Summary for Metis Project 5

Predicting Toxicity with Deep Learning

Sean Davern
Seattle, Fall Cohort, weeks 9-12

The project's README.md contains a file-by-file explanation of the project organization.

Overview

The data analyzed in this project came from the NIH's 2014 Tox21 Data Challenge. However, Johannes Keppler University of Linz, Austria compiled it into a Python-friendly collection[1,2] complete with sample code for fitting the data with a Random Forest classifier. The data set provides a training set of 12,060 compounds with 801 physiochemical features with no missing values and a sparse matrix of 272,776 features that essentially contain encodings of two stereochemical "finger prints". The first, ecfp[3], gives the number (n) of each of 194,105 chemical structures present within the compound as identified by an integer string (e.g. ecfp2:-1578247565). The second, dfs[4], gives the presence or absence [1, 0] of another set of 78,671 chemical structures also identified by integer strings (e.g. dfs6:-1788307105). These two fingerprints are one-hot encoded to generate the 272,776 feature columns.[5] The modeling targets for the compounds are a collection of 12 chemical analysis measures that indicate [0, 1] whether the compound tests as toxic against a specific analytical method for a specific human biological pathway or set of pathways. The target data has many missing values. Finally, the provided data also includes 647 compounds and their toxicological target toxicities that comprise a test set. The project goal is to predict the 12 toxicity metric values as accurately as possible for the test set.

---

[1] [Mayr2016] Mayr, A., Klambauer, G., Unterthiner, T., & Hochreiter, S. (2016). DeepTox: Toxicity Prediction using Deep Learning. *Frontiers in Environmental Science*, **3**:80. doi/10.3389/fenvs.2015.00080

[2] [Huang2016] Huang, R., Xia, M., Nguyen, D. T., Zhao, T., Sakamuru, S., Zhao, J., Shahane, S., Rossoshek, A., & Simeonov, A. (2016). Tox21Challenge to build predictive models of nuclear receptor and stress response pathways as mediated by exposure to environmental chemicals and drugs. *Frontiers in Environmental Science*, **3**:85.

[3] David Rogers*†Mathew Hahn‡, Extended-Connectivity Fingerprints, J. Chem. Inf. Model. 2010, 50, 5, 742-754. doi/10.1021/ci100050t

[4] Ralaivola L, Swamidass SJ, Saigo H, Baldi P: Graph kernels for chemical informatics. Neural Networks. 2005, 18 (8): 1093-1110. doi/10.1016/j.neunet.2005.07.009

[5] Sean Davern, input_parameter_exploration.ipynb documents some of my EDA.

Modeling

MVP DNN

Using general structural information for the neural network space explored by Mayr, et.al.,[6] I created my MVP for the first toxicity target: NR.AhR.[7,8] The MVP successfully trained a dense neural network from scratch giving an auc_roc of 0.86 in comparison to the 0.91performance of the random forest models[9]. AUC_ROC was used as the canonical performance metric for the 2014 NIH contest so I began using this metric.

DNN's for All Targets

Following up on the MVP, the network structure used for the MVP was trained on all the toxicity targets.[10] Models of two of the 12 targets out performed their random forest counterparts.[11] Generally, all the models have auc_roc's within 10% of the performance of the random forest models.

Bayesian Optimization/Hyperparameter Tuning

After quite a lot of consideration of metrics, discussed below, I implemented hyperparameter tuning using the HyperOpt library and 1-F_1 scores as the target metric.[12] The NR.AhR metric was tuned over a few optimization "spaces". Interestingly, noticing that Mayr, et.al. suggested dropouts going to 0.5 it seemed like lower dropouts might be feasible if the number of neurons was reduced. I switch the optimization space from 1024/layer to 16,384/layer to 64 to 1024/layer. Interestingly, the final model it arrived at had a single 64 neuron dense/dropout layer and performed better than the baseline architecture that had 3 x 1024 neuron dense/dropout layers. This sped up training and reduced the required computer architecture needed to train the models.

