Towards Interpretable Machine Learning Models for Materials Discovery[1]

Shreecharran Sundar (CH17B075)
Dept of Chemical engineering, IIT Madras

1. MOTIVATION

Quantitative structure activity relationship (QSAR) modeling is now more than 50 years old, and its utility has been shown in numerous research publications and by the number of new drug and agrochemical entities that have been developed with its aid. The method has evolved very substantially since the seminal linear regression QSAR models. Many new types of molecular descriptors have been developed, new mathematical methods such as neural networks, support vector machines, kernel regression, and random forest have been applied to mapping structure to activity, and QSAR has now incorporated 3D structures ,conformation, and chirality. Two main branches of QSAR have evolved. The first of these, where the model is often relatively simple and linear and interpretable in terms of molecular interactions or biological mechanisms. The second type focuses much more on modeling structure−activity relationships in large data sets with high chemical diversity using a variety of regression or classification methods, and often the interpretation of the model is obscure or impossible. A recent publication [2] provides detailed discussion on this topic. Data-driven ML models are considerably more useful if more chemically interpretable descriptors are used to train them, as long as these models also accurately recapitulate the properties of materials. Current modelling methods use arcane mathematical descriptors which meant that model predictive power and interpretability remained essentially orthogonal. This paper introduces a particular type of molecular fragment descriptor, the signature descriptor which achieves these joint aims of accuracy and interpretability.

2. INTRODUCTION

2.1 Quantitative structure property relationship (QSPR/QSAR) modeling:

QSPR/QSAR model represent mathematical relationship between the response of chemical compound(activity/property) with their structural and physiochemical information. It works on the principle that structurally similar compounds have similar properties. A QSPR has the form of a mathematical model :

Property = $f(D_1, D_2, \ldots D_4)$ + error

$D_1, D_2, \ldots D_4$ – chemical or structural property (Molecular Descriptors)

A **QSPR model** is composed of dependent and independent variables. In the **QSPR model**, the dependent variable is estimated from the independent variables. Independent variables are **molecular descriptors** computed from the chemical structure of compounds, and the dependent variable is the property under study.

---

[1] Paulius Mikulskis, Morgan R. Alexander, and David Alan Winkler, 'Toward Interpretable Machine Learning Models for Materials Discovery', *Advanced Intelligent Systems*, 1.8 (2019) <https://doi.org/10.1002/aisy.201900045>.

[2] Toshio Fujita and David A. Winkler, 'Understanding the Roles of the "Two QSARs"', *Journal of Chemical Information and Modeling*, 56.2 (2016), 269–74 <https://doi.org/10.1021/acs.jcim.5b00229>.

2.2 Molecular Descriptors

Molecular descriptors are the result of a logical and mathematical procedure which transforms chemical information encoded within a symbolic representation of a molecule into a number. Molecular descriptors are divided into two main categories: **experimental measurements**, such as log P, molar refractivity, dipole moment, polarizability, and **theoretical molecular descriptors**, which are derived from a symbolic representation of the molecule.

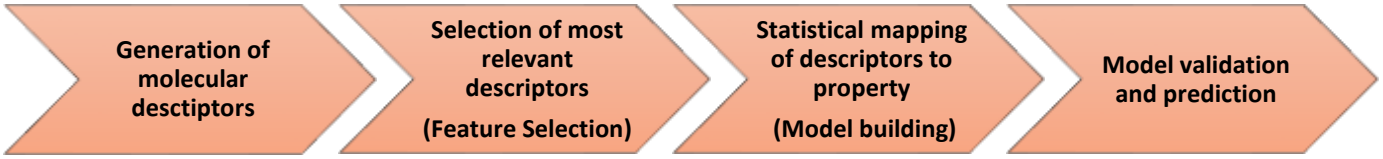
The main classes of theoretical molecular descriptors are: 1) **0D-descriptors** (i.e. constitutional descriptors, count descriptors), 2) **1D-descriptors** (i.e. list of structural fragments, fingerprints), 3) **2D-descriptors** (invariants of molecular graphs, e.g., connectivity indices, information indices, counts of paths and walk),4) **3D-descriptors** (based on spatial properties of the molecules)

Types of Descriptor		
Descriptor type	**Description**	**Examples**
Constitutional Descriptors	They represent properties related to molecular structure	molecular weight, total number of atoms in the molecule, number of aromatic rings
Electrostatic	They represent properties related to the electronic nature of the compound	atomic net partial and charges
Topological Descriptors	They represent properties which can be inferred by treating the structure of the compound as a graph, with atoms as vertices and covalent bonds as edges	total number of bonds in shortest paths between all pairs of non-hydrogen atoms
Geometrical Descriptors	They represent properties related to spatial arrangement of atoms constituting the compound	Vander Waals Area
Fragment based Descriptors	They represent properties related to substructural motifs	MDL Keys and Molecular Fingerprints

