CS573 Project : Toxic Molecule - Prediction

Omkar Patil(patilo@purdue.edu)

May 3, 2018

1 Kaggle Competition Details

Team name : **Jacen Solo**

Leaderboard standing : **6**

AUC on final(70%) test data : **0.86950**

2 Problem Statement

This is a prediction task, the Kaggle competition page for which can be found at https://www.kaggle.com/c/predict-toxic-molecule/discussion. Our goal is to predict whether or not some molecules in with the given features are toxic (target: toxic = 1, non-toxic = 0). The toxicity is with respect to humans.

3 Data Set

The complete data set consists of **7464** training instances. The features are as listed at: https://www.kaggle.com/c/predict-toxic-molecule/data

4 Feature Elimination, Selection and Encoding

4.1 Removing uninformative features

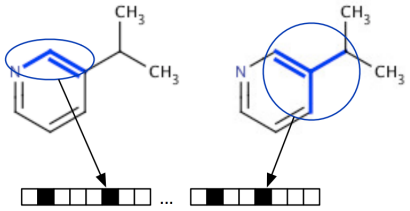
We eliminate the per instance identifiers **Index**(was used to index the prediction submission file on Kaggle) and **Inchi-key** which provide us with no information towards our classification task and merely serve as indices. On further analysis of the domain knowledge of toxicity prediction, it was found that the structural information essential to detecting toxic substructures in a compound can be found from the SMILES feature. Hence, the **Graphs** feature was redundant and was not used for training.

4.2 Understanding the SMILES feature

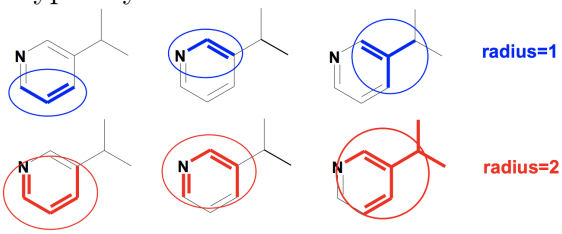
SMILES stands for **Simplified Molecular-Input Line-Entry System**. SMILES is a concise line notation for encoding molecular structures. The toxicity of a molecule is largely determined by the presence of specific substructures within the molecules structure. Our aim is to classify all the compounds which have a toxicity-lending substructure as toxic(1) and the ones that do not have such substructures as not toxic(0). Before, we do so however, we need a method to encode the SMILES of a compound into a representation that allows them to be used by a model.

4.3 Feature Encoding : Using Morgan Fingerprints

The idea is to generate a bit vector from the SMILES of a molecule, this bit vector is called a fingerprint. Each fingerprint bit basically corresponds to the presence of a particular fragment in the molecule. Hence, for molecules that have a lot of common fragments, the corresponding fingerprints will have considerable overlap.



We employ the **Morgan fingerprint** for this problem which is a similarity fingerprint that is contained in **rdkit.Chem.Draw.SimilarityMaps**. The fingerprint takes into a account the neighborhood of each atom typically within a radii of 0-3 bonds as shown below:



We now obtained a Morgan fingerprint that is a 2048 bit-wide bit-vector and in effect can be considered as 2048 features per training input. Based on the domain knowledge on toxicity alerts determined from the molecule structure, the experimentation phase of this project leaned towards the use of these new features we built as the only features necessary for the classification. However, experiments were performed with two subsets of features as described in the next section.

4.4 Feature Subsets

We consider the following feature subsets:

1. 2048 features that resulted from the bit-vector obtained from the Morgan fingerprint of the SMILES feature.(**2048 features in count**)

2. The 2048 features from the fingerprint as above augmented with the normalized values for the following 7 features: Maximum Degree, Minimum Degree, Molecular Weight, Number of H-Bond Donors, Number of Rings, Number of Rotatables and Polar Surface Area.(**2055 features in count**).

   On training with both these subsets, we found out that subset(1) resulted in higher average AUC scores over the 10-folds of our k-fold cross-validation procedure. (Note : This step was performed after parameter tuning).

