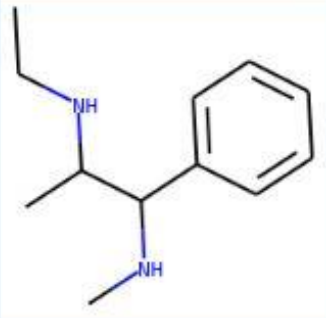

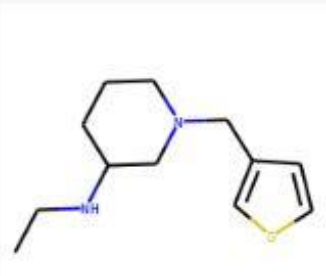


0	<chem>CCNC(C)C(NC)c1ccccc1</chem>	
1	<chem>CONC(=O)c1cnccc1</chem>	
2	<chem>CCNC1CCCN(Cc2ccsc2)C1</chem>	

COVID - Drug Discovery for COVID19(Prediction of Binding Affinity)

AI CROWD CHALLENGE

RISHAB (2019201050)|SMAI ASSIGNMENT | APRIL 30TH, 2020

ABOUT THE CHALLENGE

- The objective is to prepare a machine learning model that can be used to propose potential novel effective drugs to fight SARS-CoV-2, the virus responsible for COVID-19.
- You are provided with a dataset containing drug molecules (encoded as SMILES) and their binding affinities. The task is to use this dataset to make a regression model for binding affinity prediction.

ABOUT THE DATASET

SARS-CoV-2 virus contains proteins responsible for action and replication of the virus. The protein functions can be stopped by introducing drug molecules that are capable of blocking the protein. In other words, preparation of a drug involves finding molecules that can effectively bind to the protein i.e have a high binding affinity.

In this task, you are provided with a dataset of drug molecules and their binding affinity towards SARS-CoronaVirus Main Protease(Mpro), one of the proteins in the target virus.

The data has been generated using Protein-Ligand docking.

SMILES are character strings to represent drug molecules. For example, a carbon atom can be represented as “C”, an oxygen atom can be represented as “O”, double bond by “=”. The molecule Carbon dioxide is represented as “C(=O)=O”. Read more about SMILES [here](#)

The max length of the string is 25.

ANALYSIS

Lets see how the training data looks like.

	SMILES sequence	Binding Affinity
0	<chem>CCNC(C)C(NC)c1ccccc1</chem>	-18.0861
1	<chem>CONC(=O)c1cncnc1</chem>	-17.5783
2	<chem>CCNC1CCCN(Cc2ccsc2)C1</chem>	-20.3645
3	<chem>CC(NC(=O)CSCCN)c1ccccc1</chem>	-19.3144
4	<chem>CCC(CS)CN(C)c1ccccc1</chem>	-15.8451
...
8995	<chem>CC(O)CCNC(=O)CNCc1ccccc1</chem>	-20.3461
8996	<chem>N#Cc1cccc(-c2cnsc2)n1</chem>	-23.2916
8997	<chem>Cc1ccnc(NC(CN)C(C)C)c1</chem>	-22.1677
8998	<chem>CC(=O)CCc1cncnc1</chem>	-16.6403
8999	<chem>COCCC(=O)Nc1cnccc1C</chem>	-16.6874

9000 rows × 2 columns

Now we can clearly see that there are currently no numerical features in the given data ,we need to somehow convert it into numerical features.

Approach 1 (Naïve)

String processing : Since the SMILES sequence is nothing but a string representation of actual molecules we can try to extract the counts of each type of Atom and number of bonds or number and order of digits, and other types of symbols present, store their count for each type of molecule and then apply regression.

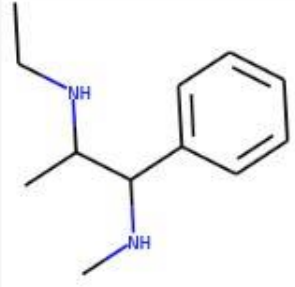


This was a very Naïve approach as it did not actually use any of the molecular features which are hidden in these representations.

