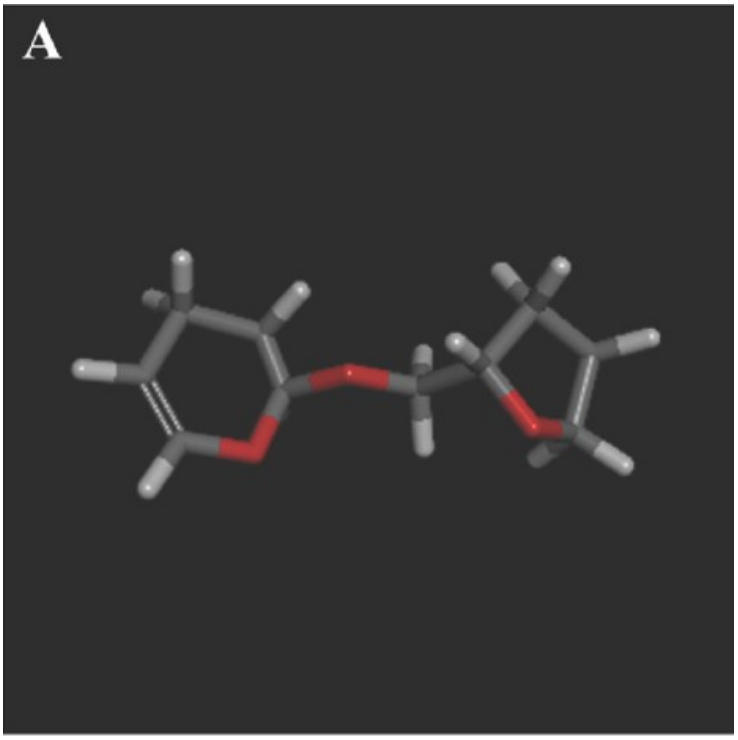
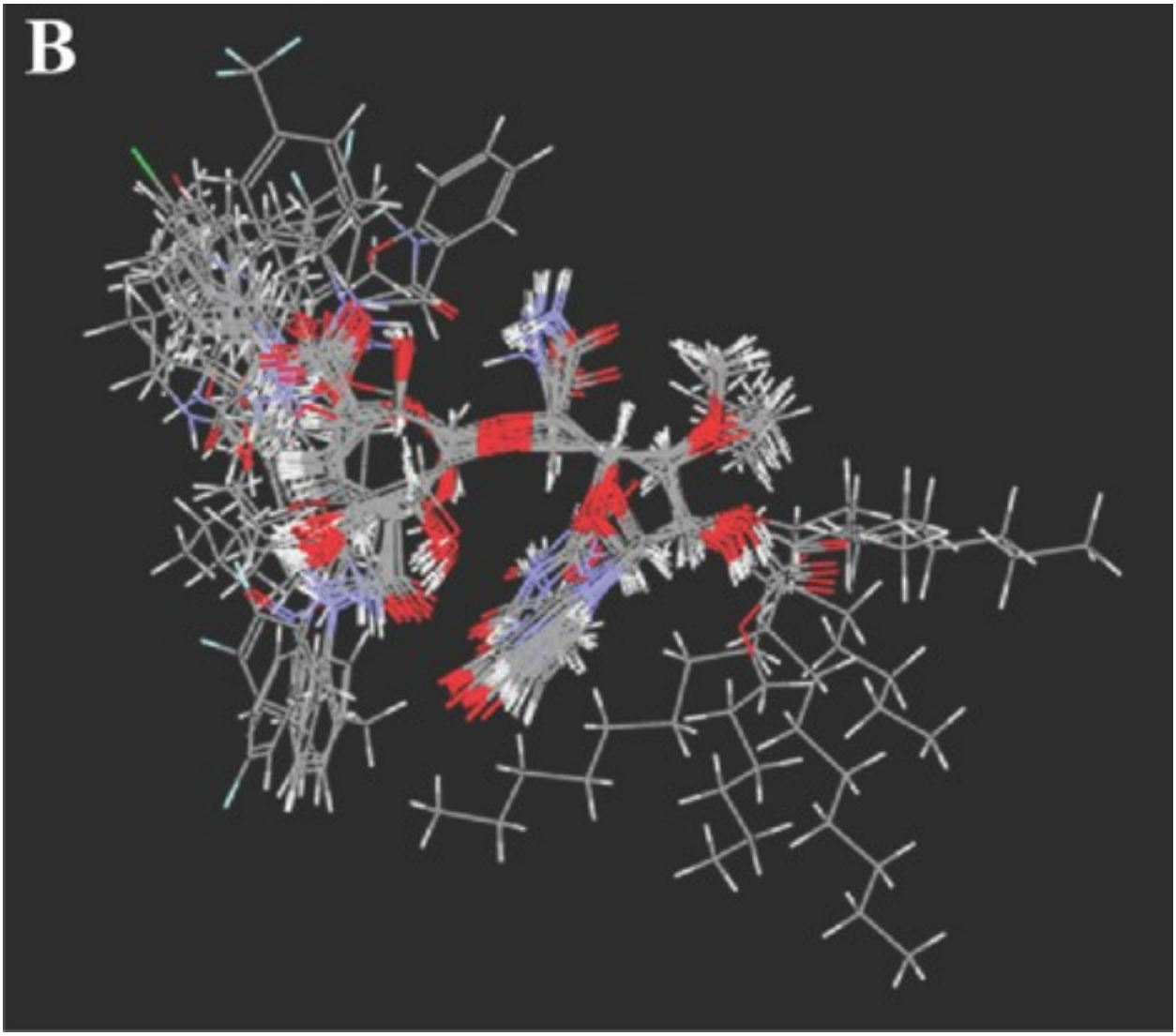
# Applications of Recent Advances in AI to the Environment and Drug Discovery