5 Classifiers

5.1 Description of Inputs and Outputs

The input provided to each classifier remains the same and it is a matrix of dimension N(number of instances) x 2048 features/columns. Where, the value of N depends on whether we are training the model, testing the model or cross-validating the model.

The output produced by the model is the probability that a particular instance belongs to the positive class(toxic = 1). For a given input, the model with return these predicted probabilities for each instance of the input matrix.

	Training	Testing	Cross-validation
Input size	(7464 x 2048)	(1867 x 2048)	(split_fraction)*7464 x 2048)
Output size	(7464 x 1)	(1867 x 1)	(split_fraction)*7464 x 1)

Table 1: Dimensions of inputs and outputs corresponding to training, testing and cross-validation

5.2 Hyper-parameters, Score Functions and Search Algorithms for Classifiers

The following models were employed to solve the classification task of this problem statement,(the hyper-parameters mentioned are the ones that were tuned:

1. **Logistic Regression** (sklearn.linear_model.LogisticRegression)
   (a) Hyper-parameters : **C** (Inverse of regularization strength)
   (b) Score Function: **Negative log-likelihood**, this is referred to as the cross-entropy loss or logistic loss. For a training instance with true label y_t in (0,1) and estimated probability y_p that $y_t = 1$,
   The negative log-likelihood is: $-logP(y_t|y_p) = -(y_t log(y_p) + (1 - y_t) log(1 - y_p))$
   Where, $y_p = \sigma(w^T.x+b)$ (σ refers to the sigmoid function, w is the vector of parameters and b is the bias).
   (c) Search algorithm is the **Gradient Descent algorithm**.
2. **Random Forest Classifier**(sklearn.ensemble.RandomForestClassifier)
   (a) Hyper-parameters:
   max_depth: The maximum depth of a tree,
   max_features: The number of features to consider when looking for the best split,
   n_estimators: The number of trees in the forest
   oob_score: (Boolean) Whether to use out-of-bag samples to estimate the generalization accuracy. We set this parameter as true.
   (b) Score Function and Search Algorithm:
   A random forest uses random feature subset selection along with bagging to fit a number of decision tree classifiers on various sub-samples of the dataset. It uses averaging to improve the predictive accuracy and control over-fitting.
   Decision trees that are a part of the ensemble are independently trained. At every node, we select the best feature for splitting which is a greedy algorithm. **GINI GAIN** was used as the split-criterion which is defined as follows:
   $Gini(x) = 1 - \sum_x p(x)^2$
   $Gain(S, A) = Gini(S) - \sum_A (|A|/|S|) Gini(A)$
3. **Gradient Boosted Decision**
   (a) Hyper-parameters : **eta**(analogous to learning rate) , **max_depth**

(b) Score Function
Our model is described by the following equation,
$\hat{y} = \sum_{k=1}^{K} f_k(x_i), f_k \subset F$
Where K is the number of trees, F is a function in the functional space , and F is the set of all possible trees.
where the score function is : $\frac{1}{2n} \sum_i ||y_i - \hat{y}||^2$

(c) Search Algorithm:
$\hat{F} = argmin_w(\sum_i [L(y_i, \hat{y_i}]) + \sum_{k=1}^{K} \Omega(f_k)$
Where, $\Omega(f_k) = \gamma T + \frac{\lambda}{2} \sum_{k=1}^{K} w_k^2$
Here,w is the vector of scores on leaves, q is a function assigning each data point to the corresponding leaf, and T is the number of leaves.

4. **Neural Network** (using PyTorch)

(a) Hyper-parameters : **epochs** (Number of training iterations of forward and backward passes),**learning_rate**

(b) Score function : Negative Log-Likelihood Loss **torch.nn.NLLLoss()**
L(y) = $-\log(y_{pred} - y_{true})$,
We minimize this value.

(c) Search algorithm
$\hat{F}$ = argmin on weights and biases of all layers ($L(y)$) using gradient descent.

