Overview:

My overall goal throughout these projects was to build and test models to predict drug toxicity. Drug toxicity is one of the most common causes for failure in drug discovery pipelines, which is especially costly if it's only found out in the later stages. Currently the most common methods to assess toxicity involve animal testing, which is expensive, time-consuming, and ethically questionable (B. Indira Priyadarshini, et al.). Machine learning would be a much better alternative. However, the current models aren't very accurate, so I worked towards making models with improved performance so that they can be added to drug discovery workflows. There are many challenges which make drug toxicity a difficult task to predict. There is a broad range of ways a compound can be toxic, and the degree of toxicity depends heavily on where it is in the body and what organs or proteins it's interacting with. It is also challenging to gather raw experimental data on toxicity since they require in-vitro/vivo experiments, and then it's hard to get access to that data. And even with that data it's difficult to translate non-human models to humans due to biological differences.

Simple SMILES Transformer (SST):

(Note: all file paths in this section are relative from lus/eagle/projects/datascience/lvairus/Repo_copy/SST.)
I started by using models with SST, which was run with the command

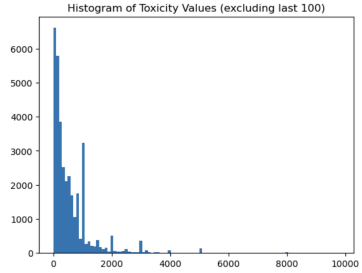
```
Python
python sst_reg_train.py -t [training datafile] -v [validation datafile]
-s [SMILES column name] -l [label column name] -c [config file] -e [num
epochs] -b [batch size] -L [learning rate, default 1e-5]
```

I tried doing regression and classification tasks on mouse intraperitoneal LD50 toxicity, cardiotoxicity, and respiratory toxicity data. However, I wasn't able to achieve good results from these. I never achieved a positive R2 metric or multiclass-AUROC over 0.70. For more information, the links above lead to each dataset's respective wandb project. However there isn't much specific information for reproducing since this was before I figured out how to properly log configurations and keep track of them between runs.

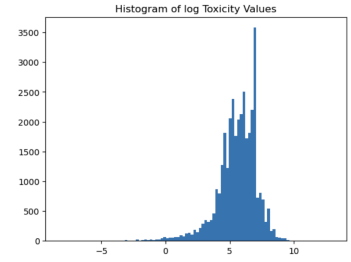
Mouse:

I used the acute toxicity mouse intraperitoneal LD50 dataset from TOXRIC. This contained SMILES of 35,000 compounds and their LD50 value (the concentration of the compound which killed half the population of mice when administered through the

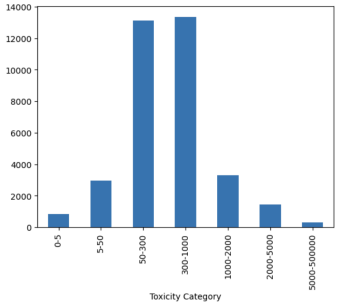
intraperitoneal route). These values ranged from 0-500,000 (mg/kg). The data was skewed towards the lower end so I also calculated the log of them to use as an alternative label for the model. I also defined toxicity ranges loosely based on the [GHS Toxicity Classification] guidelines and split the compounds into those classes, to use as labels for a classification task instead of a regression one. These class ranges were LD50 0-5, 5-50, 50-300, 300-1000, 1000-2000, 2000-5000, and 5000-500,000.



Histogram of the toxicity values in the mouse dataset (excluding the last 100 as they were too large which made the graph hard to read.)



Histogram of the log of the mouse toxicity values.



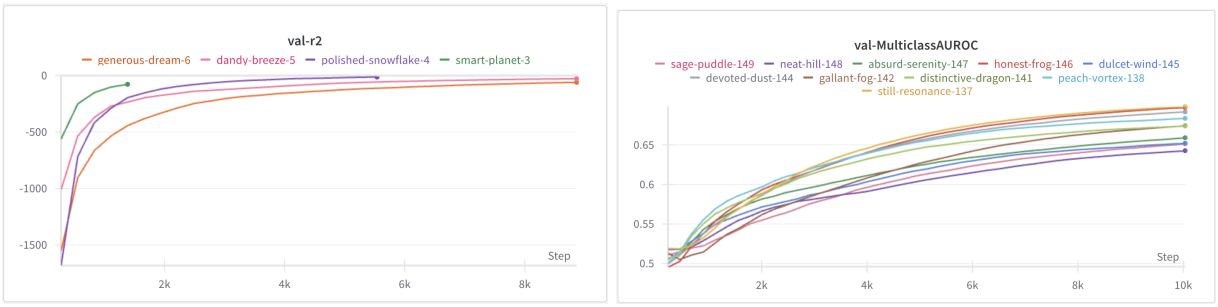
Bar graph of the distribution across toxicity categories

I also ran a t-SNE dimensionality reduction on the morgan fingerprints of the compounds and colored them by toxicity value to see if there was any clustering. There were a few small clusters of compounds with similar toxicity values but most of them shared the same space regardless of toxicity. Therefore I wasn't sure if the model would be able to learn about differences between molecules and relate them to toxicity.

The SST run script takes a data file for training and a data file for validation, so I presplit my data into files which can be found at …Mouse/data/mouse_train.csv and …Mouse/data/mouse_test.csv.

Results: I tested multiple hyperparameter configurations but didn't get an r2 score greater than or equal to zero so I tried again with the labels as the log values of the original toxicity values. Again I wasn't able to reach a positive r2. Then I tried a classification model using multiclassAUROC as my metric. I modified the existing run script for a regression model and saved it in …Mouse/sst_class_train_lv.py. However I didn't get a value greater than 0.70.

Discussion: The poor results were somewhat expected since the t-SNE above didn't show much clustering between compounds. The only thing we know about these compounds is that they were administered through the intraperitoneal route and that the recorded concentration killed 50% of the population. We have no information on how the mice died, such as which organ systems were affected. Therefore there's probably overlap between a compound that's toxic to one organ and another similarly-structured compound that's nontoxic to another organ. It's probably best to train models on data that's organ specific, as there might be a more prominent pattern that the machines will be able to find and learn. If you want to have a model that predicts general toxicity, it might be best to use an ensemble of machines that are each trained on a specific system.

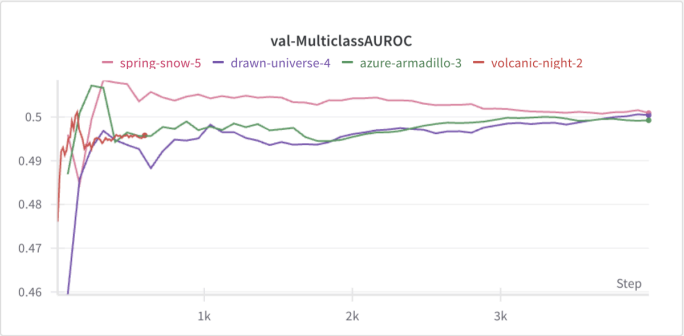


Mouse curves for val R2 of regression model and val multiclassAUROC for multiclass model.

Cardiotoxicity:

The cardiotoxicity data provided by TOXRIC consisted of four files. They all had SMILES representations for the same 1547 compounds. The toxicity value column was a binary 0 or 1. Toxicity was measured by IC50 of hERG (what concentration inhibited the hERG gene by half). Each file had a different cutoff which determined whether a compound was toxic or not. The cutoffs were IC50 1, 5, 10, and 30 (mg/kg). I combined these datasets into a single one and sorted the compounds into the classes IC50 0-1, 1-5, 5-10, 10-30, and >30 (nontoxic). I presplit this data and saved the files in …Cardio/data/cardio_train.csv and …Cardio/data/cardio_val.csv.

Results and Discussion: I thought this dataset would do better than the mouse dataset since it's specific to the hERG gene, but I didn't get a multiclassAUROC value greater than 0.50. Average AUC isn't much better than random guessing. I didn't look at class-wise AUC and looking back it's clear that we're missing important information without that. Maybe one class actually did pretty well but a different class did very poorly which brought the average down. Possible next steps could include logging AUC for each class and seeing if one does better than others. If one class does well, we could optimize a model for that class, then look into other options for the rest of the classes. Another problem could just be the lack of data points, as there are only 1547. In that case one could try finding more cardiotoxicity data online to see if performance improves at all.