Tools to calculate Molecular descriptors		
Name	**Type**	**Availability**
Chem Development Kit	Continuous	Free. https://cdk.github.io/
Padel	Continuous /Fingerprint	Free. http://www.yapcwsoft.com/dd/padeldescriptor
RDKit	Continuous /Fingerprint	Free. http://www.rdkit.org
Dragon	Continuous	Commercial. http://www.talete.mi.it/products/dragon_description.htm

2.3 QSPR Workflow

1. calculation of molecular descriptors (mathematical representations of molecular properties)
2. descriptor selection (i.e. choosing the subset of descriptors most relevant to the problem)
3. finding an optimum (usually nonlinear) relationship between the descriptors and response variable
4. validation of the model (i.e. how predictive/ robust is it, and domain of applicability)



Feature Selection
Importance of feature selection : • Improves Interpretation • Less features, simpler models. • Expert-driven feature selection enhances the mechanistic interpretation of the models. • Reduces Overfitting • Less redundant data means lesser decisions based on noise. • Reduces Training Time • Less data to learn from ensures quicker model development. **Common feature selection methods:** • Filter: e.g.: **Chi-squared test, T-test, Correlation coefficient** • Wrapper: e.g.: **Genetic search, Recursive feature elimination** • Embedded: e.g. : **Decision trees ,Support vector machines, Lasso regularization**
Model development
• Multiple linear regression, k-Nearest Neighbors (k-NN), linear discriminant analysis, decision trees, random forests, support vector machines, artificial neural networks and other ensemble methods are some of the most used methods for QSPR modelling
Model validation
• **Internal validation** **• K-fold cross validation:** The dataset is split into K parts. K models are developed using (K-1) sets and the Kth set is used as the test set. **• Leave one out cross-validation:** N models are developed each with (N − 1) chemicals as training set and 1 chemical as the test set. • **External test set validation** **•Regression metrics:** Root-mean-squared-error, Mean Average Error, Coefficient of Determination **•Classification metrics:** Accuracy, Sensitivity, Positive Predictivity, Recall, Precision, F1- score

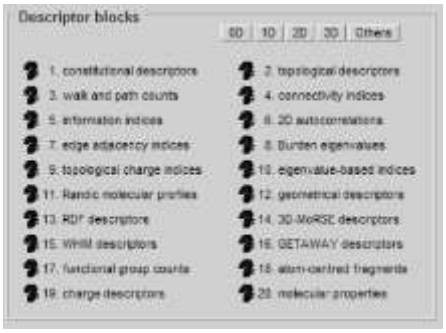
2.4 Signature and Dragon Descriptors

(1) Dragon : Dragon is an application for the calculation of molecular **descriptors**. The Dragon program can calculate 5000 molecular descriptors, some of which are easy to interpret, but the **majority are quite arcane and almost impossible to understand in terms of chemical structure**.

DRAGON was designed as a user-friendly software which performs descriptor calculation using a simple logical sequence (Figure 1) :

1. loading of the molecular files
2. selection of the descriptors
3. calculation of the descriptors
4. saving of the calculated descriptors

Together with the molecular descriptors, DRAGON also allows the calculation of the Principal Components (PCs) of descriptors included in the selected block to compress into a small number of variables.



(2) Signature descriptors

The signature of an atom in a polymer is a representation of the atom's connected environment up to a predefined depth d. Signature descriptor is generated from the chemical graph of a molecule or material, and describes the number of atoms, atom types and hybridizations, bond types. Signature descriptors are more chemically interpretable than many other types of descriptors as they have been used successfully in a range of drug discovery applications due to their inherent **ability to be mapped back to chemical structures** to provide clear guidance to which functionalities in molecules contribute to, desired biological responses. A brief description of how signature descriptors are generated is described in figure 1. The algorithms for generating signature descriptors is included in these publications[3][4].