(d) Forward pass equations:
$z_1 = W_1^T X + b_1$, (Hidden layer 1 inputs)
$h_1 = LeakyReLU(z_1)$

$z_2 = W_2^T h_1 + b_2$, (Hidden layer 2 inputs)
$h_2 = LeakyReLU(z_2)$

$z_3 = W_3^T h_2 + b_3$ (Output layer inputs)
$\hat{y} = Softmax(z_3)$ (Output)

6 Hyper-parameter Tuning

6.1 Procedure

We use **discrete-optimization-within-validation** approach for hyper-parameter tuning. The procedure can be described as follows:

1. We perform k-fold cross-validation and within each fold we have a training split(k-1 parts) and a testing split(1 part).

2. We further divide the training split of the fold into two parts. (80-20 split).

   (a) We now go through all the possible parameter combinations for each model.

   (b) We then train on the 80% split with each parameter combination and test on the 20% split to identify the best parameters for the current fold for the current model. This is our discrete optimization step.

With the best parameter values chosen for the fold we train the model on the training split(k-1 parts) for the fold and test on the testing split(1 part).

We then choose the best parameters across all folds as parameters on which we get the highest AUC score on the test split of a fold.

6.2 Accommodating for class imbalance while tuning

The training data has strong class imbalance with **304** positive training instances and **7160** negative training instances. To accomodate for the same when training we use stratification of the folds in our k-fold crossvalidation splits when performing discrete-optimization-within-validation using **sklearn.model_selection.StratifiedKFold** The stratification ensures the representation of the minority class in each split as the ratio of the positive instances to the negative instances is approximately the same in the training data.

6.3 Hyper-parameter grids for each classifier

We tune our classifiers on the following hyper-parameters and a list of values for each hyper-parameter:

1. **Logistic Regression**
   C = logspace(-20, 20, 100)

2. **Random Forest**
   "n_estimators" : [50, 100, 150, 200]
   "min_samples_leaf" : [1, 5, 10]
   "max_depth" : [2, 3, 4, 10, 20, 30, 40, 45]

3. **Gradient Boosted Decision Tree**
   "eta": [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7]
   "max_depth": [2, 3, 4, 5, 6, 7]

4. **Neural Network**
   "epochs" = [100, 150, 200]
   "learning_rate" = [0.001, 0.01, 0.05, 0.1]