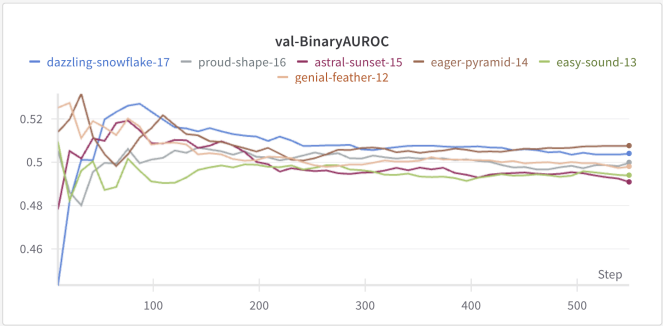


MulticlassAUROC for cardiotoxicity using an SST model

Respiratory:

The TOXRIC respiratory toxicity data was a single file with 1348 compounds, consisting of SMILES strings and a binary 0 or 1 to indicate toxicity. I again presplit the data and saved it in …Resp/data/resp_train.csv and …Resp/data/resp_val.csv. I modified my multiclass code to perform binary classification with a BinaryAUROC metric and saved the code in …Resp/sst_binclass_train_resp.py.

Results and Discussion: Again, I didn't get an AUC greater than 0.50. Maybe there are multiple ways a compound can be toxic to the respiratory system so the machine isn't able to learn all of them, or maybe 1300 data points isn't enough to properly train it. Another possibility could be that the model is too small to learn about this task, in which case you could try training with a different model to see if it performs better.



BinaryAUROC for respiratory toxicity using an SST model

Since there was no progress with this model I moved onto using MolFormer instead of SST to calculate SMILES encodings.

ConPlex

ConPlex is a model available online for public use that takes a compound SMILES and a protein sequence and predicts the binding probability between them. The github can be reached

here. All the code I used can be found in /lus/eagle/projects/datascience/lvairus/ConPlexdir, and all the following file paths in this section will be relative to this path.

Usage: Pip installing ConPlex didn't work so I compiled it from source following the code in the readme in their Github. The usage follows the command line prompt

```
Python
conplex-dti predict --data-file [pair predict file].tsv --model-path
./models/ConPLex_v1_BindingDB.pt --outfile ./results.tsv
```

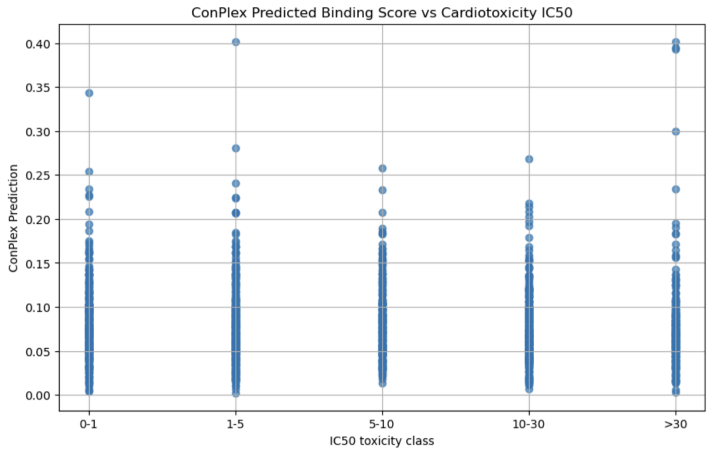
It takes a tsv of all the compounds and protein pairs you want to compare, the path to the model, which is provided by the github, and a path to save the output file in. the input tsv file follows the format [protein ID]\t[molecule ID]\t[protein Sequence]\t[molecule SMILES].

Cardiotoxicity:

I first tried using this with the cardio IC50 data mentioned before. Since the compound toxicities were based on inhibition of hERG, we thought there might be a correlation between toxicity class and KCNH2, the protein that hERG codes for. Our hypothesis was the lower IC50 a compound had, the higher binding score it would have.

I obtained the protein sequence for KCNH2 from uniprot. To download it, go to the "Sequence and Isoforms" section (found in the right menu), and click the download button at the start of the gray box with the sequence of letters. I saved that download in a text file here …/KCNH2_human.txt. My input file for the cardiotoxicity compounds can be found here …/ConPLex/conplex_input.csv, And the output file is here …/ConPLex/results_IC50.tsv. I was having trouble getting my input file to work as a tsv so I modified the code in …/ConPLex/conplex_dti/cli/predict.py to take a csv file instead. The code also wasn't reading my SMILES and sequences as strings so I added lines which converted them to str.

Results and Discussion: All of the scores I received were very low and I didn't find a correlation between IC50 and binding score. This could either be because of an error in my code or because there really isn't a correlation between them. Maybe it inhibits hERG not by binding with KCNH2 but by interfering with some other process that produces KCNH2 or that works together with KCNH2. It might be best to read a paper or talk to someone who knows more about hERG inhibition and what it means biologically in order to find out where you're supposed to look to see causation effects and how you can incorporate that into a machine learning model.

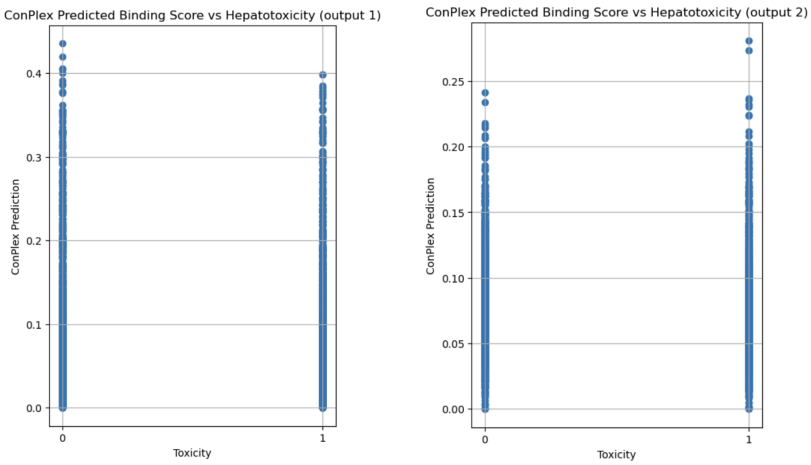


ConPlex binding score predictions vs Cardiotoxicity IC50 class

Hepatotoxicity:

I tried again using the hepatotoxicity data from TOXRIC, which is a single file of 2889 compounds, with SMILES and a binary 0 or 1 for toxicity. In a paper by Galetin, A. et al., 2024, I read that there was "a progressive decrease in OATP1B activity occurs in patients with increasing hepatic impairment." I wanted to check if there was a correlation between hepatotoxicity and binding score with OATP1B proteins. I searched up OAT proteins on wiki and decided to focus on the protein SLCO1B1. I found its sequence on uniprot using the same method as above and created a new input file, found at …/ConPLex/conplex_hepa.csv. I don't remember what I did but I changed something in the predict.py code to try to debug it so there are two output files: …/ConPLex/conplex_hepa_out1.tsv and …/ConPLex/conplex_hepa_out2.tsv.

Results and Discussion: Neither output files show a significant correlation between toxicity and binding score. It might be worth it to recompile the ConPlex code from github to see if there's a bug in my code because it seems like a pretty big warning that it works two ways and doesn't crash at all. Other than programming error, it could be that, similar to the cardiotoxicity attempt above, hepatotoxic compounds do not bind to SLCO1B1 and instead affect some other process/system which in turn causes the decrease in OATP1B activity that was observed in the paper above. Again, it could be helpful to consult an expert to figure out what hepatotoxic compounds bind to.



ConPlex binding score predictions vs Hepatotoxicity, both outputs

Since I wasn't getting any meaningful results, I returned to focusing on creating models for predicting toxicity.

Single-task vs. Multi-task:

(Note: all of the code in this section is from /lus/eagle/projects/datascience/lvairus/Repo_copy/MolFormer/Multitask_Class and all file paths will be relative from here.)

I started training models using MolFormer. MolFormer is a pretrained model available online for public use which was trained on molecular structure with a billion compounds, obtained from PubChem and Zinc datasets. I began by just using mouse or cardiotoxicity data, but the metrics didn't improve that much. I took a step back and read papers to gain insight into what kind of model might do better. We eventually decided to test the performance of a multitask model, as opposed to a single task one.

Data: I used 13 datasets of oral toxicity among different species (summary statistics in the "Acute Toxicity, Oral" table in the TOXRIC stats google sheet). All animal data was LD50 and human data was TDLo (the lowest record dose to cause a toxic effect). All the data can be found in …/oral_data. The total model was made up of MolFormer, a defined NNModel with linear layers, and then 13 different output layers, one for each dataset. The nnmodel structure is defined in …/classification_layer_multi.py.

To get single-task performance, I ran each dataset separately with the same model and run script, using only one of the output layers, to have a benchmark to determine whether the AUC improved or not. For multi-task performance, I read in all the datasets and used a task ID to specify what output layer they should go to. To run this yourself, use the script at …/run_script_lv_multi.py, and either set the hyperparameters you want in the default config file

at …/multi.yaml or make your own yaml file and pass it as an argument.

```
Python
python run_script_lv_multi.py -y my_config.yaml
```

The run script takes a directory of data files instead of a single data file for the “dataset” parameter so without code modification your data will have to be isolated in its own directory.