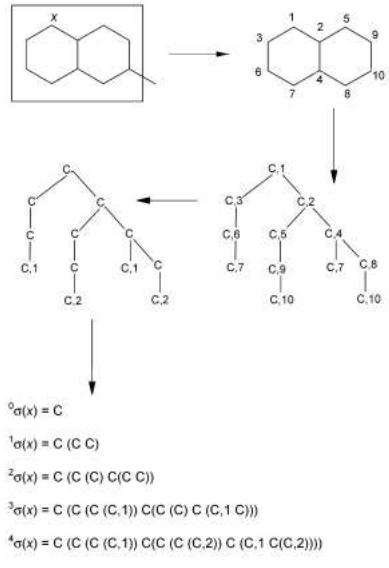


Figure 1. Atomic signature. The figure illustrates the five steps procedure for computing the atomic signature of atom x in methylnaphthene. (1) The subgraph containing all atoms at distance 4 from atom x is extracted. (2) The subgraph is canonized atom x having label 1. (3) A tree spanning all edges of the subgraph is constructed. (4) All labels appearing only once are removed, and the remaining labels are renumbered in the order they appear. (5) The signature is printed reading the tree in a depth-first order. Here we show atomic signatures from $h = 0$ to $h = 4$.

[3] Pablo Carbonell, Lars Carlsson, and Jean Loup Faulon, 'Stereo Signature Molecular Descriptor', *Journal of Chemical Information and Modeling*, 53.4 (2013), 887–97 <https://doi.org/10.1021/ci300584r>.

[4] Jean Loup Faulon, Michael J. Collins, and Robert D. Carr, 'The Signature Molecular Descriptor. 4. Canonizing Molecules Using Extended Valence Sequences', *Journal of Chemical Information and Computer Sciences*, 44.2 (2004), 427–36 <https://doi.org/10.1021/ci0341823>.

2.5 Machine Learning frameworks for model building and feature selection

2.5.1 Multiple linear regression with EM algorithm for feature selection (MLREM)

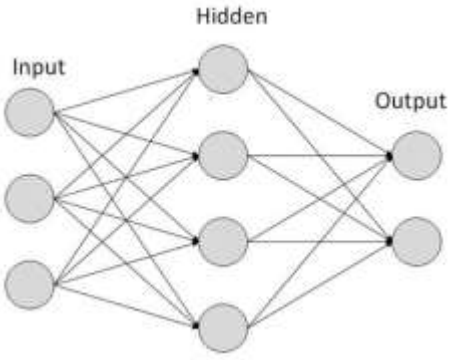
Feature selection represents the most critical step when deriving QSAR/QSPR models from a large parametric space. QSAR belongs to a family of mathematical problems, called grossly underdetermined systems, in which the number of independent variables greatly exceeds the number of dependent variables.

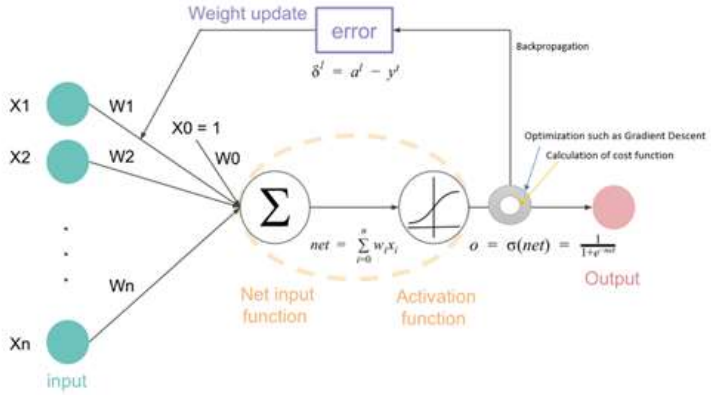
MLREM [5] is a Bayesian method that uses a sparse prior to eliminate uninformative or low-relevance descriptors by compressing them to zero. It is an L1 feature selection method like the least absolute shrinkage and selection operator method. Feature significance was encoded by sizes and magnitudes of the regression coefficients for each descriptor. Descriptors identified using the MLREM feature selection procedure were used to train nonlinear neural network regression models. In this project, I implemented (L1 feature selection method) Least absolute shrinkage and selection operator that selects the most relevant descriptors and shrinks other coefficients to zero. The chosen descriptors where then used to build the neural network model

2.5.2 Artificial Neural Networks (feed forward and backpropagation)

ANNs are artificial systems simulating the function of the human brain. Three components constitute a neural network: the processing elements or nodes, the topology of the connections between the nodes, and the learning rule by which new information is encoded in the network. While there are a number of different ANN models, the most frequently used type of ANN in QSAR is the three-layered feed-forward network In this type of networks, the neurons are arranged in layers (an input layer, one hidden layer and an output layer). Each neuron in any layer is fully connected with the neurons of a succeeding layer and no connections are between neurons belonging to the same layer.

