol Sci*.

[7] M. E. Pittman, S. W. Edwards, C. Ives, and H. M. Mortensen, “AOP-DB: A database resource for the exploration of Adverse Outcome Pathways through integrated association networks,” *Toxicology and Applied Pharmacology*, vol. 343, pp. 71–83, Mar. 2018.

[8] H. Yang, L. Sun, W. Li, G. Liu, and Y. Tang, “In Silico Prediction of Chemical Toxicity for Drug Design Using Machine Learning Methods and Structural Alerts,” *Front. Chem.*, vol. 6, 2018.

[9] Bell SM, Phillips J, Sedykh A, Tandon A, Sprankle C, Morefield SQ, Shapiro A, Allen D, Shah R, Maull EA, Casey WM, Kleinstreuer NC. 2017. An integrated chemical environment to support 21st century toxicology. Environ Health Perspect 125(5):054501 (DOI 10.1289/EHP1759)

[10] Davis AP, Grondin CJ, Johnson RJ, Sciaky D, McMorran R, Wiegers J, Wiegers TC, Mattingly CJ The Comparative Toxicogenomics Database: update 2019. Nucleic Acids Res. 2018 Sep 24.

[11] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, p. 2825−2830, Oct. 2011.