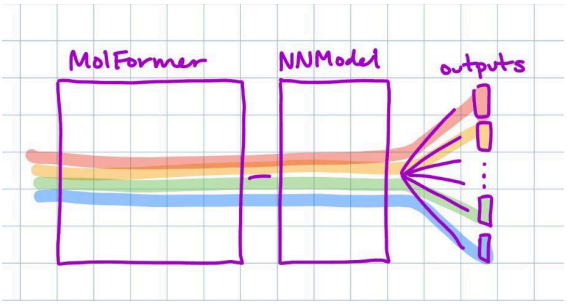
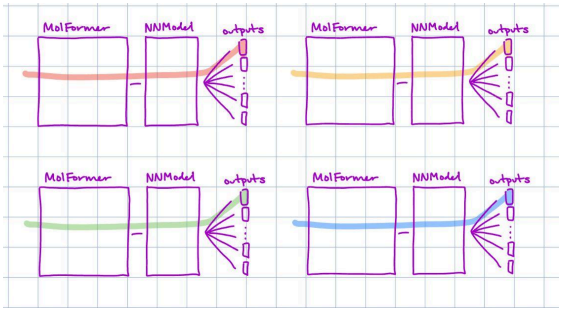


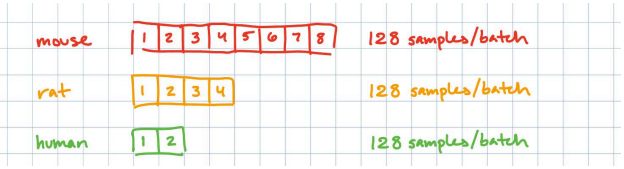
Diagram of multi-task model



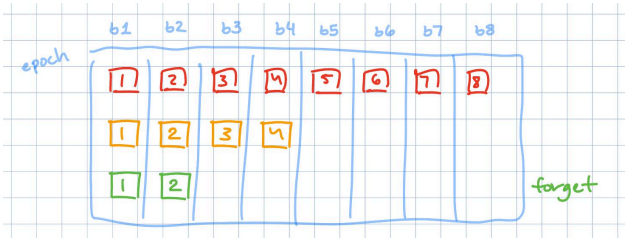
Diagrams of single-task models

Training Curriculum: There was a wide range of dataset sizes between all the animals, from 100 to 20,000 data points. A single dataset would be split into mini-batches, and a macro-batch would be some collection of mini-batches for an iteration in the train/val loop. I considered 3 options for how to train all my data. For explanation purposes assume my data is from the following diagrams.

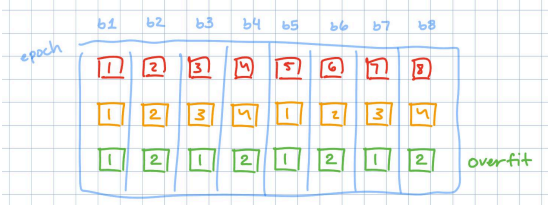
I have the most mouse data and the least human data. One option could be to split each dataset into mini-batches of the same size.



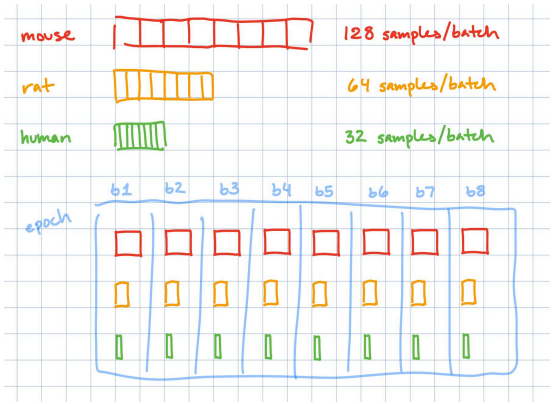
In the diagram beside, each square is a mini-batch and each column is a macro-batch. I could line up the mini-batches for however far they reached. However, at one point the model would only be training on mouse data and I thought it might forget everything about the smaller datasets.



Another option was to repeat the mini-batches of the smaller datasets so the model would always see data from each species. However, I was worried it might overfit on those smaller datasets.

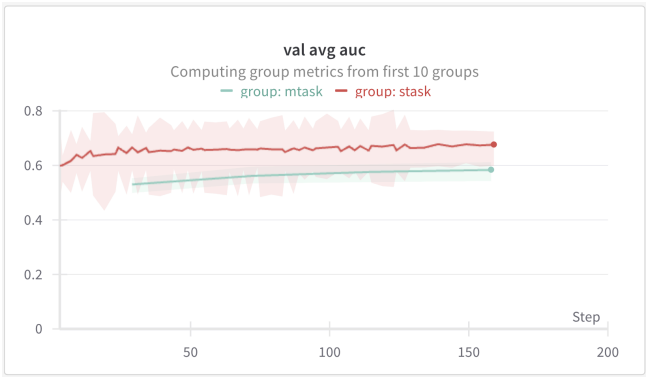


Another way to split the data was to split them into the same number of batches. This way, you can see part of every dataset for every macro-batch without repeating any data. This is the option I decided to use.



To do this in a controlled manner I made a custom batch sampler called "RoundRobinBatchSampler" (code in …/data_utils.py) which takes a total number of mini-batches and then splits the data points across all mini-batches evenly. This way I got an equal number of mini-batches for every dataset, and just combined one mini-batch from every set to make a training batch.

Results: on average, the multi-task model performed worse than the single-task one. Most prominently, the human data performed worse in the multi-task model. All the summary AUC results and comparisons can be found here stask vs mtask. You can find the wandb runs here. To view only the single-task or multi-task runs, filter by "Group", and set it to "stask" or "mtask", respectively. There are some other groups but you can ignore them (more information about wandb use in the wandb section below). In the mtask runs, I log the train loss, validation loss, and validation AUC for each animal. There are also bar charts for total best scores. All the "best_own_ep*" metrics are taken for a task when it reaches its own highest AUC, and all the "best_avg_ep*" metrics are taken for every task when their average AUC is the highest.



Average AUC of single-task models vs multi-task models

Discussion: We expected the multi-task model to perform better since there was much more data that it could learn from. Especially the human data, since there was so little (120 data points). We thought that although the added 30,000 data points were from other species, it would still teach the model something about toxicity that it would be able to translate to human toxicity. However, this was not the case. This could be because there were too many different species. Although they were all animals, they all had different responses to toxic compounds which could have confused the model and made it difficult to learn and translate from all of them. I think it would be better if we had less tasks. Most of the datasets were pretty small (under 1000 data

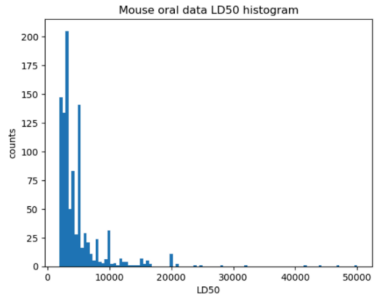
points) so I think they just introduced a degree of confusion to the model that was not balanced by the information the machine learned from them. Therefore I think it would be best to start small and just use mouse data (20,000 data points) and possibly add rat data (10,000 data points). There are also other tasks we could try to add to the machine other than toxicity such as binding score, solubility, etc. Or maybe instead of having the machine predict on it we could just add those values to the input of the model.

Mouse Models

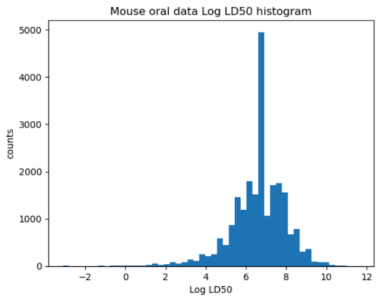
(Note: all the file paths in this section are relative to /lus/eagle/projects/datascience/lvairus/Repo_copy/MolFormer/Mouse)

Before making a model to predict toxicity for humans, I wanted to figure out what architecture would perform the best, so I started to test out a variety of different models with the acute toxicity mouse oral LD50 data. The wandb logging for all of these runs can be found here. You can filter by group name (in parenthesis at the header of each section) in the Workspace section or go to the Sweeps section where you'll see each model run on 10 random seeds. The graph with bar plots of all f1 scores for each model can be found at the end of this section under "Overall'.

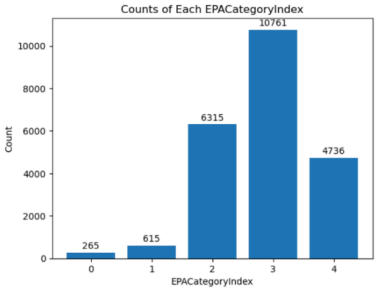
Data: The dataset has 22,000 data points, with SMILES and an LD50 toxicity value for every compound. The toxicity ranged from 0-50,000 (mg/kg). I used these original values, the log of them to get a more normal distribution, and split them into classes based on the EPA oral toxicity classes for terrestrial organisms. Below are the distribution plots.