As expected there was hardly any correlation between the binding affinity and the string features which we extracted which didn't have any molecular speciality. So no regression model performed good on this problem. So now we needed to extract some molecular features from the SMILES representation.

Approach 2:

Use of RDKit : Upon looking from various resources ,I decided to use RDKit to convert the SMILES representation to a molecular representation by using the 'AddMoleculeColumnToFrame' function of PandasTools and 'RDKit.Chem.MolFromSmiles' functions.

Lets see how the molecular representations of the corresponding SMILES representation look like.

	SMILES sequence	mol
0	<chem>CCNC(C)C(NC)c1ccccc1</chem>	
1	<chem>CONC(=O)c1cnnc1</chem>	
2	<chem>CCNC1CCCN(Cc2ccsc2)C1</chem>	

Now we can use functions present in RDKit for the prediction task.

The functions are basically calls to methods which extract the molecular properties of the molecule like the degree of accepting nature of various Atoms and Ions. Types of Bonds and various others.

The functions to extract the properties which I tried were:

1. Descriptors.ExactMolWt(mol)
2. Chem.Lipinski.FractionCSP₃(mol)
3. Chem.Lipinski.NOCOUNT(mol)
4. Chem.Lipinski.NHOHCount(mol)
5. rdMolDescriptors.CalcNumHBA(mol)
6. rdMolDescriptors.CalcNumHBD(mol)
7. rdMolDescriptors.CalcTPSA(mol)
8. Descriptors.HeavyAtomMolWt(mol)
9. Descriptors.NumValenceElectrons(mol)
10. rdMolDescriptors.CalcNumHeteroatoms(mol)
11. rdMolDescriptors.CalcNumAromaticRings(mol)
12. Chem.Crippen.MolLogP(mol)
13. Descriptors.MaxAbsPartialCharge(mol)
14. Descriptors.MinAbsPartialCharge(mol)
15. Descriptors.MaxPartialCharge(mol)

After extraction all these features, the data looked like as follows.

exact_mw	fraction_csp3	no_count	nhoh_count	hba	hbd	tpsa	heavymwt	numvalence	n_hetero_atoms	n_aromaitc_rings	mol_logP	max_abs_partial
192.162649	0.500000	2	2	2	2	24.06	172.146	78	2	1	1.9451	(
153.053826	0.166667	5	1	4	1	64.11	146.085	58	5	1	-0.2322	(
224.134720	0.666667	2	1	3	1	15.27	204.213	84	3	1	2.3220	(
238.113984	0.416667	3	3	3	2	55.12	220.212	88	4	1	1.5557	(
209.123821	0.500000	1	0	2	1	3.24	180.206	78	2	1	3.0788	(

After this I tried to use various regression models like Linear Regression, Ridge CV (because it has inbuilt cross validation for best parameters) , Ensemble Regressors like XGB Regressor , Random Forest Regressor and Support Vector Regressor. For RDkit transformations RidgeCV seemed to give the best performance on the validation data with a root mean squared error of 2.75 and mean absolute error of 1.95.

Still I thought I could do something more than these approaches.

Approach 3: Mol2Vec

Mol2Vec is an unsupervised machine learning approach to learn vector representations of molecular substructures, it is inspired by NLP techniques (Word2Vec).

Mol2vec learns vector representations of molecular substructures that point in similar directions for chemically related substructures. Compounds can finally be encoded as vectors by summing the vectors of the individual substructures and, for instance, be fed into supervised machine learning approaches to predict compound properties. The underlying substructure vector embeddings are obtained by training an unsupervised machine learning approach on a so-called corpus of compounds that consists of all available chemical matter. The resulting Mol2vec model is pretrained once, yields dense vector representations, and overcomes drawbacks of common compound feature representations such as sparseness and bit collisions.

So I used a pre-trained Mol2Vec embedding to transform the Molecular Representation obtained from the SMILES sequence.