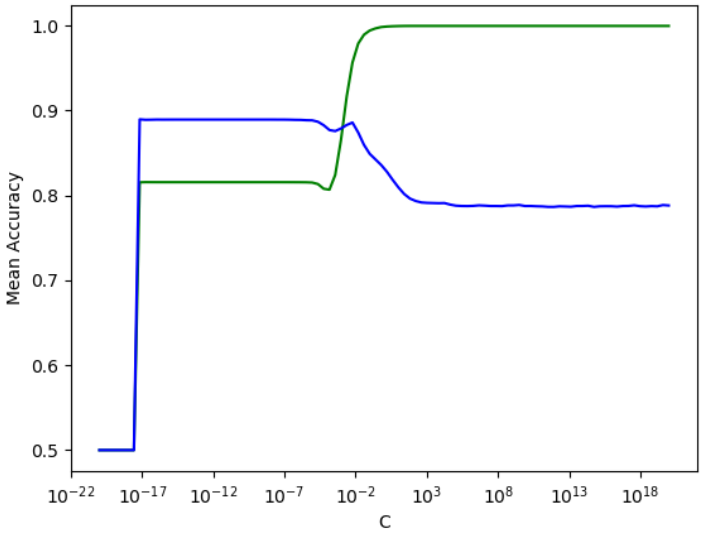
6.4 Fold-wise results of the hyper-parameter tuning procedure for Logistic Regression, Random Forest Classifier and Gradient Boosted Decision Tree

```
Parameter tuning fold 1
Model LR {'C': 0.01519911082952927}  AUC : 0.8004144
Model RF {'n_estimators': 100, 'min_samples_leaf': 1, 'max_depth': 10}  AUC :
↪  0.8416381
Model XG {'eta': 0.1, 'max_depth': 2}  AUC : 0.8396783

Parameter tuning fold 2
Model LR {'C': 0.01519911082952927}  AUC : 0.8588259
Model RF {'n_estimators': 150, 'min_samples_leaf': 1, 'max_depth': 40}  AUC :
↪  0.8859479
Model XG {'eta': 0.1, 'max_depth': 2}  AUC : 0.8854072

Parameter tuning fold 3
Model LR {'C': 0.01519911082952927}  AUC : 0.8625428
```

```
Model RF {'n_estimators': 50, 'min_samples_leaf': 5, 'max_depth': 30}  AUC :
↪  0.9018291
Model XG {'eta': 0.1, 'max_depth': 2}  AUC : 0.8567309

Parameter tuning fold 4
Model LR {'C': 0.005994842503189421}  AUC : 0.8676788
Model RF {'n_estimators': 50, 'min_samples_leaf': 5, 'max_depth': 30}  AUC :
↪  0.84920706
Model XG {'eta': 0.1, 'max_depth': 2}  AUC : 0.9130248

Parameter tuning fold 5
Model LR {'C': 0.005994842503189421}  AUC : 0.5
Model RF {'n_estimators': 50, 'min_samples_leaf': 5, 'max_depth': 30}  AUC :
↪  0.8408519
Model XG {'eta': 0.1, 'max_depth': 2}  AUC : 0.8118947

Parameter tuning fold 6
Model LR {'C': 0.005994842503189421}  AUC : 0.5
Model RF {'n_estimators': 150, 'min_samples_leaf': 1, 'max_depth': 10}  AUC :
↪  0.9141061
Model XG {'eta': 0.1, 'max_depth': 2}  AUC : 0.9317039

Parameter tuning fold 7
Model LR {'C': 0.005994842503189421}  AUC : 0.8108705
Model RF {'n_estimators': 150, 'min_samples_leaf': 1, 'max_depth': 10}  AUC :
↪  0.8109869
Model XG {'eta': 0.1, 'max_depth': 2}  AUC : 0.8164338

Parameter tuning fold 8
Model LR {'C': 0.005994842503189421}  AUC : 0.7847998
Model RF {'n_estimators': 150, 'min_samples_leaf': 1, 'max_depth': 10}  AUC :
↪  0.7428072
Model XG {'eta': 0.1, 'max_depth': 2}  AUC : 0.7996042

Parameter tuning fold 9
Model LR {'C': 0.005994842503189421}  AUC : 0.7857774
Model RF {'n_estimators': 150, 'min_samples_leaf': 1, 'max_depth': 10}  AUC :
↪  0.8294227
Model XG {'eta': 0.1, 'max_depth': 2}  AUC : 0.7969040

Parameter tuning fold 10
Model LR {'C': 0.005994842503189421}  AUC : 0.8603817
Model RF {'n_estimators': 150, 'min_samples_leaf': 1, 'max_depth': 10}  AUC :
↪  0.8239292
Model XG {'eta': 0.1, 'max_depth': 2}  AUC : 0.84399441

Best Accuracy over all folds {'LR': 0.8676788, 'RF': 0.9141061, 'XG': 0.9317039}
Best parameters overall
{
 'LR': {'C': 0.005994842503189421},
 'RF': {'n_estimators': 150, 'min_samples_leaf': 1, 'max_depth': 10},
```

```
 'XG': {'eta': 0.1, 'max_depth': 2}
 }
```

Below is the curve for mean accuracy on training(green curve) and validation splits(blue curve) as we tune the hyper-parameter 'C' for our Logistic Regression model. We can see that we get the best value of an AUC of 0.8676788 on the validation split at around 'C' = 0.00599484250318942.