Histogram of original values



Histogram of log values



Bar chart of toxicity classes

Data Preprocessing: To make models more comparable, I presplit the mouse into training and validation sets so that all models would be trained and validated on the same data. (data at …/data/mouse_train_df.csv and …/data/mouse_test_df.csv)

Metrics: since the data was uneven, I changed from using AUC to f1 scores to evaluate performance. F1 score takes the harmonic mean between precision (how many predicted positives are true) and recall (how many actual positives are predicted positive). For example, if our data has more negative samples, and just predicts everything to be negative, AUC will be high while f1 score will be low.

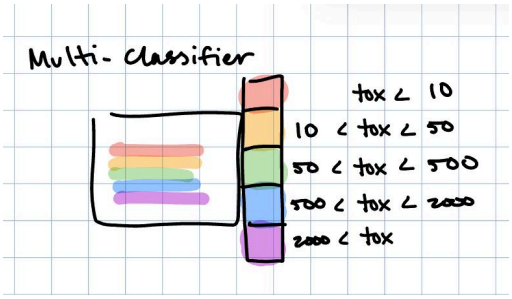
Weighted loss function: for all classification or binary models, I ran them with an unweighted loss function as well as a weighted one. All weights were calculated such that each class is weighted inversely to its proportion of the total.

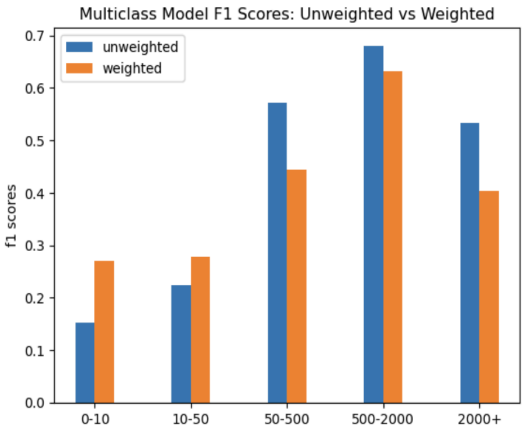
$$weight_{class\ i} = \frac{total\ number\ of\ samples}{samples\ in\ class\ i}$$

$$normalized\ weight_{class\ i} = \frac{weight_{class\ i}}{sum\ of\ all\ weights}$$

You can specify in the config file whether you want the model to be "weighted" or "unweighted", and it will follow an if block using that value to define the loss function.

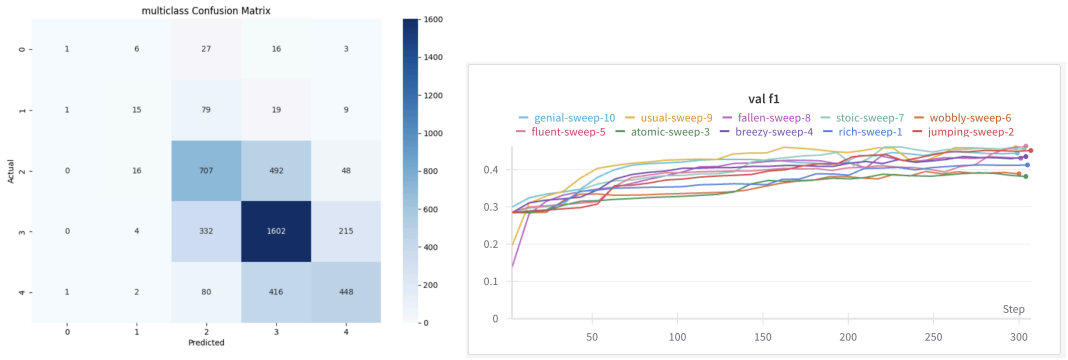
Multiclass (multiclass):



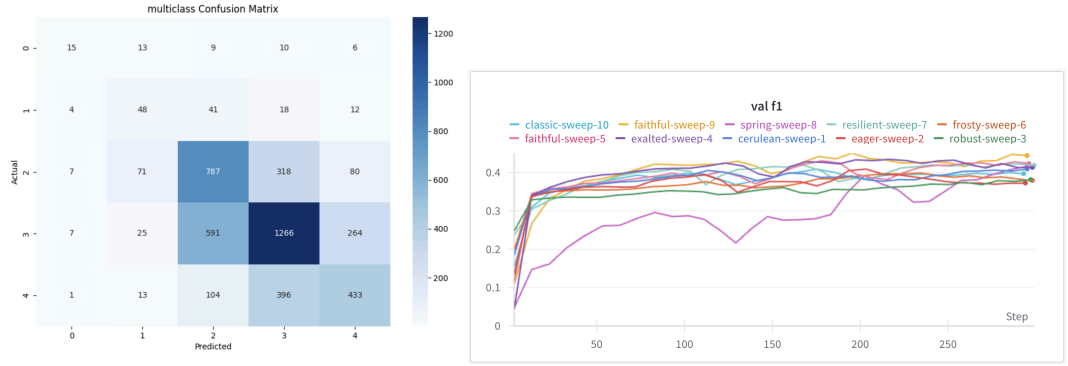


The first thing I tried was a simple multiclass model, with classes 0-10, 10-50, 50-500, 500-2000, and >2000. However, it didn't perform well, as it barely predicted anything to be in the first class. When I tried using a weighted loss function, it only improved performance for the first two classes and worsened the rest. (runscript at …/multiclass_runscript.py)

Discussion: The data was skewed, there were very few samples in the first two classes so it makes sense that those f1 scores were low. I thought it was interesting how the weighting function improved some classes at the cost of performance in others, as if it's distributing performance ability across them. You could try other weights, such as random or learnable ones. Another idea I had was to split your data, not by toxicity level but just by whatever ranges give you the most even split across all categories. This way, although you might not get the classes you're most interested in, you might get better performance since classification models do the best when the classes are evenly distributed.

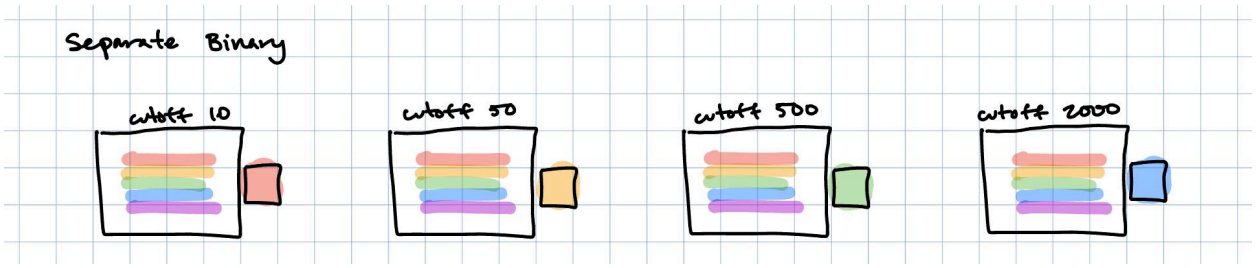


Unweighted multiclass confusion matrix and f1 curve (averaged across classes)



Weighted multiclass confusion matrix and f1 curve (averaged across classes)

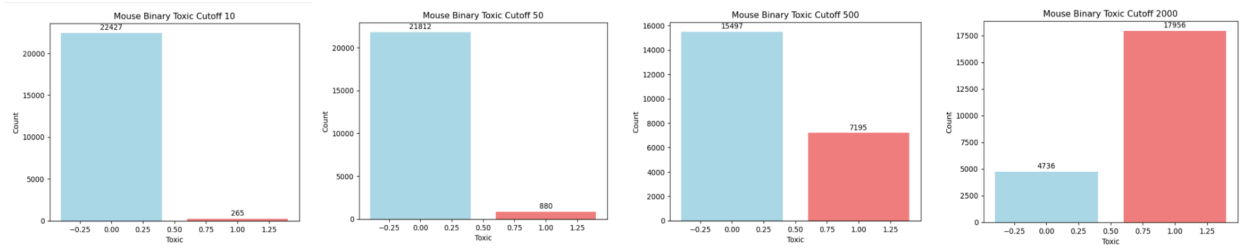
Separate Binary (binary):



Next, I tried making 4 separate binary models. Each would train on all data and would predict whether toxicity was above or below a cutoff. I had a model to predict between each 10, 50, 500, and 2000. The distributions between toxic and nontoxic for each cutoff are shown below. (runscript at …/binary_runscript.py)

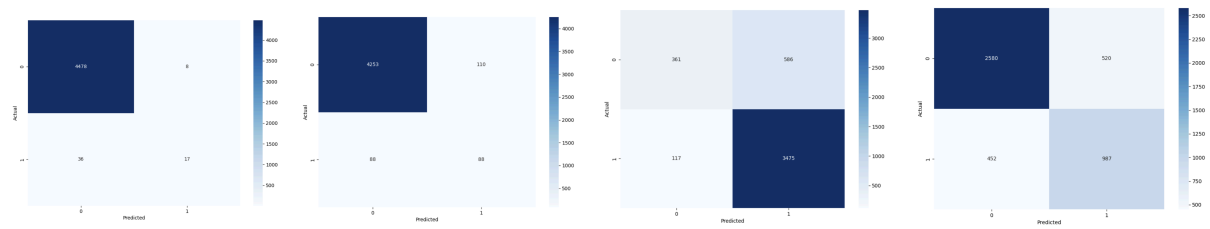
Results and Discussion: These performed better, which I think is because they had more nodes to work with and focus on one specific task. (data statistics in the TOXRIC stats sheet, under "Mouse Oral LD50 Data", and "Binary Models"). When I trained with the weighted loss function, it performed

worse for every class. This was surprising, especially for the first two classes, since those were extremely unbalanced. Looking at the confusion matrices for the cutoff 10 models, it looks like weighting only helped the model get a few more true positives, but a lot more false positives. So it was predicting more compounds to be positive, just not the right ones. Maybe instead of weighting by number of samples in each class, we should instead look at the molecular diversity of the compounds and weigh them according to the distribution of scaffolds or area in the molecular embedding space.