According to the supervised learning adopted, the networks are taught by giving them examples of input patterns and the corresponding target outputs. Through an iterative process, the connection weights are modified until the network gives the desired results for the training set of data. A back-propagation algorithm is used to minimize the error function.





[5] F. R. Burden and Dave A. Winkler, 'Optimal Sparse Descriptor Selection for QSAR Using Bayesian Methods', *QSAR and Combinatorial Science*, 28.6–7 (2009), 645–53 <https://doi.org/10.1002/qsar.200810173>.

2.6 Feature significance of the models

Feature significance (parametric sensitivity) for ML models was evaluated by generating the partial derivative of the response variable **(a)** with respect to the descriptor at a position on the response surface close to the response variable maximum. This was achieved by inducing a small change of 0.01 to each normalized descriptor value **(d)** in turn, with the remaining descriptors kept fixed. The difference between original predicted and perturbed predicted property value obtained from neural network (NN) allowed the partial derivative to be calculated

$$partial\ derivate\ (delta) = \frac{\Delta a}{\Delta d} = \frac{change\ in\ output}{change\ in\ feature}$$

3. Implementation

This section explains the different implementation steps followed to reproduce the results obtained in the paper. The paper demonstrates the ability of signature descriptors to generate predictive and interpretable QSPR models by modelling seven diverse materials data sets. I have chosen and modelled "**aqueous solubility**" property and compared the results obtained with those in the paper. The dataset for this is associated with the publication [6] and available from the http://pubs.acs.org.

3.1 Dataset:

Summary of dataset				
Property	**Dataset size**	**Train-Test split**	**No of signature descriptors generated**	**Modelled value**
AQSOL	1143	(80-20 %)	3378	Log (octanol-water partition coefficient)

3.2 RDKit Package

RDKit is an opensource cheminformatics toolkit written in C++ and Python. In this project, for converting the SMILES structure to .mol format the RDKit package is utilized. RDKit package can be installed using Anaconda prompt using the following command:

```
conda install -c rdkit rdkit
```

```
# the following code will creat mol files for the 1143 compounds in the data set from their smiles representation
from rdkit import Chem
df=pd.read_csv("aqsol_data.txt",header=None)
# the format of the csv file is (Chemical name of the molecule,Aqsol true, predicted aqsol, smiles representation)
name=df[0]
smiles=df[3]
for i in range(len(smiles)):
    molObj = Chem.MolFromSmiles(smiles[i])
    imported = Chem.MolToMolBlock(molObj)
    with open(str(name[i])+".mol") as newfile:
        newfile.write(imported)
```

[6] John S. Delaney, 'ESOL: Estimating Aqueous Solubility Directly from Molecular Structure', *Journal of Chemical Information and Computer Sciences*, 44.3 (2004), 1000–1005 <https://doi.org/10.1021/ci034243x>.

Example Mol file generated for 1,1,1,2- Tetrachloroethane:

1,1,1,2-Tetrachloroethane - Notepad

File Edit Format View Help

```
     RDKit          2D

  6  5  0  0  0  0  0  0  0  0999 V2000
   -1.2990    0.7500    0.0000 Cl  0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    1.2990    0.7500    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.5981    1.5000    0.0000 Cl  0  0  0  0  0  0  0  0  0  0  0  0
    2.0490   -0.5490    0.0000 Cl  0  0  0  0  0  0  0  0  0  0  0  0
    0.5490    2.0490    0.0000 Cl  0  0  0  0  0  0  0  0  0  0  0  0
  1  2  1  0
  2  3  1  0
  3  4  1  0
  3  5  1  0
  3  6  1  0
M  END
```

3.3 SSCAN package (Linux Executable)

sscan is an open-source package that has the algorithm to generate signature descriptor of compounds from their mol file. It is in Linux executable format.

1. Download the package from https://sourceforge.net/projects/molsig/

2. An executable sscan file will be generated on running make file.

```
INSTALLATION
1. Go to src folder and run "make".
2. Executable "sscan" will be installed in bin folder.
```

```
srijoo@LAPTOP-I3EGEPHT:~$ cd /mnt/c/Users/sreen/Desktop/molsig-code/src
srijoo@LAPTOP-I3EGEPHT:/mnt/c/Users/sreen/Desktop/molsig-code/src$ make -f Makefile
/usr/bin/gcc -o ../bin/sscan -m32 \
        allocation_da.o random_da.o structure_da.o structure_nb.o structure_tp.o structure_ut.o twin_da.o hea.o structur
e_he.o allocation_si.o signature.o cansig.o candag.o utility_si.o signature_mo.o in_mo.o out_mo.o utility_mo.o   \
        DFConst.o DFManageMemory.o DFFindGeometry.o PCStereoSig.o PCcip.o \
error.o vector.o main.o  -lm
```