The final hyper-parameters chosen for each model after parameter tuning are as follows:

```
Best parameters chosen for each model after the parameter tuning procedure are
 ↪  as follows:
{
 'LR': {'C': 0.005994842503189421},
 'RF': {'n_estimators': 150, 'min_samples_leaf': 1, 'max_depth': 10},
 'XG': {'eta': 0.1, 'max_depth': 2},
 'NN': {'epochs': 100 , 'learning_rate': 0.001}
}
```

7 Evaluating classifier performance

7.1 Learning Curves

Figure 1: Logistic Regression

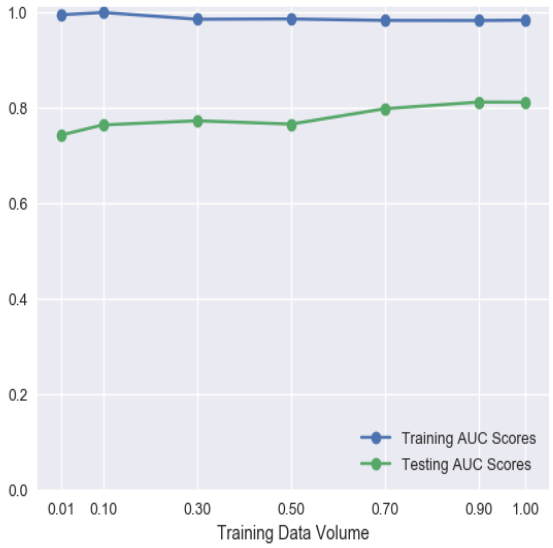


Figure 2: Random Forest Classifier

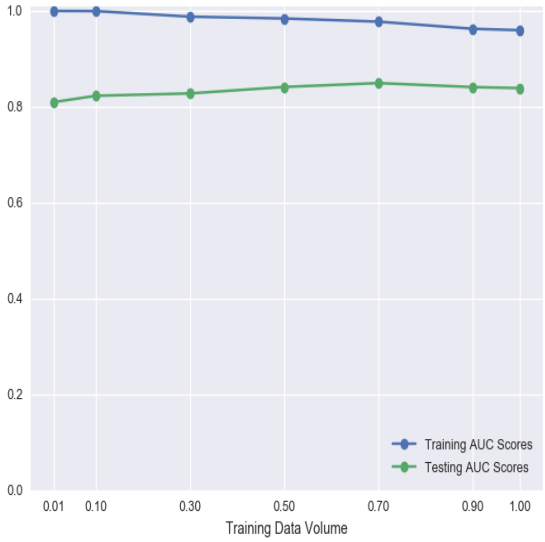


Figure 3: Gradient Boosted Decision Tree

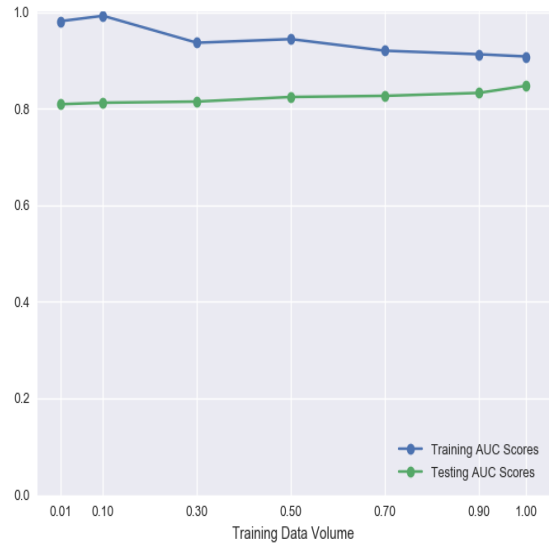
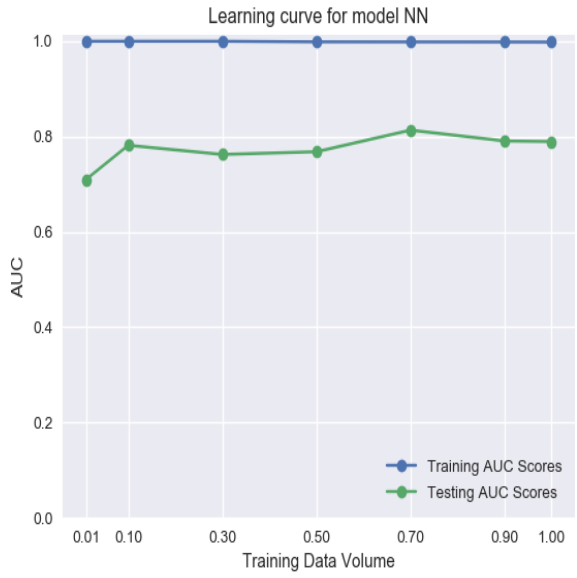


Figure 4: Neural Network



7.2 Receiver Operating Characteristic curves

The area under the ROC gives the of accuracy for the classifier.