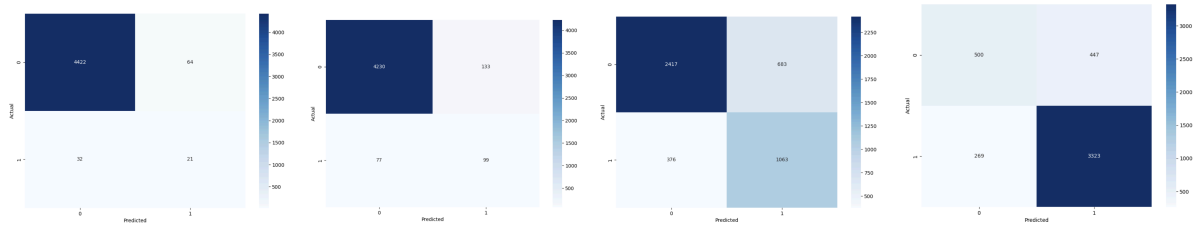
Cutoff: 10 *Cutoff: 50* *Cutoff: 500* *Cutoff: 2000*

Bar charts for distribution of nontoxic vs toxic depending on each cutoff



Cutoff: 10 *Cutoff: 50* *Cutoff: 500* *Cutoff: 2000*

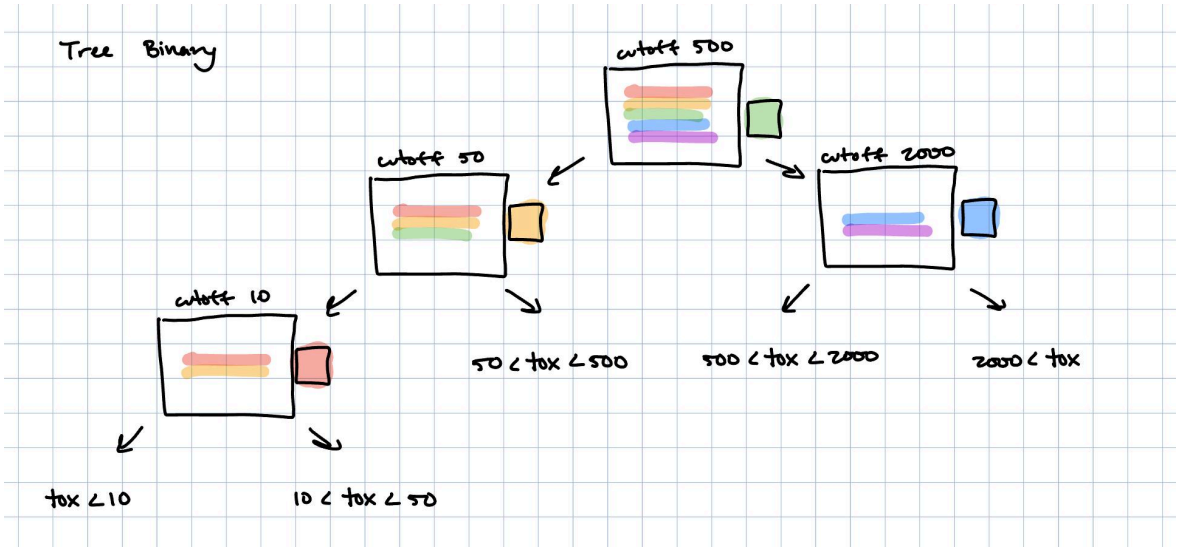
Unweighted confusion matrices



Cutoff: 10 *Cutoff: 50* *Cutoff: 500* *Cutoff: 2000*

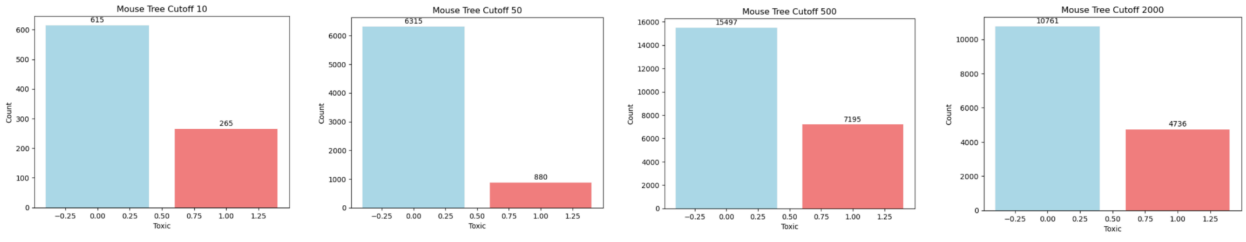
Weighted confusion matrices

Tree Binary (tree):



Since some of the separate binary models were very skewed, we wanted to see if organizing those binary models into a tree structure would do better. The idea behind this model was to give each model a smaller range of data to train on such that the nontoxic vs toxic splits would be more even.

Model structure: the data was most evenly split at the cutoff 500, so we chose that to be the root of our tree. First all the data would go through that model. Then whatever compounds were predicted to have a toxicity value less than 500 would be inputted to the cutoff 50 model. If it was less than 50 it would then go to the cutoff 10 model. Similarly whatever was predicted to be above 500 would be inputted to the cutoff 2000 model. These models were only trained with the data that they should receive. For example, the cutoff 50 model was only trained on data that had a toxicity value less than 500. (More information in the "Mouse Oral LD50 Data", "Tree Models" table in the TOXRIC stats sheet).

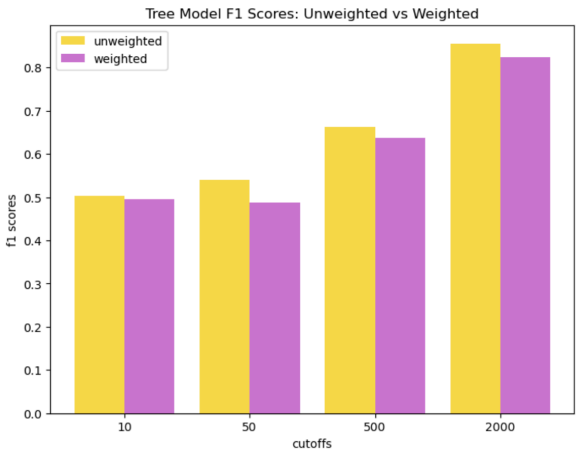


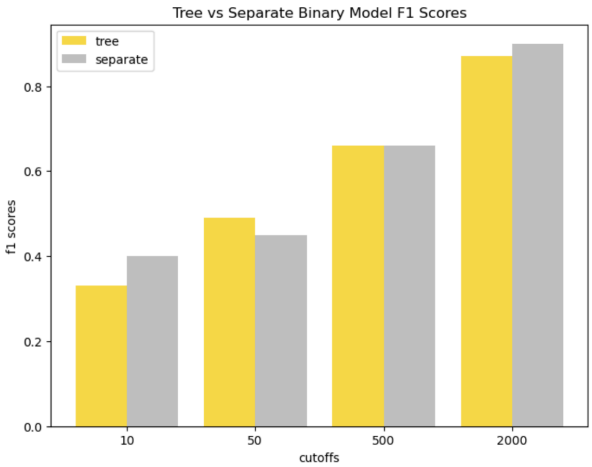
Cutoff: 10 *Cutoff: 50* *Cutoff: 500* *Cutoff: 2000*

Bar charts for distribution of nontoxic vs toxic depending on each tree model

To train and validate the models, I used all the data that was in the correct toxicity value range. However, for inference, I ran all the data through the root model and then whatever was inputted to the child models was determined by the root's predictions, so there was some uncertainty of whether the data points going through the child models were actually in the correct toxicity range.