3. Signature descriptors can be generated using the sscan file for all the compounds in our dataset up to depth 4 from its mol file (The paper has generated up to depth 7 but due to limited computational capacity I generated up to depth 5).

```
Stereo Signature Molecular Descriptor

Usage: sscan <file.mol> <output-type>
       <file> is mol file
       <output-type>
output types are (x is the depth, any integer between 0 and 100000):
        scanx: signature DAG
        sscanx: stereo signature DAG (CIP rules)
        fscanx: fast stereo signature mode
```

```
srijoo@LAPTOP-I3EGEPHT:/mnt/c/Users/sreen/Desktop/molsig-code/bin$ ./sscan 1,1,1,2-Tetrachloroethane.mol fscan5
```

4. Signature descriptors for all the molecules are generated and compiled

3.4 Implementation approach

3.4.1 Feature set preparation

Compiling all the descriptors and removal of duplicates; creation of [X] matrix

```
# Compiling all the generated signature descriptors
for i in range(len(filename)):
    for j in range(5):
        new=pd.read_csv(str(filename[i])+".fscan"+str(j),header=None,sep=" ")
        new=new[:-1]
        for k in range(len(new)):
            feature.append(str(new[1][k]))
```

```
len(feature)
```

```
37546
```

```
#removal of duplicates
final_feature = []
for i in feature:
    if i not in final_feature:
        final_feature.append(i)
```

```
len(final_feature)
```

```
3378
```

```
# Mapping each compound to its corresponding features for model building
X= np.zeros((1143,3379))
blank_df = pd.DataFrame(X,columns=final_feature+['Chemical name'])
map_dict1 = {}
map_dict2 = {}
for i in range(len(filename)):
    for j in range(5):
        new=pd.read_csv(str(filename[i])+".fscan"+str(j),header=None,sep=" ")
        new=new[:-1]
        for k in range(len(new)):
            feature.append(new[1][k])
            if new[1][k] not in list(map_dict2.keys()):
                map_dict2.update([(new[1][k],[new[0][k]])])
        act_feature =list(map_dict2.keys())
    feature_df = pd.DataFrame.from_dict(map_dict2)
    blank_df.loc[i,act_feature] = feature_df.loc[0,act_feature]
    blank_df.loc[i,'Chemical name'] = filename[i]
    map_dict2={}
#saving the generated matrix to a csv to be exported for model building
blank_df.to_csv("file1.csv")
```

3.4.2 Data Matrix preparation, preprocessing and normalization

Generating X (feature set) and Y (prediction) matrix for model building .Removal of highly corelated variables (higher than 90 percent correlation)

```
# X and y matrix generation
y_vec=pd.read_csv("aqsol data.txt",header=None)
Y=y_vec[1] #Target matrix ()
Y=np.array(Y)
x_vec=pd.read_csv("file1.csv",index_col=0)
X=x_vec.values
X=X[:,:-1] #feature matrix
```

```
print("total no of generated signature descriptors",len(x_vec.columns))
new_vec=x_vec.iloc[:,:-1]
```

```
total no of generated signature descriptors 3379
```

```
# function to remove features with more than 90 percent correlation
def correlation(dataset, threshold):
    col_corr = set() # Set of all the names of deleted columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if (corr_matrix.iloc[i, j] >= threshold) and (corr_matrix.columns[j] not in col_corr):
                colname = corr_matrix.columns[i] # getting the name of column
                col_corr.add(colname)
                if colname in dataset.columns:
                    del dataset[colname] # deleting the column from the dataset

    return(dataset)
uncor=correlation(new_vec,0.9)
X=uncor.values
X=X.astype(int)
```

```
print("total no of signature descriptors after removing feature with more than 90% correlation",len(uncor.columns))
```

```
total no of signature descriptors after removing feature with more than 90% correlation 1571
```

Data Normalisation

```
Scaler= MinMaxScaler()
X_scaled=Scaler.fit_transform(X)
```

3.4.3 L1 feature selection (hyperparameter tuning for the best model)

```
param_grid = {'alpha': uniform()}
model = Lasso()
rand_search = RandomizedSearchCV(estimator=model,
                                 param_distributions=param_grid,random_state=40,
                                n_iter=200)

rand_search.fit(X_scaled, Y)

print(rand_search.best_estimator_.alpha)
print(rand_search.best_score_)
best_alpha=rand_search.best_estimator_.alpha
```

```
0.0042243686391818525
0.8037845591191454
```