Figure 5: Logistic Regression

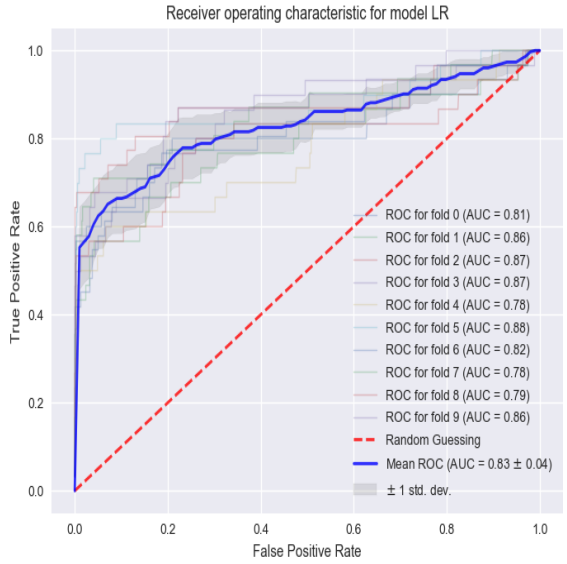


Figure 6: Random Forest Classifier

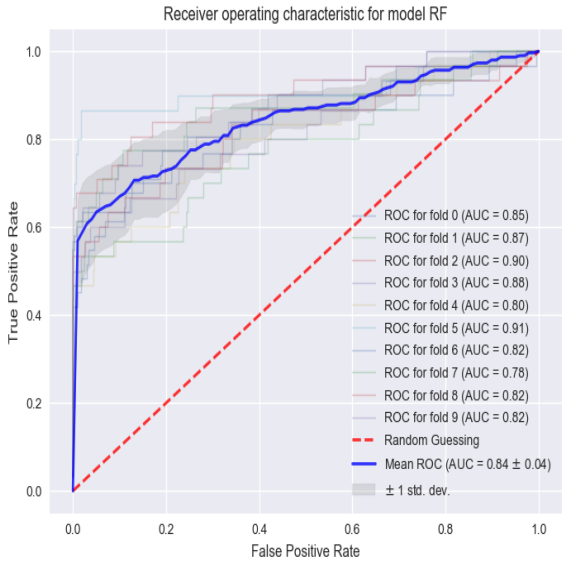


Figure 7: Gradient Boosted Decision Tree

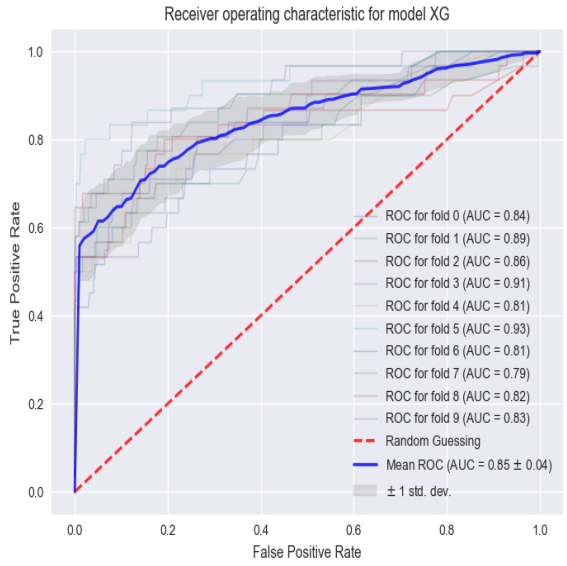
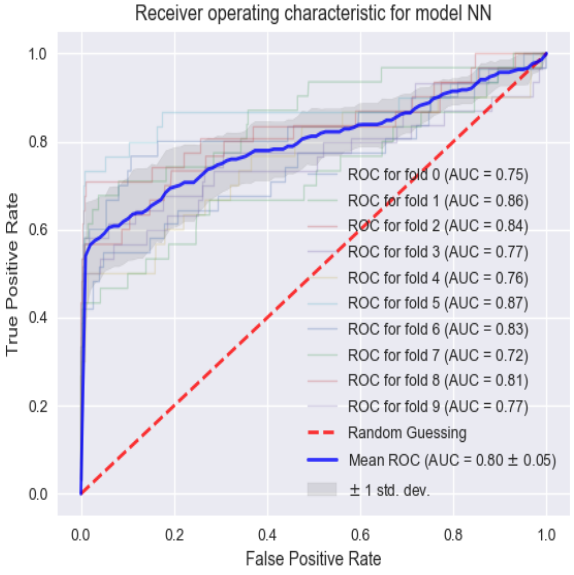