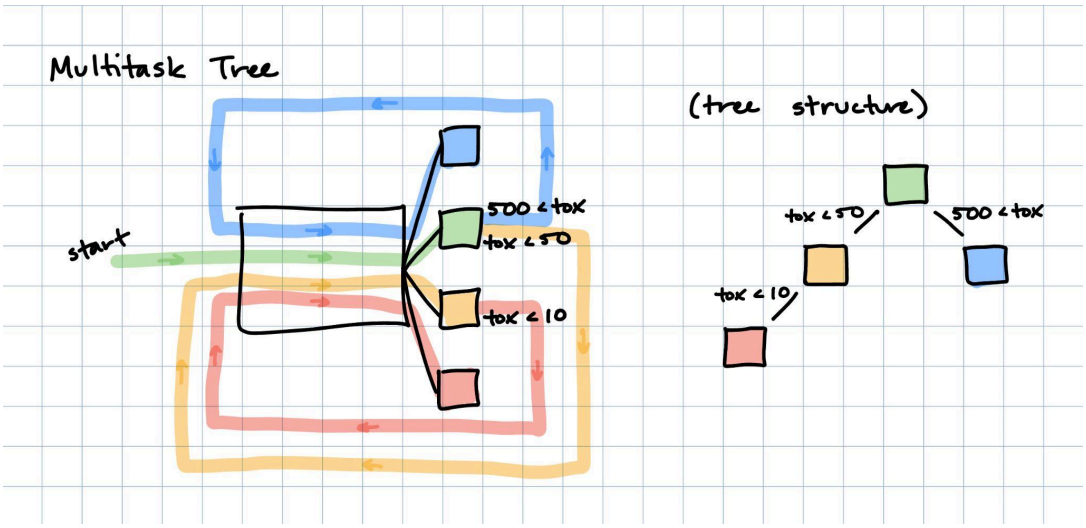
The training script is the same one I used above for the separate binary models (…/binary_runscript.py), since it has code where, if the model type specified in the config file was "tree", it would filter the data according to the cutoff, also specified in the config file (…/binary_config.yaml). The code for this is in the Data Preprocessing section of the runscript. The inference runscript is at …/inference_runscript.py. There is code that specifies which weights to load in based on model type, cutoff, and seed index, as well as code for child tree models which load in the specific data from their parent models predictions. Those data files follow the naming convention "tree_input_[cutoff]_[seed index].csv" and can be found in the …/data folder.





Results and Discussion: The tree model for cutoff 50 had a higher f1 score than the separate binary cutoff 50 model (bar plots can be found below in the Overall section). However, the cutoff 10 and 2000 models did worse (cutoff 500 is the same model). Trying the tree model with a weighted loss function did not improve performance. We thought that even though the tree models had less data to work with, that data was more evenly distributed so maybe the model would be able to perform better on its specific task. However, in general this seemed not to be the case. It is interesting that the tree cutoff 50 and 2000 models had the same depth but the 50 one did better than its counterpart while the 2000 one did worse. Maybe having more toxic (less than 2000) data points was actually beneficial to the separate binary model and we just took away that extra data for the 2000 model. We would have to actually look at the model predictions to determine if that's the case. Another reason for decreased performance could be uncertainty: since the compounds that each "child node" model receive come from predictions of its "parent node" model, there's an uncertainty that those compound toxicity values are actually in the range those child models are supposed to work with. Theoretically, those compounds in the wrong range should have a different structure or position in the embedding space that the child models haven't seen before so it's more likely for them to perform badly with those predictions. Since the tree cutoff 10 model has 2 parent models, it has the most uncertainty, so it makes sense that it has a larger difference in performance from its separate binary counterpart than the cutoff 2000 model.

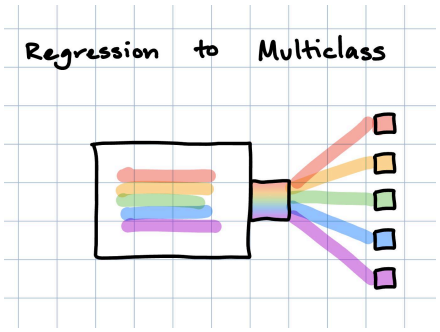
Multi-task Tree (multitasktree):

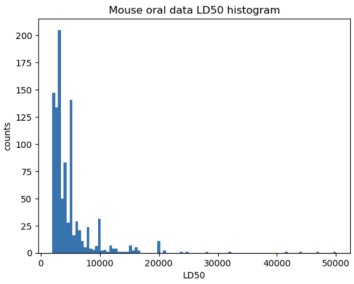


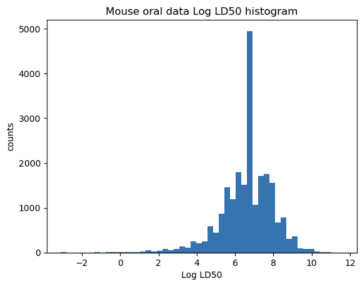
I then tried a different version of the tree idea where instead of 4 different models, it was just one multitask model with four different output layers, one for each node of the tree. Instead of a model for each node, there was a task for each node. Each task had a different DataLoader. Those DataLoaders were the same respective ones that each of the tree models above used. The task ID specifying which output layer to use was determined by the DataLoader. Each total model epoch was a sequential run of one epoch from each of the four DataLoaders. I did this for simplification so I wouldn't have to worry about matching up batches or batch sizes like I did in the multitask model above with all the different species. (NNModel structure at …/multitask_nnmodel.py and run script at …/multitask_runscrip.pyt)

Results and Discussion: I didn't train this model with a weighted loss function so that is a possible path to explore but I'm not sure if it will improve performance since it hasn't for the past models. When I trained and validated this model, I got worse performance than the previous model, so I didn't think it would be worth it to formally compare inference runs. But if you ever want to revisit that for any reason, you'd have to first run inference for all 10 seeds on the cutoff 500 task, save all of those predictions, input them into the next tasks, and so on. I think the main reason this model performed worse than the tree models above is because each of those node models got the whole structure to learn about one cutoff while the multitask outputs had to share those weights and biases. So it wasn't able to capture or learn as much. We were hoping to solve the lack of data problem for the tree models and have them learn from each other but maybe it was too complicated to do so. Maybe you could try this with a larger model, such as adding layers or increasing layer sizes.

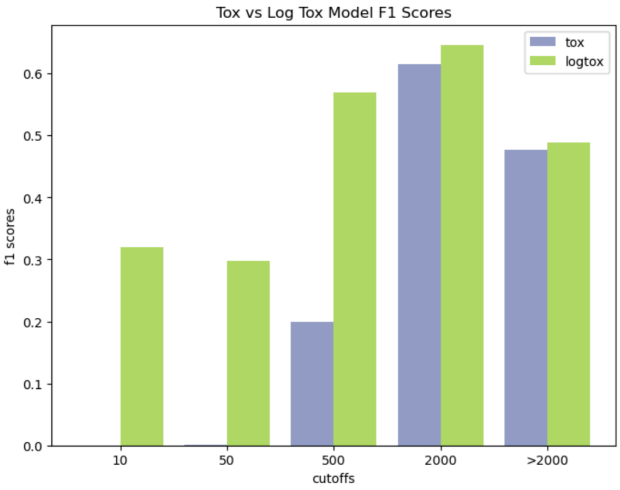
Regression to Multiclass (regmulticlass):





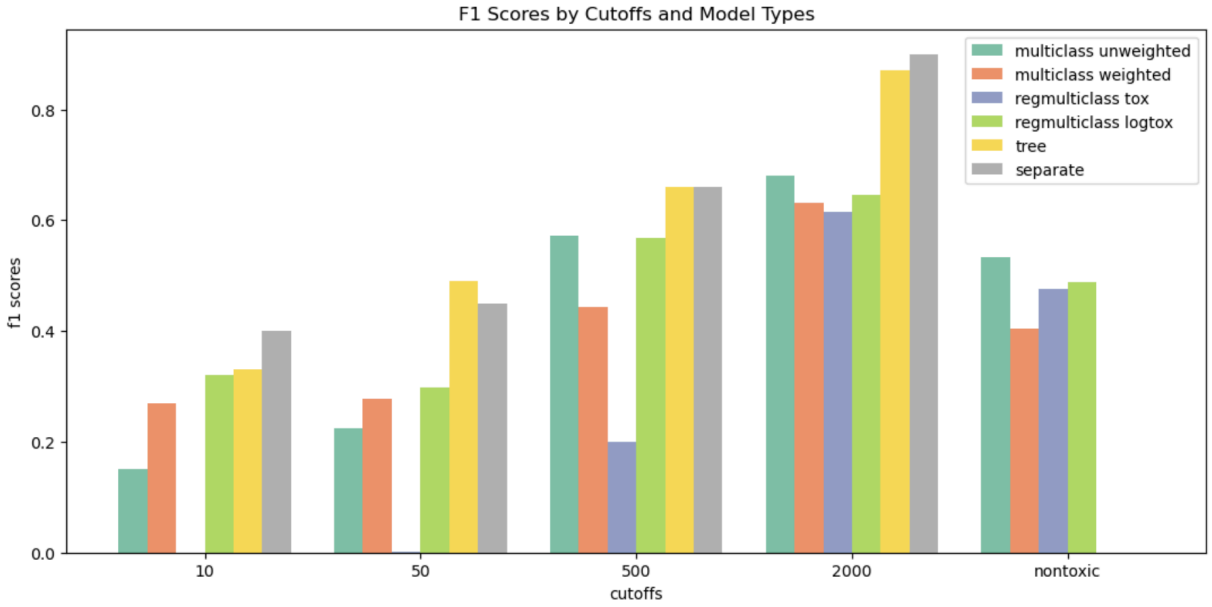


The last architecture I experimented with was a regression model that mapped from continuous toxicity value predictions to toxicity classes. The idea was to let the model be able to learn about the hierarchy and order of toxicity values, which is lost when the objective is to just assign a class. The NNModel structure is in …/regmulticlass_nnmodel.py and the runscript is at …/regmulticlass_runscript.py. I tried this with the original toxicity values as well as the log of them.



