CSCI 555: Toxicity Prediction Challenge II

Report

Name: Sanket Dilip Vagal

StFX Student ID Number: 202207184

StFX email ID: x2022fjg@stfx.ca

Introduction:

The Toxicity Prediction Challenge II aims to tackle a real-life challenge - predicting the toxicity of chemicals using machine learning. With the constant synthesis of new chemicals, it is crucial to predict their toxicity before releasing them into the market. Traditionally, this has been done using in-vivo methods, which are time-consuming and involve animal testing. However, in-silico techniques offer a promising alternative to reduce time, cost, and animal testing. This dataset, prepared using a resource that includes data for around 9,000 chemicals and over 1500 high-throughput assay endpoints, can be used to create models that accurately predict toxicity, which could potentially have a significant impact on the safety of new chemicals.

Data preparation:

The data is provided as train data (train_II.csv) and test data (test_II.csv). The given train data has 2 columns, 'Id' and 'Expected', where the 'Id' has the SMILES and Assay IDs delimited by a semicolon. The 'Expected' column is the label indicating whether the given sample is toxic or not, denoted as 1=toxic, 2=non-toxic. The test data is similar, except the 'Id' column is named as 'x' and it does not have an 'Expected' (labels) column.

First, we separated the train's 'Id' column and test's 'x' column into two new columns, 'SMILES' and 'Assay ID' respectively, using python's split method.

The Assay ID column has the data in 'object' datatype. To make the datatypes uniform, we converted the Assay ID column data to numeric datatype.

SMILES by itself do not give any information on whether there are any "invalid" SMILES. To do so, we converted the SMILES in train and test data to their Molecular representations using RDKit. Doing so, we found some molecules which had invalid valency. These molecules were dropped altogether to avoid incorrect predictions.

The 'Expected' column was separated from the train dataset to create labels for validating and fitting the model.

Feature Selection:

RDKit provides numerous methods which return various descriptors for a given SMILE. Other than Assay ID, we experimented on approximately 1500+ different features created by RDKit, out of which 1200 were selected. The features are:

- **num_atoms**: number of atoms in the molecule
- **mol_weight**: molecular weight of the molecule
- **logp_i**: calculated logarithm of the partition coefficient of the molecule
- **h_bond_donor_i**: number of hydrogen bond donors in the molecule
- **h_bond_acceptors_i**: number of hydrogen bond acceptors in the molecule
- **rotb**: number of rotatable bonds in the molecule
- **tpsa**: topological polar surface area of the molecule
- **heavy**: number of heavy atoms in the molecule
- **morgan_{i}**: ith bit in the Morgan fingerprint of the molecule (1024 features)
- **maccs_{i}**: ith bit in the MACCS keys fingerprint of the molecule (166 features)
- **aromatic_cc**: number of aromatic carbocycles in the molecule

- **peoe_vsa1**: partial electrostatic potential-derived atomic surface area of the molecule.

These features were calculated for the SMILES present in both train and test data.

To further reduce the number of features, Variance Threshold feature selection technique is used. We then remove the features with low variance. Here we experimented with threshold values, and finally found a threshold of 0.2 provided a good measure. Using this, 1130 features were ultimately removed, finalising 71 features. These were:

‘num_atoms’, ‘mol_weight’, ‘logp_i’, ‘h_bond_donor_i’, ‘h_bond_acceptors_i’, ‘rotb’, ‘tpsa’, ‘heavy’, ‘morgan_33’, ‘morgan_64’, ‘morgan_80’, ‘morgan_294’, ‘morgan_650’, ‘morgan_659’, ‘morgan_695’, ‘morgan_726’, ‘morgan_807’, ‘morgan_849’, ‘morgan_875’, ‘morgan_893’, ‘maccs_87’, ‘maccs_103’, ‘maccs_106’, ‘maccs_107’, ‘maccs_109’, ‘maccs_110’, ‘maccs_112’, ‘maccs_113’, ‘maccs_117’, ‘maccs_118’, ‘maccs_120’, ‘maccs_121’, ‘maccs_122’, ‘maccs_123’, ‘maccs_124’, ‘maccs_125’, ‘maccs_126’, ‘maccs_127’, ‘maccs_128’, ‘maccs_129’, ‘maccs_132’, ‘maccs_133’, ‘maccs_134’, ‘maccs_136’, ‘maccs_137’, ‘maccs_140’, ‘maccs_141’, ‘maccs_142’, ‘maccs_143’, ‘maccs_144’, ‘maccs_145’, ‘maccs_146’, ‘maccs_147’, ‘maccs_148’, ‘maccs_149’, ‘maccs_150’, ‘maccs_151’, ‘maccs_152’, ‘maccs_153’, ‘maccs_154’, ‘maccs_155’, ‘maccs_156’, ‘maccs_157’, ‘maccs_158’, ‘maccs_159’, ‘maccs_160’, ‘maccs_161’, ‘maccs_162’, ‘aromatic_cc’, ‘peoe_vsa1’

Model training/testing:

To train the data on a machine learning model, we tried various machine learning models, including Random Forest classifier, Histogram-based Gradient Boosting Classification Tree, XGBoost, LightGBM, etc. The stacking combinations of these models were also experimented with.