I also learned that the 512 sample mini-batch size suggested by Mayr, et.al. might be higher than optimal.[13] You'll see my later optimization space looked at batch sizes in the 16-256 sample range.[14]

The final NR.AhR DNN model had an F1 score of 0.59 compared to the decision threshold-optimized random forest model value of 0.51. So, that's a third target with DNN's better than the random forest. I believe the others would have been able to be trained to be better but I ran out of time.

---

[6] ibid 1. see section 2.2.4 and Table 2. Note: I don't think the authors ever divulge the exact structure of their optimal models. I picked intermediate values of the ranges they gave.

[7] Sean Davern, mvp.ipynb

[8] Sean Davern, Imbalanced+metric.ipynb Note: worked out balancing and started metrics consideration.

[9] Sean Davern, random_forests.ipynb

[10] Sean Davern, single_task_dnn_base.ipynb

[11] Sean Davern, model_performance_tables.ipynb Note that section 2 of the notebook displays a table for each target. Model (index) 2 of each of the tables shows results for the initial DNN architecture.

[12] Sean Davern, bayesian_optimization_singe_dnn.ipynb

[13] Frank Dernoncourt, Stack Exchange answer to Tradeoff batch size vs. number of iterations to train a neural network

[14] ibid 11. See section 4.

Another significant lesson I encountered was that due to some bug, Keras sequential models with an InputLayer as its first layer can be saved but can't be reloaded! I documented this pretty thoroughly for later blogging.[15]

Remote (AWS) Processing

The initial Bayesian optimization was worked out on my mac. It took it 6 hours to complete ~55 iterations. I tried to use the Nvidia GPU on my mac[16] but realized that this is not supported by MacOS 10.15 though it was on 10.13.[17,18] At this point I created a new AWS EC2 instance with GPUs and completed the Bayesian optimization work above on this resource, MUCH more quickly! This spurred me to learn and implement how to save, restart and continue optimizations that might stop or be stopped mid-run.

Metrics Evaluation

I spent quite a lot of time considering, reading about, and exploring discussion about metrics for imbalanced data sets. While there seems to be a lot of online wisdom suggesting the use of precision-recall curves over ROC curves for imbalanced data sets,[19] Luque et.al. suggest the Matthews Correlation Coefficient is the best option for error consideration and that F1 is highly biased.[20] However, without having time to consider this with instructors, I stuck with F1 since I had a lot of work completed by the time I read this paper.

Python Modules

Continuing to develop my Python module skills, I created help_functions.py which was used various places to load data, save models trained, evaluate and track model performance results.

Flask App

Originally intending to create a Flask app to "deploy" my models, I began this development.[21] I had the flask app to the point that a user could pick a file from their computer and "upload" it (compound input parameters file: a pickled DataFrame). The filename was passed to a second route where it was displayed intending that this file would be used by the predictor.py whose values would be displayed on the page. I started the development of the predictor[22] function. However, after discussions with Cliff I decided to pivot away from this initiative since an app isn't really a realistic way this project would be "deployed", probably. This work was tabled.

---

[15] Sean Davern, keras_save-load_issue.ipynb

[16] Sean Davern, GPU_attempt.ipynb

[17] Uxio Pineiro, Nvidia eGPU + MacOS + TensorFlow-GPU? The Definitive Setup Guide to Avoid Headaches

[18] Gllagher, William and Mike Wuerhele, Apple's management doesn't want Nvidia support in macOS, and that's a bad sign for the Mac Pro

[19] Why Can't I just Use the ROC Curve?

[20]Amalia Luque, Alejandro Carrasco, Alejandro Martín and Ana de las Heras, The impact of class imbalance in classification performance metrics based on the binary confusion matrix, July 2019, doi/10.1016/j.patcog.2019.023

[21] Sean Davern, flask_app folder and individual_compound_generation.ipynb

[22] Sean Davern, predictor.ipynb