Results and Discussion: The original model didn't perform very well. It didn't predict anything to be in the first two classes, which makes sense since the data was so skewed and there were less data points from those classes. The log model did better than the original one, most likely because the distribution was much more normal, which is the best distribution for regression task learning. It does much better for the first three classes but only a little better for the last two. This is probably because there was a lot more data for the last two classes so the original model didn't have trouble identifying them. I think in any case the log values are a much better label to use as the normal distribution is much better for machine learning.

Overall:



Plot of f1 scores for all models.

Notice that the tree, and separate binary models don't have a bar for the nontoxic section. This is because it doesn't apply to what the models were trained on. Instead of a 0 f1 score, it's an N/A.

Overall, the separate binary models did the best. The tree model also did well, and it even did better than the separate binary for cutoff 50. I'm not sure if this was due to luck or if the model is actually better with this specific architecture and class. One way to check that could be to run these models with different data and see if the differences in performance stay the same. Once you've trained all those models you could take the best model from each cutoff and use them as an ensemble. It seems that in general binary models do better than multiclass or regression models for this kind of toxicity problem, probably because they have more nodes to focus on less tasks.

Organ Systems:

(Note: all the file paths in this section are relative to /lus/eagle/projects/datascience/lvairus/Repo_copy/MolFormer/Organs)

To get an idea of the data I had available to me I did some quick training on the side for most of the datasets in the TOXRIC database. For each one I conducted a hyperparameter optimization sweep to get a better idea of the learning potential for each dataset. The wandb project where all those sweeps were run can be found here, and there's a quick "best f1 score"

table on the rightmost side of the TOXRIC statistics sheet. F1 score was the main metric I used to evaluate performance but you can edit the graphs on wandb to show different metrics such as precision and recall.

The hyperparameters I primarily swept through were:

- Encoding size: 256, 512, 1024
- Activation function: relu, gelu, selu
- Linear layer type: OrthoLinear, XavierLinear

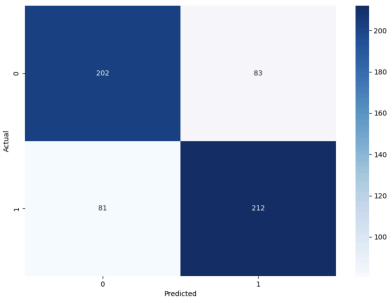
The layers of the model were set up so that they would decrease by half from the encoding size until 16, then to whatever the config's "output_size" was. You can find the code for this model at …/classification_nnmodel.py.

I mainly found that the best options were embedding size 1024, activation function ReLU, and layer type XavierLinear, though there were exceptions and sometimes there wouldn't be that big of a difference in performance. More specific information about which organs had the best performance with which hyperparameters can be found in the wandb project sweeps linked above.

I compiled a folder in Polaris at /lus/eagle/projects/datascience/lvairus/Pharmacokinetic_Modeling/ModelTraining/models_for_archit with the runscript and the necessary config files to run the best model I found for the TOXRIC datasets. The yaml files have the link to the original existing run and sweep as well as the f1 score it should result in.

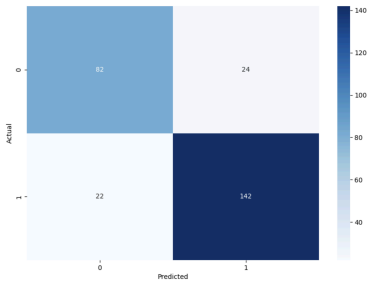
Hepatotoxicity

The hepatotoxicity data on TOXRIC consists of a single file with 2889 compounds with their SMILES string and a binary 1 or 0 indicating toxicity. 50% of the compounds are toxic. The highest f1 score it got was 0.7416. This isn't too bad but it could also be better. To improve it you could try different hyperparameters, a different type of model, or add features or information such as binding score with related proteins.



Respiratory toxicity

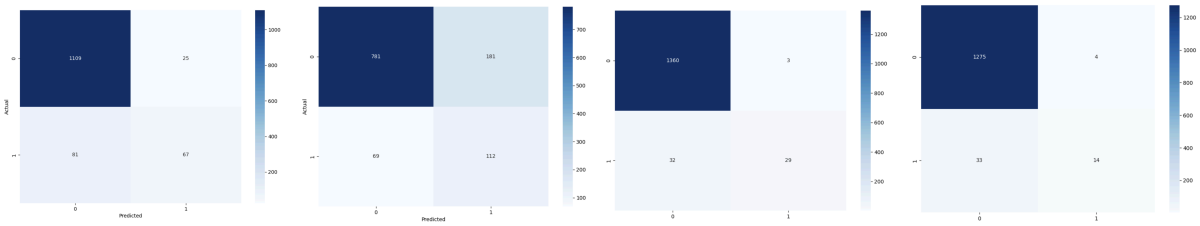
The respiratory toxicity data on TOXRIC consists of a single file with 1348 compounds with their SMILES string and a binary 1 or 0 indicating toxicity. 60% of the compounds are toxic. The highest f1 score it got was 0.8606, which is greater than the hepatotoxicity data. This is interesting because this dataset has less compounds and a less balanced toxicity ratio than the other. It could be by chance, or maybe respiratory data is simpler or easier to learn, such as there are more specific compounds with a shared



structure or embedding space that cause respiratory toxicity, while the structure for compounds that are hepatotoxic varies more.

Endocrine Disruption

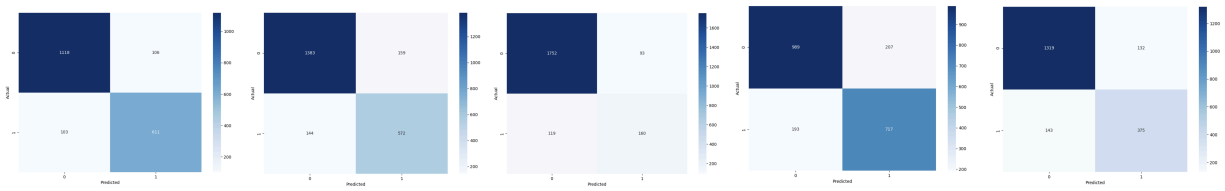
The endocrine disruption data on TOXRIC consists of 12 files, each with a different nuclear receptor or stress response target. They each have around 6000 compounds with their SMILES string and a binary 1 or 0 indicating toxicity. The percentage of toxic compounds varies from around 3% to 16%. Since they were so unbalanced, the models trained on them didn't do well so I only ran a few before moving on to the next dataset. I didn't try a weighted loss function so that could be a possible next step but there are so few toxic data points (200-900) that I'm not sure if the machine would learn anything regardless. I trained these before deciding to use f1 as a metric so the current sweeps measure by AUC or recall. AUC was between 0.80-0.90 but I think they were only high because the data is so unbalanced toward the nontoxic data points. These models were not predicting toxic compounds reliably. The recall scores never exceeded 62% so they were not predicting almost half of the toxic compounds to be toxic.



NR_AhR *SR_ARE* *NR_AR* *NR_AR_LBD*

CYP450

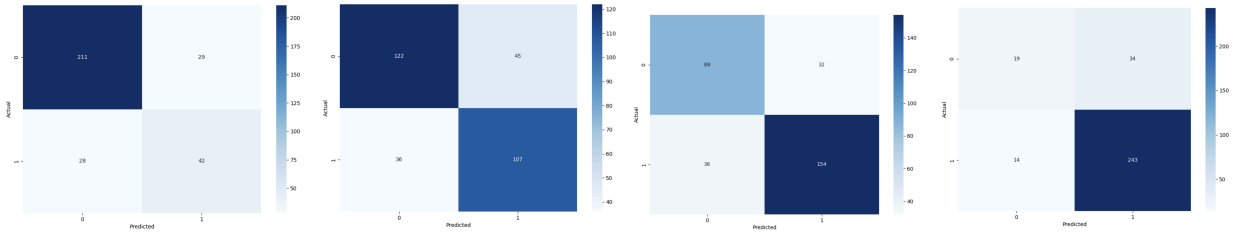
The CYP450 data on TOXRIC consists of 5 files, each with a different CYP450 enzyme target. They each have around 10,000 compounds with their SMILES string and a binary 1 or 0 indicating toxicity. The percentage of toxic compounds varies from around 10% to 40%. The models trained on these datasets performed better, with an f1 score between 0.75-0.85, with the exception of one scoring 0.62. This one was the most imbalanced (CYP2D6, 13% toxic) so it makes sense that it did worse. I believe the other datasets performed much better than the endocrine models because they were slightly less unbalanced, had more compounds, and were specific to a single enzyme. Given how multitask hasn't improved results in previous experiments, I'm not sure how successful this would be, but since these enzymes are all CYP450 enzymes, they might have some shared characteristics so you could try to train a multitask model on all of them.



