Drug Discovery of COVID19

Python version :- 3.6.9

How to Run : -

1. Install mol2vec.
   ```
   pip install git+https://github.com/samoturk/mol2vec
   ```
2. Install rdkit.
   ```
   wget -c https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
   chmod +x Miniconda3-latest-Linux-x86_64.sh
   time bash ./Miniconda3-latest-Linux-x86_64.sh -b -f -p /usr/local
   time conda install -q -y -c conda-forge rdkit
   ```

   import sys

   ```
   sys.path.append('/usr/local/lib/python3.7/site-packages/')
   ```

3. Download file `model_300dim.pkl`.
4. Change hardcoded path

```
fdf = pd.read_csv('/content/data/22bba507-efcf-4af0-b9d0-f26193605457_train.csv')
tdf = pd.read_csv('/content/final_test_q3/4b566bb4-3155-49ff-91e1-19942504b20c_test.csv')
```

Approaches :- Two approaches have been taken. One is using rdkit to extract features of molecules.

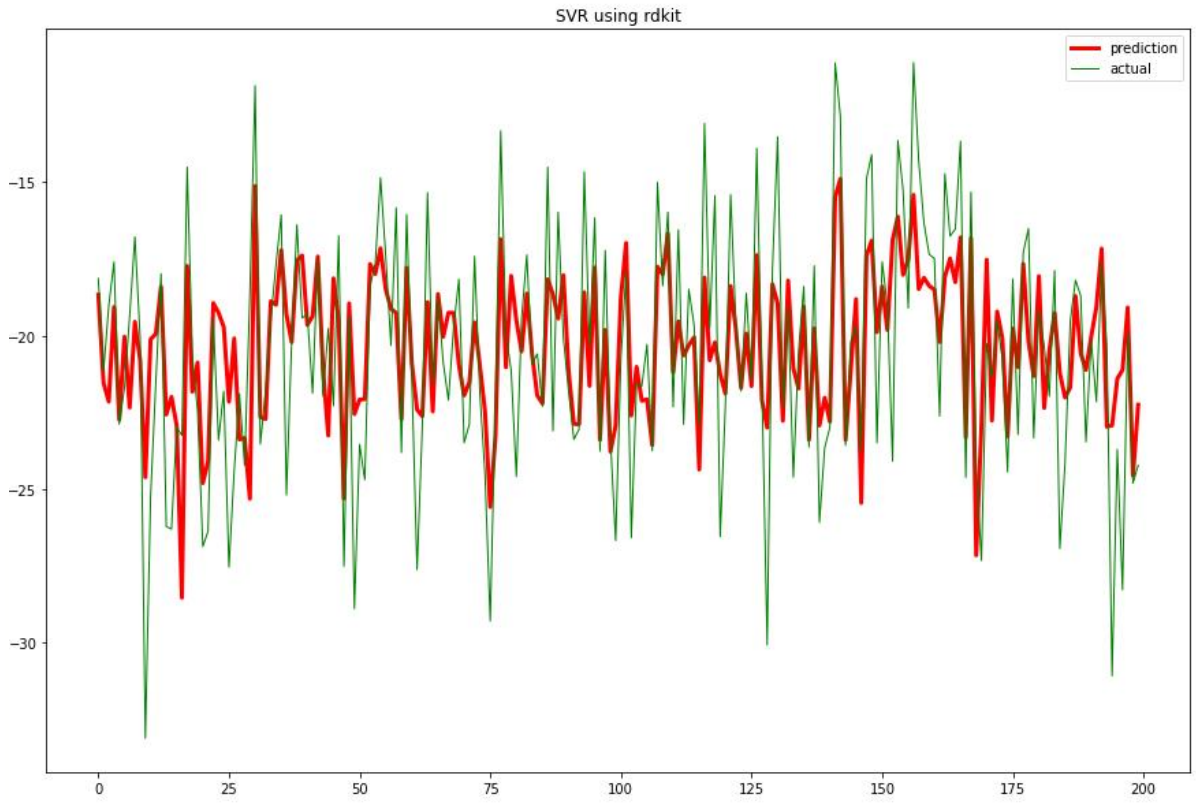
Second approach is using pretrained mol2vec model of 300 dimensions.