3.4.4 MLR model with best hyperparameters chosen from gridsearch

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, random_state=40)
scaling= MinMaxScaler()
X_train=scaling.fit_transform(X_train)
X_test=scaling.transform(X_test)
```

```
model_lasso = Lasso(alpha=best_alpha)
model_lasso.fit(X_train,y_train)
y_pred_train=model_lasso.predict(X_train)
y_pred_test=model_lasso.predict(X_test)
r2train= r2_score(y_train,y_pred_train)
r2test=r2_score(y_test,y_pred_test)
```

```
r2train,r2test
```

```
(0.8777022860095753, 0.8276239373798526)
```

3.4.5 Features chosen from L1 feature selection technique and feature significance

```
coeff = model_lasso.coef_
a=[]
for i in range(len(coeff)):
    if(coeff[i]!=0):
        a.append(i)
XNN=X[:,a]
print(" No of Signature descriptors selected using L1 feature selection method:",len(XNN[0]))
```

```
No of Signature descriptors selected using L1 feature selection method: 83
```

```
dict={'weights':coeff[a].astype(float),'descriptor':uncor.columns[a]}
new_df=pd.DataFrame(data=dict)

```

```
new_df.sort_values(by= [new_df.columns[0]],axis=0, ascending=True, inplace=True)
print(new_df.head(12))
print("\n")
print(new_df.tail(12))
```

3.4.6 Regularized Neural Network model building with chosen features

Baseline model and performance:

```
X_train, X_test, y_train, y_test = train_test_split(XNN, Y, test_size=0.20, random_state=40)
scaling= MinMaxScaler()
X_train=scaling.fit_transform(X_train)
X_test=scaling.transform(X_test)
```

```
#baseline model
baseline_model = Sequential([
  Dense(len(X_train[0]), activation='relu'),
  Dense(8, activation='relu'),
  Dense(1,activation='linear')])
# Compile the model.
baseline_model.compile(optimizer='adam',loss='mean_squared_error')
# Train the model.
history = baseline_model.fit(
  X_train,
  y_train,
  epochs=100,
  validation_data = (X_test,y_test)
)
```

Hyperparameter tuning of the neural network model to improve model performance

```
#defining callback to save the best model during training
def model_callbacks(weight_file='Weights_v3{epoch:02d}.h5'):
    checkpoint = ModelCheckpoint(weight_file, monitor = 'val_loss', verbose=1, save_best_only= True,save_weights_only= True,
    #checkpoint = ModelCheckpoint(weight_file, monitor = 'val_loss', verbose=1)
    return [checkpoint]
#defining function to grid search best hyperparameter for learning rate and batch_size
def my_nn_model(alpha=0.001):
    model = Sequential([
    Dense(len(X_train[0]), activation='relu',activity_regularizer = regularizers.l2(1e-3)),
    Dropout(0.2),
    Dense(8, activation='relu',activity_regularizer = regularizers.l2(1e-4)),
    Dropout(0.01),
    Dense(1,activation='linear')])
    model.compile(optimizer=Adam(learning_rate=alpha),loss='mean_squared_error')
    return model
```

```
#searching for the best hyperparameter and saving the weights of the best model
step_size_list = [0.0001,0.001,1e-1,1e-2]
batch_size_list = [16,32,64,128]
param_grid = {'step_size': step_size_list,'batch_size': batch_size_list}
grid_search = list(ParameterGrid(param_grid))
val_loss = []
for index,item in enumerate(grid_search):
    test_model = my_nn_model(item['step_size'])
    weight_file='Weights_gs'+str(index)+'{epoch:02d}.h5'
    history = test_model.fit(X_train,y_train,epochs=500,batch_size=item['batch_size'],validation_data = (X_test,y_test),call
    val_loss.append(min(history.history['val_loss']))

```

Performance of the tuned neural network model:

```
Y_pred=test_model.predict(X_test)
from sklearn.metrics import r2_score
print('R squared (test):',r2_score(y_test,Y_pred))
from sklearn.metrics import mean_squared_error,mean_absolute_error
print('Root mean squared error (test):',np.sqrt(mean_squared_error(y_test,Y_pred)))
print('Mean absolute error (test):',mean_absolute_error(y_test,Y_pred))
```

```
R squared (test): 0.8686955540327765
Root mean squared error (test): 0.7180816030369687
Mean absolute error (test): 0.5187011401268072
```