CYP1A2 *CYP3A4* *CYP2D6* *CYP2C19* *CYP2C9*

Cardiotoxicity

The cardiotoxicity data on TOXRIC consists of 4 files, each with the same 1547 compounds and their SMILES but with a different IC50 cutoff that determines toxicity as 1 or 0. The cutoffs are 1, 5, 10, 30 mg/kg, and the percentage of toxicities are about 20%, 40%, 60%, and 80%. Toxicity was measured by IC50 of hERG (what concentration inhibited the hERG gene by half). I trained a binary classification model for each cutoff. The f1 scores were about 0.60, 0.70, 0.80, and 0.90 respectively. This makes sense as each subsequent cutoff had more toxic data points so the model leaned more towards toxic predictions. However, for the last model (cutoff 30), although it got a high f1 score it did not predict nontoxic compounds well. I didn't log metrics that tracked negative predictions but if you want to put more importance on avoiding false positives, you might want to do that. Since we're working with toxicity, false negatives are more costly so I just focused on metrics involving positive predictions.



Cutoff 1 *Cutoff 5* *Cutoff 10* *Cutoff 30*

Transfer Learning: Mouse to Human

I wasn't able to finish this project but we wanted to try transfer learning from the mouse oral LD50 data to the human oral TDLo data. The mouse data has 20,000 datapoints while the human datasets have less than 200 each so we thought the model might be able to use the mouse data to learn about toxicity, and then fine tune it on the human data so it learns about toxicity in humans specifically.

I pretrained several mouse models on a different hyperparameters. Those weights are at /lus/eagle/projects/datascience/lvairus/Pharmacokinetic_Modeling/ModelTraining/Molformer_Transfer/best_model_weights. The wandb sweep for those runs can be found here. The hyperparameter options were the same ones I used in the Organ Systems sections:

- Encoding size: 256, 512, 1024
- Activation function: relu, gelu, selu
- Linear layer type: OrthoLinear, XavierLinear

I dont know if the mouse model with the best f1 score will necessarily transfer the best for humans, so I think it would be interesting to transfer learn with all, or just multiple, pretrained mouse models and see how they do in comparison. The human data on TOXRIC is 3 files in the acute toxicity section: women oral TDLo, men oral TDLo, and human oral TDLo. However they all record pretty different toxicity values for the same compounds, possibly because of differences in how the data was collected. Therefore it might be better to train them in separate models. I started with a separate binary cutoff 2000 model because that did the best from all the architectures in the Mouse Models section and according to the EPA, 2000 is the cutoff between "slightly toxic" and "practically nontoxic" compounds for humans. In the future you could also try adding rat data to the pretrained model because that has 10,000 more data points that might help the model learn even more about toxicity.

Workflow

YAML config file

- Instead of having an argparse for hyperparameters, which didn't work too well when trying to implement with wandb, I switched to using a yaml file.
- Instead of

```
Python
	args = parser.parse_args()
```

I use

```
Python
	args, unknown_args = parser.parse_known_args()
```

Because wandb sweep agents call the run script using argparse arguments from the config file, which won't be recognized by the code.

Reproducibility

- Added code for making runs reproducible with a seed
- Source: Reproducible Deep Learning Using PyTorch | by Darina Bal Roitshtain
- 10 seeds were produced randomly with numpy: [53844, 837465, 800662, 910250, 543584, 179839, 707873, 482701, 278083, 198125]

- Code is in almost all of my run scripts, for example: /lus/eagle/projects/datascience/lvairus/Repo_copy/MolFormer/Organs/binary_runscript.py

RoundRobinBatchSampler

- if you'd like to use a batch sampler that specifies number of batches instead of batch size, you can use the code at /lus/eagle/projects/datascience/lvairus/Repo_copy/MolFormer/Multitask_Class/data_utils.py

Weights and Biases (WandB):

- I used wandb to plot metrics and keep track of model configurations
- It was extremely helpful, pretty simple to implement, there was a lot of documentation and resources online, and it had a lot of functionality for data visualization (such as filtering and averaging runs)
- Quick tutorials:
  - Welcome to Weights & Biases - Introduction Walkthrough (2020)
  - 🔥 Integrate Weights & Biases with PyTorch
  - 🧹 Tune Hyperparameters Easily with W&B Sweeps
- Custom bar graph: I figured out how to plot my own bar graphs
  - First I made a pandas df with the data I wanted (in this example performane_df), then make a wandb table with it

```python
performance_table = wandb.Table(dataframe=performance_df)
```

  - Then log a bar chart using data from that table. In this case "task" is the column name for the labels and "best_avg_ep_AUC" is the column with the values.

```python
wandb.log({"best_avg_ep_AUC" : wandb.plot.bar(performance_table, "task", "best_avg_ep_AUC", title="best_avg_ep_AUC")})
```

  - Full code at the bottom of /lus/eagle/projects/datascience/lvairus/Repo_copy/MolFormer/Multitask_Class/run_script_lv_multi.py
- Sweeps:
  - Wandb lets you submit a yaml file with all the hyperparameter options you want to try and automatically runs and logs each configuration

- I first link my runs to a regular config file to provide it with default values. Then I initialize a sweep using a sweeping config file to then submit agents to carry out the hyperparameter configurations.
    - For example, in /lus/eagle/projects/datascience/lvairus/Repo_copy/MolFormer/Organs:
    - Run script: …/binary_runscript.py
    - Regular config: …/binary_config.yaml
    - Sweeping config: …/binary_sweep.yaml
    - Bash script: …/binary_sweep_qsub.sh
- I had some trouble getting parallel wandb sweep runs in the polaris queue, since you have to first initialize the sweep and then submit an agent to actually run the sweep.
- A workaround I got was that I made a sweep config yaml file, initialize the sweep in the terminal with the code below, this would give me the ID for the sweep, then I'd just copy that into my .sh file and submit that.

```
Python
wandb sweep sweep_config.yaml
```

    - Example sweep config file: /lus/eagle/projects/datascience/lvairus/Repo_copy/MolFormer/Organs/binary_sweep.yaml
    - Example .sh file: /lus/eagle/projects/datascience/lvairus/Repo_copy/MolFormer/Organs/binary_sweep_qsub.sh
- You can cancel sweeps with the following code

```
Python
# format:
`wandb sweep --cancel entity/project/sweep_id`
# example:
`wandb sweep --cancel lvairusorg/Toxric/05ixe3i2`
```

Conclusions / What I Learned

- Although machine learning is very powerful, it still can't do everything, especially when faced with complex problems like drug toxicity. How you structure your models and input your data heavily influences their performance.

- The more specific to a system/protein a dataset is, and the more balanced it is between classes, the more likely it is to perform better.
    - For example, the acute toxicity datasets only specified an administration route of the compounds while the cardiotoxicity dataset specified the inhibition of a specific gene
    - Even though the organ system datasets had less points, they still performed comparably to the mouse oral toxicity data, which had 20,000 points.
- I learned a lot about implementing machine learning using pytorch
    - Modified existing models, such as changing a regression model to a classification one.
    - Made training runs reproducible by seeds
    - Implemented a custom batch sampler
- I figured out how to run Conplex predictions, though the results were always small and didn't correlate with my data and hypotheses.
- I found that multiple binary classifiers have higher f1 scores than multiclassifiers and regression models
- The best hyperparameters are usually embedding size of 1024, activation function ReLU, and layer type XavierLinear, but this can vary and sometimes the differences in performance are small.
- In general, future directions for all the model projects above include: finding more data, trying different hyperparameters/architectures, and adding more features to compound inputs, such as their binding score with related proteins.
- I learned how to to use wandb to track configurations, log metrics, and automate hyperparameter sweeps, and I realized how important/helpful it is to have a good system for tracking hyperparameters
- I got a lot of practice with summarizing and presenting my work with the biweekly intern presentations, which was very helpful
- I realized that code organization and documentation is extremely important when it comes to understanding, even if it's your own code, as you can write a lot over a long period of time and having good organization helps you look back at and understand code you've already written.
- I got better at reading papers. It felt like I had learned a new language when I finally understood what information I wanted to get out of a paper and knew where to look or when I found it. I also learned how important it is to read papers, since they have so much information on things related to what you're working on that could help.
- It's bad practice to copy entire folders, especially when I don't know what's in it/how big it is.