Prepared by
Al Mercado, AI engineer consultant at Retention Inc, New York, NY

*Rational drug design is still considered to be challenging, expensive, and time consuming. Usually computational tools, and methodologies for a structure guided approach are used for the target of the drug, which is a biomolecule. It is beyond the scope of this case study to discuss all aspects of drug discovery but a brief overview is provided for a computer aided drug design approach.*



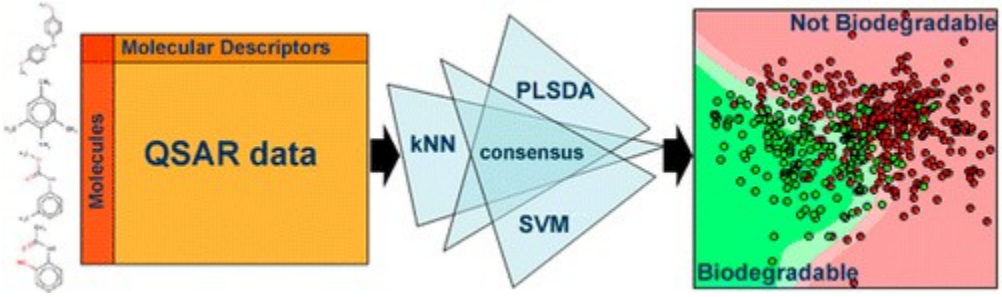
*Quantitative Structure Activity Relationship (QSAR) modeling in drug research software, such as AutoQSAR by Schrodinger, is one of the tools for generating improvement of drug like properties for protein targets. The target in a protein is typically identified by the docking interactions for a potent drug.*




*The QSAR model determines the relationship between chemical structure and biological activity and predicts the activities of new molecules. The reactivity of a drug like molecule includes its biodegradation profiles and they can alternatively be used in assessing the environmental impact of new drug candidates such as toxicology.*

.

QSAR models for the **understanding** of molecular interactions have been implemented for assessing the biodegradability of various chemicals. The core objective of the initial study by researchers, Quantitative Structure–Activity Relationship Models for Ready Biodegradability of Chemicals, was to develop various classification models for biodegradability to compare with QSAR models for degradation. By comparing k nearest neighbors, partial least squares discriminant analysis, and support vector machines with QSAR models the researchers demonstrated good classification performance such as accuracy.




The goal of this new AI driven use case study was to build on the findings of the original researchers and update their initial results using the latest automatic machine learning libraries. It was to find the best models for classification from the entire spectrum of open source libraries. This goal has been achieved with the use of the TPOT Classifiers library and H2O AI's AutoMl on the original data

# Using  TPOT as the First Test for Automating ML Pipeline Builds

In addition to leveraging data science libraries like the  Python-based scikit-learn the Tree-Based Pipeline Optimization Tool (TPOT) developed by University of Pennsylvania Researchers can also automate the building of ML pipelines with stochastic search algorithms such as genetic programming.

