

REPORT ON TOXICITY PREDICTION

Karthik Reddy M

202205247

INTRODUCTION

It is impractical to determine toxicity using conventional laboratory animal testing because thousands of novel compounds are created every day. The use of "in-silico" procedures is a fantastic substitute for "in-vivo" approaches since they could cut down the time, expense, and animal testing associated in testing the toxicity. This dataset was created utilizing a database that contains information for over 1500 high-throughput test endpoints covering a variety of high-level cell responses, together with data for over 9,000 compounds.

In this toxicity prediction challenge hosted on Kaggle, the main goal is to predict whether a chemical is toxic or non-toxic using machine learning models.

STEP 1: PREPARING DATASET

- Loading the train and test file which are downloaded from Kaggle using pandas.
- Dataset contains two columns:
 - ID
 - Expected
- ID is combination of chemical compound and Assay ID.
- Splitting the ID into two columns namely Id1 and Id2.
- Generating smiles for the chemical compounds using Rdkit.
- For some chemicals we don't have smiles, so we will just drop those rows and reset the index.
- Writing a function to get all the 209 descriptors for the smiles passed as input.
- This function will return Descriptors names and those values for all smiles.
- Created a data frame by merging those names and values.
- After creating a data frame adding Id1, Id2, smiles and Expected value to create a final data frame.
- Exporting the final data frames.

STEP 2: DATA PRE-PROCESSING

- Imported the train and test descriptor files which are created in the previous step.
- The train and test data has around 4000 null values each for 8 columns and 400 each for 4 columns

```

null_counts = train.isnull().sum()
# Filter for columns with non-zero null counts
null_cols = null_counts[null_counts > 0]
print(null_cols)

```

```

MaxPartialCharge      420
MinPartialCharge      420
MaxAbsPartialCharge   420
MinAbsPartialCharge   420
BCUT2D_MWHI           3813
BCUT2D_MWLOW          3813
BCUT2D_CHGHI          3813
BCUT2D_CHGLO          3813
BCUT2D_LOGPHI         3813
BCUT2D_LOGPLOW        3813
BCUT2D_MRHI           3813
BCUT2D_MRLOW          3813
dtype: int64

```

- To handle the null values, we are filling the null values with mean of those columns.
- After handling the null values, the dataset looks like below.

```
train.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 75377 entries, 0 to 75376
Columns: 213 entries, MaxEStateIndex to Expected
dtypes: float64(104), int64(106), object(3)
memory usage: 122.5+ MB

```

STEP 3: FEATURE SELECTION

- Removing the high correlated features whose correlation values is greater than 0.8.
- After removing the highly correlated columns, we are left with 150 features.

```
train.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 75377 entries, 0 to 75376
Columns: 151 entries, MaxAbsEStateIndex to Expected
dtypes: float64(65), int64(83), object(3)
memory usage: 86.8+ MB

```

- Used Recursive feature elimination technique to select top features.
- To know the number of the top features the above step need to return, I used Grid Search CV to find the count of top features.

- I gave n value as 60 to Decision tree classifier in recursive feature elimination technique.
- It gave top 60 features as output.

```
selected_features #without id and no append
```

```
Index(['MaxAbsEStateIndex', 'MinAbsEStateIndex', 'qed', 'MinPartialCharge',
      'MaxAbsPartialCharge', 'MinAbsPartialCharge', 'FpDensityMorgan3',
      'BCUT2D_MWHI', 'BCUT2D_MWLOW', 'BCUT2D_CHGLO', 'BCUT2D_LOGPHI',
      'BCUT2D_LOGPLOW', 'BCUT2D_MRHI', 'BCUT2D_MRLOW', 'BalabanJ', 'Chi4v',
      'HallKierAlpha', 'Ipc', 'Kappa2', 'Kappa3', 'PEOE_VSA1', 'PEOE_VSA10',
      'PEOE_VSA12', 'PEOE_VSA14', 'PEOE_VSA5', 'PEOE_VSA6', 'PEOE_VSA7',
      'PEOE_VSA8', 'PEOE_VSA9', 'SMR_VSA1', 'SMR_VSA6', 'SlogP_VSA1',
      'SlogP_VSA11', 'SlogP_VSA2', 'SlogP_VSA5', 'EState_VSA10',
      'EState_VSA2', 'EState_VSA3', 'EState_VSA4', 'EState_VSA5',
      'EState_VSA6', 'EState_VSA7', 'EState_VSA8', 'VSA_EState10',
      'VSA_EState3', 'VSA_EState4', 'VSA_EState5', 'VSA_EState7',
      'VSA_EState8', 'VSA_EState9', 'FractionCSP3', 'NumHeteroatoms',
      'MolLogP', 'MolMR', 'fr_allylic_oxid', 'fr_aniline', 'fr_benzene',
      'fr_ester', 'fr_nitro_arom', 'fr_thiazole'],
      dtype='object')
```

MODELLING:

- The train data is split in 80% and 20%
- After splitting the data, it is passed as input to the Cat Boost model to fit.

CatBoost:

CatBoost is an algorithm for gradient boosting on decision trees. Reduce time spent on parameter tuning, because it provides great results with default parameters. It reduces overfitting when constructing your models with a novel gradient-boosting scheme. Apply your trained model quickly and efficiently even to latency-critical tasks using CatBoost's model applier.

- After training the data, I passed the test data to predict, it gave an accuracy around 91 % and f1-score of 66
- I used F1 score for internal validation.

```
f1= f1_score(y_test, y_pred)
print(f1)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
```

```
0.666497975708502
```





```
Accuracy: 0.9125762801804193
```

PREDICTIONS

- For predictions, we have created a new variable “prediction” and stored the data using `pd.DataFrame()` pandas function.
- I stored the predictions in `y_pred` variable.
- To generate the output csv file, I extracted the column `x` from test data file and concatenated it with `y_pred`.
- After concatenating using `to_csv` I exported the csv file.

LEADERBOARD SCORE:

In this competition I made around 158 submissions and scored around 82.798 in private and 81.601 in public leader board.

#	△	Team	Members	Score	Entries	Last	Solution
1	▲ 6	x2022fic		0.83242	38	8d	
2	▲ 27	x2022fij		0.83168	46	2d	
3	▲ 13	x2021guf		0.82871	53	3d	
4	—	x2022fip		0.82798	158	3d	