```
Y_pred2=test_model.predict(X_train)
from sklearn.metrics import r2_score
print('R squared (train):',r2_score(y_train,Y_pred2))
from sklearn.metrics import mean_squared_error,mean_absolute_error
print('Root mean squared error (train):',np.sqrt(mean_squared_error(y_train,Y_pred2)))
print('Mean absolute error (train):',mean_absolute_error(y_train,Y_pred2))
```

```
R squared (train): 0.9743088772087087
Root mean squared error (train): 0.3403829653076838
Mean absolute error (train): 0.24099120221265816
```

4. Results obtained from implementation vs results obtained in paper

4.1 Multiple Linear regression model

Performance of Multiple Linear regression model from my implementation					
Property	**No of Sig descriptors**	**R^2 test**	**R^2 train**	**RMSE test**	**RMSE train**
AQSOL	83	0.827	0.877	0.8227	0.7426

Performance of Multiple Linear regression model in the review paper					
Property	**No of Sig descriptors**	**R^2 test**	**R^2 train**	**RMSE test**	**RMSE train**
AQSOL	258	0.88	0.91	0.72	0.62

Summary of AQSOL models in literature trained on tradition descriptors (e.g. Dragon)

		training set		test set	
study	data set size	r^2	SEE	r^2	SEP
Duchowicz et al.[58]	97 train 48 test 21 ext test	0.87	0.903	0.85	0.899 1.202
Wang et al.[8]	3664 LOO 85/15 crossval.	0.83 0.84	0.840	0.84 0.83	0.823 0.841
Delaney[15]	2874	0.72	0.97		
Votano et al.[23]	3343/772 1674/166	0.88 0.88	0.74 0.61	0.77 0.84	0.91 0.75
Johnson et al.[28]	581	0.88	0.61	0.93	0.50
this work	4376	0.90 ± 0.00	0.645 ± 0.001	0.90 ± 0.00	0.665 ± 0.003

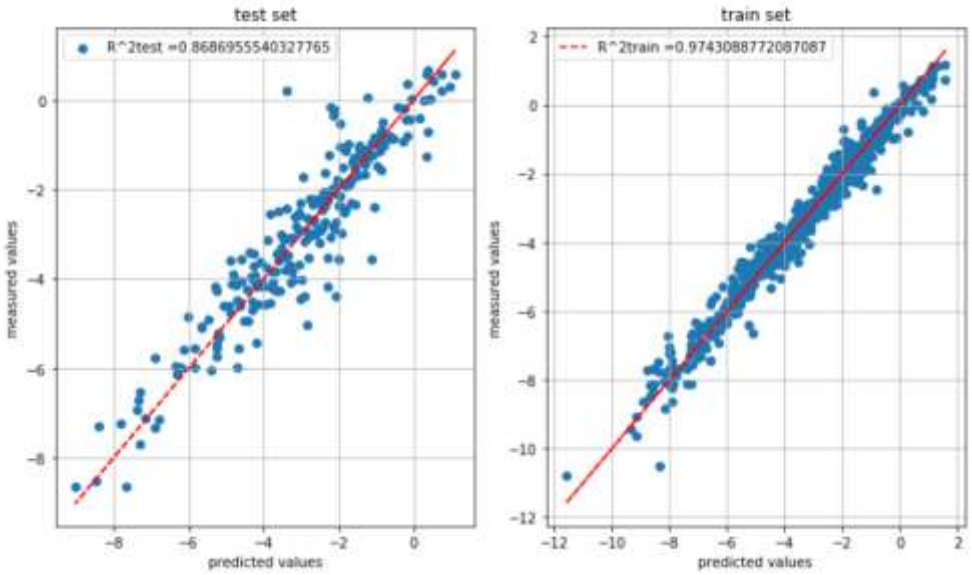
Clearly, the AQSOL model derived from signature descriptors could predict the properties of the

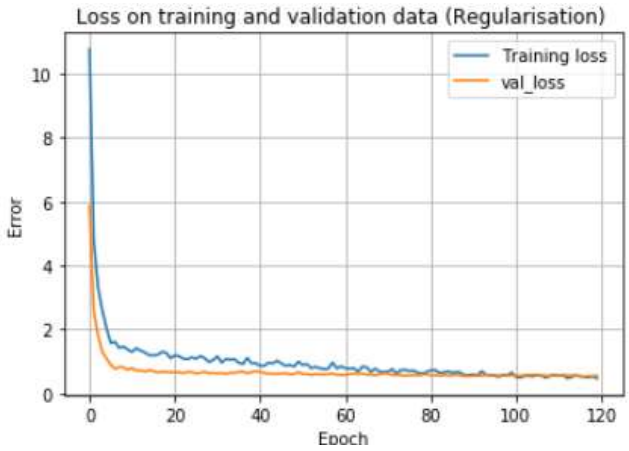
training and test set with good fidelity with r2 values of 0.877 and 0.827 and standard errors of 0.82 and 0.74 logS for the training and test sets, respectively. This is comparable with many comprehensive and accurate AQSOL models available in the literature, trained on Dragon descriptors. There is a small difference between the results in the paper and obtained from my implementation but the accuracy of the model is still very similar. The small difference arises mostly due to generating signature descriptors to a lower depth (5) compared to 7 in the paper. (due to limited computational power

4.2 Neural Network model:

Performance of neural network model (Non linear) from my implementation					
# of hidden layers	**# of hidden layer neuron**	**R^2 test**	**R^2 train**	**RMSE test**	**RMSE train**
1	8	0.868	0.974	0.718	0.34

Performance of neural network model (Non linear) from the paper					
# of hidden layers	**# of hidden layer neuron**	**R^2 test**	**R^2 train**	**RMSE test**	**RMSE train**
1	2	0.91	0.97	0.65	0.39





4.3 Descriptors with most influence on Aqsol property