After  installing it to  the Jupyter Notebook the tuning parameters were  set for the model pipeline with the API.

```
!pip install deap update_checker tqdm
```

```
!pip install tpot
```

All the definitions for the predictors were defined in the QSAR dataset description in the UCI repository.

```
df_bio=pd.read_csv("biodeg_qsar.csv")
```

The file was read in with the pandas library as a dataframe. The target column for the predictors in the dataframe had to be transformed to a new column with an integer value from  the dictionary for defining the class to be  biodegradeable (RB) or not.

```
#assign values to features
d = {'RB': 1, 'NRB': 0}
import numpy as np
```

```
y=np.array(df_bio.iloc[:,-1].map(d))
```

In [9]:

```
#Assign features from QSAR values
X = np.array(df_bio.iloc[:, 0:41])
```

The predictors in the dataset for the model were transformed by the preprocessing library in sklearn before splitting it for the train and test phases.

```
#Normalize & scale features
from sklearn import preprocessing as pre
scaler = pre.StandardScaler().fit(X)
X_scaled = scaler.transform(X)
```

In [11]:

```
# split data into train and test sets
seed = 7
test_size = 0.33
```

The sklearn cross validation library was used for randomly splitting the data before running  the classifier pipeline with all the QSAR predictors in the train data set .

```
from sklearn.model_selection import train_test_split
```

In [13]:

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=test_size, random_state=seed)
```

In [16]:

```
pipeline_optimizer = TPOTClassifier(generations=5,
population_size=20, cv=5,random_state=42, verbosity=2)
```

First, the TPot ML pipeline was trained using the default parameters:

In [17]:
```
pipeline_optimizer.fit(X_train, y_train)
```

```
Optimization Progress:  32%|███       | 38/120 [00:18<00:58,
1.39pipeline/s]
Generation 1 - Current best internal CV score: 0.876874010018
Optimization Progress:  47%|████      | 56/120 [00:30<00:50,
1.26pipeline/s]
Generation 2 - Current best internal CV score: 0.876874010018
Optimization Progress:  63%|█████     | 76/120 [00:36<00:13,
3.19pipeline/s]
Generation 3 - Current best internal CV score: 0.876874010018
Optimization Progress:  78%|███████   | 94/120 [00:39<00:06,
4.28pipeline/s]
Generation 4 - Current best internal CV score: 0.876874010018

Generation 5 - Current best internal CV score: 0.881129329166

Best pipeline: LogisticRegression(CombineDFs(input_matrix,
input_matrix), LogisticRegression__C=0.5,
LogisticRegression__dual=True, LogisticRegression__penalty=l2)
```

Out[17]:

```
TPOTClassifier(config_dict={'sklearn.ensemble.GradientBoostingClassifier':
{'max_features': array([ 0.05,  0.1 ,  0.15,  0.2 ,  0.25,  0.3 ,  0.35,  0.4
,  0.45,
        0.5 ,  0.55,  0.6 ,  0.65,  0.7 ,  0.75,  0.8 ,  0.85,  0.9 ,
        0.95,  1.  ]), 'learning_rate': [0.001, 0.01, 0.1, 0.5, 1.0],
'min_samples_... 0.7 ,  0.75,  0.8 ,  0.85,  0.9 ,
        0.95,  1.  ])}, 'sklearn.preprocessing.RobustScaler': {}},
        crossover_rate=0.1, cv=5, disable_update_check=False,
        generations=5, max_eval_time_mins=5, max_time_mins=None,
        mutation_rate=0.9, n_jobs=1, offspring_size=20, population_size=20,
        random_state=42, scoring=None, subsample=1.0, verbosity=2,
        warm_start=False)
```

```
print(pipeline_optimizer.score(X_test, y_test))
```

```
0.859195402299
```

```
In [19]:
pipeline_optimizer.export('tpot_exported_pipeline.py')
```

The progress of the training over five generations showed that the Logistic Regression turned out to have the best fit on the training data. The accuracy score was determined on the test data. If the TPOT classifier ran for more generations, then the score would have improved. However, in order to scale for the increasing size of any new QSAR dataset alternate machine learning libraries have to be implemented instead of the python pandas and the sklearn libraries.

If we had to train on a big data set, H2O ai, an open source, in-memory, distributed, ML and predictive analytics platform, would be the choice for the model build phase. It could easily be implemented in production on a Hadoop platform. The data is distributed across the cluster and stored in memory in a compressed columnar format, allowing you to read the data in parallel as an H2O Frame.


H2O's core code is written in Java
but the model can be developed in Python, R or Scala. Once the categorical target (RB vs. NRB class) in the dataset was encoded, the H2O library was loaded, and an H2O instance was initialized.