Approach 1(Extracting feature using rdkit) :-

In this approach rdkit is used to extract features from the molecules. The extracted features are like Number of atoms, number heavy atoms, Number H donors, Number O donors, Number of C atoms, Number of N atoms, number of O atoms, number of H atoms, number of S atoms, molecular weight, weight of heave atoms etc.

1. **Support Vector Regressor with RBF kernel(**`kernel = 'rbf',C =50,gamma='scale'`**) :-**
   ```
   svr = SVR(kernel = 'rbf',C =50,gamma='scale')
   svr.fit(x_train, y_train)
   ```

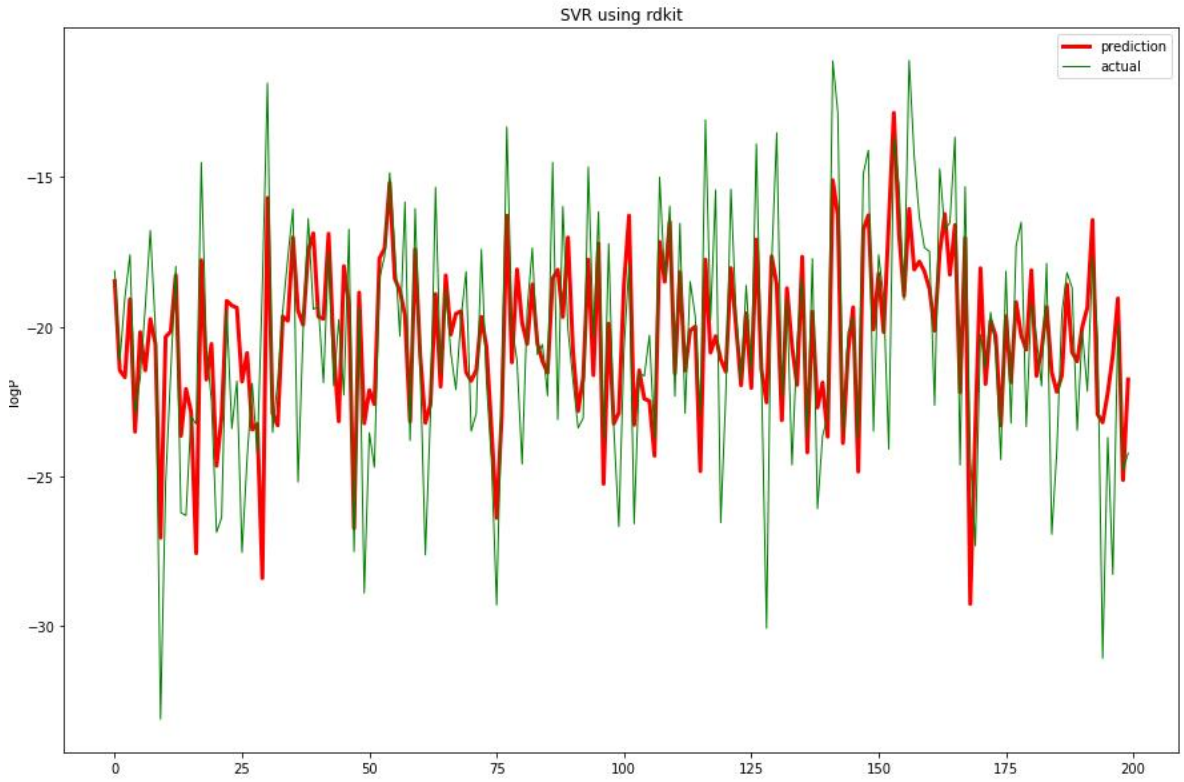
   It's been noticed that C is the regularization parameter and regularization is inversely proportional to regularization. So if we increase the C the then regularization will be reduced. Keeping that on mind, I have increased the C until I have found the desired result. And I have found the C value as 50.
   Root mean squared error - `2.714003186551878`
   Mean absolute Error - `2.020515548896202`