All these models were internally validated using a 5 split Stratified Cross validation technique. This approach splits the train data into 5 folds and trains the model 5 times using each fold as a validation set and the remaining data as a training set. The use of StratifiedKFold ensures that the distribution of the target variable (labels) is approximately equal in each fold. The shuffling and random state parameters are used to randomly shuffle the data before splitting and set a seed value for reproducibility. This provides a more reliable estimate of the model's performance than a single train-test split because it uses multiple splits and averages the results, which reduces the variance of the estimate.

Out of all the models which were experimented with, LightGBM performed the best, and hence was selected as the final model. To further improve the performance, we performed GridSearchCV. GridSearchCV is a tool for hyperparameter tuning that searches for the best combination of hyperparameters in a defined search space, using a specified scoring metric and cross-validation. After fitting the GridSearchCV object to the training data, the best hyperparameters found during the search were max_depth = 12 and n_estimators = 1600.

As the given data is highly imbalanced, with a 6:1 ratio of both classes of samples, scale_pos_weight=0.6 is used. This is a parameter that can be used to adjust the balance between the positive and negative classes in the training data. In this case, it sets the weight for the positive class to be 0.6 times that of the negative class. A random seed was also set to enable the reproducibility of the code.

The final LGBM model was defined as:

```
lgb.LGBMClassifier(
   max_depth=12, n_estimators=1400, scale_pos_weight=0.6, random_state=42
)
```

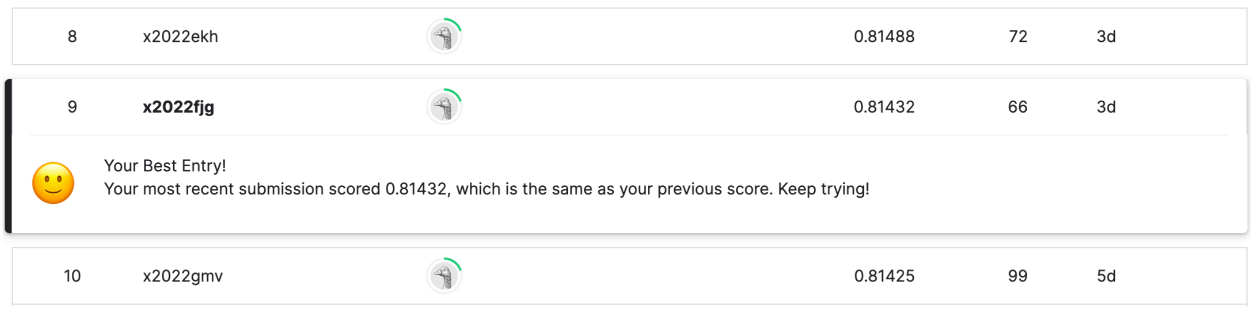
To evaluate this model's performance, the previously mentioned 5 split Stratified Cross validation was used. It resulted in a mean F1 score of **0.80561**. The public leaderboard score for this submission was **0.81432** and private leader score was **0.82280**.

Leaderboard:

Public leaderboard:

Rank: 9

Score: 0.81432



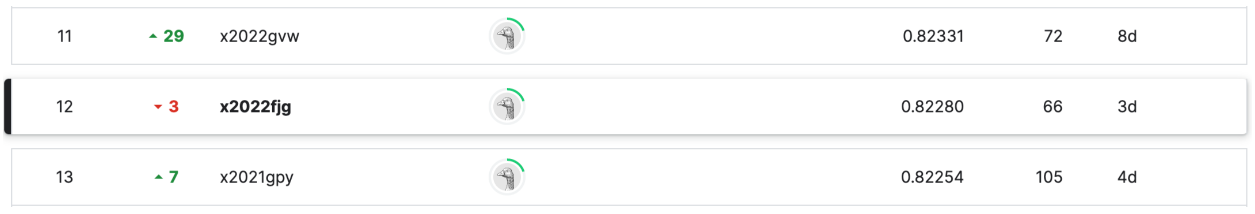
8	x2022ekh	0.81488	72	3d
9	**x2022fjg**	0.81432	66	3d
10	x2022gmv	0.81425	99	5d

Your Best Entry!
Your most recent submission scored 0.81432, which is the same as your previous score. Keep trying!

Private leaderboard:

Rank: 12

Score: 0.82280



11	▲ 29	x2022gvw	0.82331	72	8d
12	▼ 3	**x2022fjg**	0.82280	66	3d
13	▲ 7	x2021gpy	0.82254	105	4d

Code:

For Final submission, we have created a Docker image, which can be found in our repository here:

https://github.com/sanketvagal/the-toxicity-prediction-challenge-ii

To execute the project as a docker image or as a script, follow the instructions in the README.md provided in the repository.

Alternatively, the code as a python notebook can also be viewed on Kaggle here:
https://www.kaggle.com/code/sanketvagal/x2022fjg