It required 3 steps:

1. Mol2AltSentence
2. Mol2Sentence on each Mol2AltSentence
3. Sentence2Vec.

Since the pretrained model to extract the embeddings I used was a 300 dim Mol2Vec model, the data was transformed to a 300 dimensional dataset.

After conversion to sentences:

```
1 train_files['sentence']
0      (2246728737, 3542456614, 2245384272, 773607102...
1      (2246728737, 3975275337, 864674487, 903112553,...
2      (2246728737, 3542456614, 2245384272, 773607102...
3      (2246728737, 3537119515, 2245273601, 146777865...
4      (2246728737, 3542456614, 2245384272, 150656359...
...
8995   (2246728737, 3537119515, 2245273601, 161474856...
8996   (847433064, 2551483158, 2245900962, 551287920,...
8997   (2246728737, 422715066, 3217380708, 3207567135...
8998   (2246728737, 3545365497, 2246699815, 208074702...
8999   (2246728737, 3975275337, 864674487, 2222621677...
Name: sentence, Length: 9000, dtype: object
```

Now these are numerical features

Further converting these to Vectors:

The data looked like this.

Each row is a numerical 300 dimensional representation of the molecules we obtained from SMILES using RDKit.

```
x_train
```

```
array([[ -0.35749012, -1.2064799 ,  0.11197358, ..., -3.540155 ,
        -2.9649398 , -1.7696643 ],
       [  1.3296323 , -1.7659448 , -2.3831866 , ..., -3.1224864 ,
        -4.79916   , -1.5164237 ],
       [  1.0471523 , -3.901737 ,  0.7891791 , ..., -3.2860694 ,
        -6.1752906 , -0.8618055 ],
       ...,
       [  0.45528913, -0.767791 , -1.8602836 , ..., -3.6912436 ,
        -6.570815 , -0.97852397],
       [  0.86796445, -1.3686302 , -2.3809779 , ..., -2.3511865 ,
        -5.7286477 , -1.6859957 ],
       [-0.06253776, -2.0537918 , -1.566068 , ..., -4.3287835 ,
        -6.14024   , -2.9561985 ]], dtype=float32)
```

Now these are pure numerical feature vectors which we need which the model can learn and adjust the regression line.

RESULTS

After obtaining data from Mol2Vec conversion and after training and testing by splitting 9000 training points to 7200: 1800 train-test ratio, we saw the following results on different models.

- 1 Gradient boosting Regressor (250 estimators) – RMSE 2.55, MAE 1.81
2. KNN Regressor (K=11) – RMSE 7.43 , MAE 2.007
3. Random Forest Regressor (250 estimators) - RMSE 2.62 , MAE 1.909
4. Neural Network(SKlearn MLP regressor with 2 hidden layers) – MSE 3.48 , MAE 2.54
5. RidgeCV with best cross validated alpha = 100 - RMSE 2.35 , MAE 1.70

6. Support Vector Regressor with rbf kernel ,C=80 and epsilon = 1.6 obtained after GridSearch CV.- RMSE 2.187 , MAE 1.614.

Clearly the winner was Support Vector Regressor.

On testing it on the final test data,it performed very well.

The result obtained on test set was RMSE 2.229 and MAE 1.654 which is a pretty good performance.

ABOUT THE CODE:

PYTHON VERSION USED 3.7

TRAIN FILE AND TEST FILE PATH ARE HARDCODED AS

'train.csv' and 'final_test_file.csv'

THE CODE ALSO USES A PRETRAINED EMBEDDING MODEL FOR MOL₂VEC CONVERSION AS A PKL FILE WHICH IS CALLED AS 'model_300dim.pkl' DOWNLOADED FROM THE CREATORS WEBSITE.

THE CODE IS DIRECTLY DOWNLOADED AS PY FROM A IPYNB SO IT IS NOT BE USED DIRECTLY,USE THE FUNCTIONS REQUIRED BY TAKING THE SNIPPETS FROM THE PY FILE.