Figure 8: Neural Network



8 Hypothesis Testing : Multiple Hypotheses

We run k-fold cross validation on each of the four candidate models and then obtain a list of AUC scores for each fold for each candidate model.
From the values of mean and standard deviation for the k-folds for each model, we can see that the mean AUC for the Gradient Boosted Decision Tree is slightly higher than the other models hence we perform Hypothesis using paired t-tests to assess whether we can say with high confidence that is better than the rest of the models.

8.1 K-fold Cross-validation Statistics for model Logistic Regression(LR)

Mean AUC : 0.8348130293746621
Std. Dev of AUC: 0.03869584259574496

8.2 K-fold Cross-validation Statistics for model Random Forest(RF)

Mean AUC : 0.8377852616086983
Std. Dev of AUC: 0.05102634285521786

8.3 K-fold Cross-validation Statistics for model Gradient Boosted Decision Trees(XG)

Mean AUC : 0.8488044392383014
Std. Dev of AUC: 0.04443980112848005

8.4 K-fold Cross-validation Statistics for model Neural Network(NN)

Mean AUC : 0.8086232414444555
Std. Dev of AUC: 0.06738034341430565

8.5 Formalizing the multiple hypothesis tests

As from the above statistics we observe that the mean AUC for Gradient Boosted Decision Trees is higher than the rest of the models we formalize the following three hypotheses : We use **scipy.stats.ttest_rel** to get the p-values and T-statistic for each hypothesis test as follows:.
Hypothesis 1:
H_0: $\mu_{AUC-XG} - \mu_{AUC-LR} = 0$
H_1: $\mu_{AUC-XG} > \mu_{AUC-LR}$
T-statistic=1.7844964883141075, p-value=0.10800638272215714

Hypothesis 2:
H_0: $\mu_{AUC-XG} - \mu_{AUC-RF} = 0$
H_1: $\mu_{AUC-XG} > \mu_{AUC-RF}$
T-statistic=1.0671103786065952, p-value=0.31370891449928795

Hypothesis 3:
H_0: $\mu_{AUC-XG} - \mu_{AUC-NN} = 0$
H_1: $\mu_{AUC-XG} > \mu_{AUC-NN}$
T-statistic=4.0864233596523, p-value=0.0045224342313212

As we are performing multiple hypotheses test, we apply the Bonferroni correction. Hence our confidence limit becomes 0.05/3= 0.0167
Considering the above confidence limit, we can say that we cannot reject the null hypothesis in tests 1 and 2. However we can reject it for test 3.
Hence, we cannot say that Gradient Boosted Decision Trees performed better than Logistic Regression or the Random Forest Classifier , however we can say so for for the Neural Network with high confidence(95%).

9 Conclusion

As we saw that the performance for Gradient Boosted Decision Trees and the Random Forest Classifier were comparable, both these models were used with the best parameters obtained from

parameter tuning to run on the test data provided to make the two predictions to be submitted for the Kaggle contest.

10 References

1. http://www.rdkit.org/UGM/2012/Landrum_RDKit_UGM.Fingerprints.Final.pptx.pdf
2. https://www.frontiersin.org/articles/10.3389/fenvs.2015.00080/full
3. http://home.deib.polimi.it/gini/papers/CIDM.pdf