```
df_bio_b['bio_class']=df_bio_b.iloc[:,-1].map(d)
```

In [7]:

```
import h2o
```

In [8]:

```
### start the server
h2o.init()
```

```
hf=h2o.H2OFrame(df_bio_b)
Parse progress: |
████████████████████████████████████████████████████████████████| 100%
```

```
train, test = hf.split_frame([0.66], seed=1234)
X = hf.col_names[ 0:41]
```

The same predictors and targets that were used for the training with TPOT were also used for building the H2O models. The AutoML library in H2O eliminates the need to tune hyperparameters and search for the best performing model similar to  a grid search for estimator parameter tuning (GridSearchCV) in Python's scikit-learn. The default parameters for running AutoML were used. H2O requires the datatype transformation of the original categorical targets in both the train and test sets.

```
Y = "bio_class"
from h2o.automl import H2OAutoML
```

In [21]:

```
aml = H2OAutoML(max_runtime_secs = 30)
```

In [22]:

```
train[Y] = train[Y].asfactor()
```

In [23]:

```
test[Y] = test[Y].asfactor()
```

In [24]:

Before the  run with h2o.automl() occurred, the setting used for x and y parameters were the same predictor columns  and target columns from before. The training_frame was the split portion with the  training data and the leaderboard_frame was the validation data set used by H2O in generating the top models that didn't overfit the data.

```
aml.train(x = X, y = Y,training_frame = train,leaderboard_frame
= test)
AutoML progress: |
████████████████████████████████████████████████████████| 100%
Parse progress: |
████████████████████████████████████████████████████████| 100%
lb = aml.leaderboard
```

The results for fitting with the models  were stored in the automl_h2o_models object and the ranking is by top models according to accuracy. In this case we wanted to see how the best model compared to the initial pipeline of model building  using TPOT.

| model_id | auc | logloss |
|---|---|---|
| StackedEnsemble_BestOfFamily_0_AutoML_20180817_193456 | 0.932298 | 0.306657 |
| StackedEnsemble_AllModels_0_AutoML_20180817_193456 | 0.931684 | 0.308195 |
| GBM_grid_0_AutoML_20180817_193456_model_0 | 0.921749 | 0.325291 |
| GLM_grid_0_AutoML_20180817_193456_model_0 | 0.921711 | 0.333788 |
| GBM_grid_0_AutoML_20180817_193456_model_3 | 0.918412 | 0.332507 |
| XRT_0_AutoML_20180817_193456 | 0.91354 | 0.503885 |
| DRF_0_AutoML_20180817_193456 | 0.913042 | 0.505123 |
| GBM_grid_0_AutoML_20180817_193456_model_2 | 0.912582 | 0.333928 |
| GBM_grid_0_AutoML_20180817_193456_model_1 | 0.911623 | 0.345984 |
| GBM_grid_0_AutoML_20180817_193456_model_4 | 0.907058 | 0.35512 |

The best model based on the leaderboard by AutoML is a Stacked Ensemble model with a 93.2% AUC, which was a significant

improvement over the accuracy by the model built with the TPOT library.

However, in a production environment new data would be scored with h2o.predict() and the model can be specified to be the leader or any other benchmark model. The final model can be incorporated in a workflow as an H2O-generated Model Object, Optimized (MOJO) and easily embeddable in any Java environment. The . jar file gets saved with h2o.download_mojo() and implemented in a Java workflow orchestration framework such as Knime. KNIME is one example of a platform built for powerful analytics on a GUI based workflow that runs on the JVM.

It's possible to apply this AutoML approach towards other studies that concentrate on finding the targets through global gene expression data analysis for drug modification and molecular responses to categories of diseases. Some studies have already been conducted where AI and machine learning can improve or complement traditional gene expression analysis approaches. Now the hope is that these models can be extended with little additional effort.