Most relevant features from my implementation (negative MLR coefficients reduce solubility, and positive MLR coefficients increase solubility). It matches to a good accuracy with the result in paper

weights	descriptor
-7.288989	[C]
-5.702998	[Cl]
-3.243925	[C]([C]([C])[C]([C]))
-3.157147	[C]=[C]
-2.818981	[Br]
-2.283436	[C]([C]=[C])[C]([C]=[C])
-1.813920	[C]([C][C])
-1.813215	[P]=[S]
-1.653292	[N]([C][O]=[O])
-1.627357	[I]
-1.532294	[S]
-1.293141	[C]([C]=[C][Cl])

weights	descriptor
0.536141	[C]=[O]
0.541980	[C]([C])[C]([C][C][O])
0.546411	[C]([C])[C]([O])
0.555033	[C](\=\[C][C])[O]
0.706211	[O]=[P]
0.733317	[C]([C]([C])[C]([C][O]))
2.160589	[O]([C])
2.564086	[O]

Results from the paper:

Signature descriptor	MLR weights	Signature descriptor	MLR weights
C=C	−0.52	O=P	+0.07
C(CC)	−0.44	C(C=CS)	+0.07
Cl	−0.39	C=O	+0.08
S	−0.21	O(C)	+0.22
C(C=C)C(C=C)	−0.15		
C(C)	−0.14		

5. Conclusion

We have been able to demonstrate that signature descriptors generate robust and predictive models for polymer properties relevant to their biological applications. ML-based models trained using traditional but hard-to-interpret molecular descriptors and molecular signature descriptors exhibited similar prediction accuracies for the structure–property relationships models studied. The signature descriptors have the advantage of more direct molecular interpretability as the most important **signatures can be easily mapped back onto exemplar materials**

6. References

Burden, F. R., and Dave A. Winkler, 'Optimal Sparse Descriptor Selection for QSAR Using Bayesian Methods', *QSAR and Combinatorial Science*, 28.6–7 (2009), 645–53 <https://doi.org/10.1002/qsar.200810173>

Carbonell, Pablo, Lars Carlsson, and Jean Loup Faulon, 'Stereo Signature Molecular Descriptor', *Journal of Chemical Information and Modeling*, 53.4 (2013), 887–97 <https://doi.org/10.1021/ci300584r>

Delaney, John S., 'ESOL: Estimating Aqueous Solubility Directly from Molecular Structure', *Journal of Chemical Information and Computer Sciences*, 44.3 (2004), 1000–1005 <https://doi.org/10.1021/ci034243x>

CH5650 : Molecular Data Science and Informatics Project report

Faulon, Jean Loup, Michael J. Collins, and Robert D. Carr, 'The Signature Molecular Descriptor. 4. Canonizing Molecules Using Extended Valence Sequences', *Journal of Chemical Information and Computer Sciences*, 44.2 (2004), 427–36 <https://doi.org/10.1021/ci0341823>

Fujita, Toshio, and David A. Winkler, 'Understanding the Roles of the "Two QSARs"', *Journal of Chemical Information and Modeling*, 56.2 (2016), 269–74 <https://doi.org/10.1021/acs.jcim.5b00229>

Mikulskis, Paulius, Morgan R. Alexander, and David Alan Winkler, 'Toward Interpretable Machine Learning Models for Materials Discovery', *Advanced Intelligent Systems*, 1.8 (2019) <https://doi.org/10.1002/aisy.201900045>