2. **Support Vector Regressor with Polynomial kernel(`kernel = 'poly',C =50,gamma='scale'`):-**

```
svr = SVR(kernel = poly,C =50,gamma='scale')
svr.fit(x_train, y_train)
```

As explained above, same rule also has been followed here.
Root mean squared error – `2.657798120021596`
Mean absolute error - `1.9780328709433654`

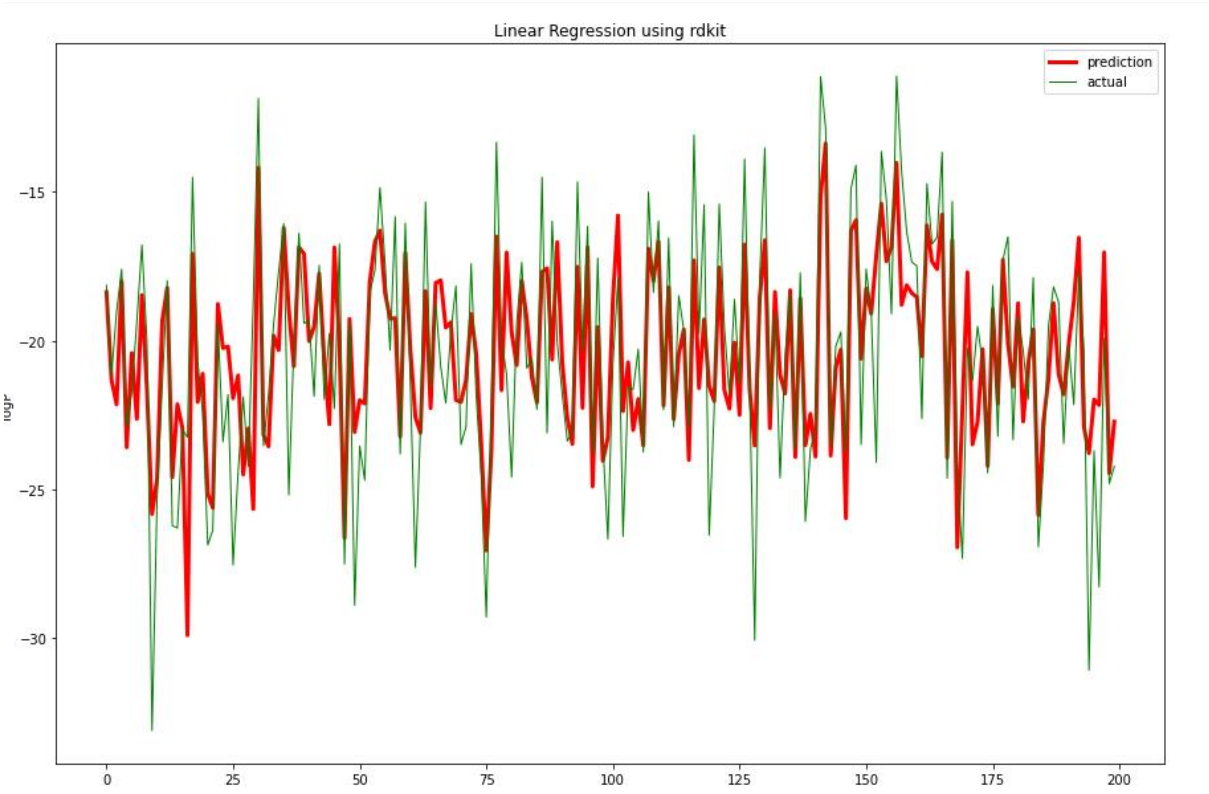


3. **Linear Regression (Default Parameters):-**

```
reg = LinearRegression().fit(x_train, y_train)
```

Linear regression is used with all the default parameters. It's been observed that it works quite good the default parameters.

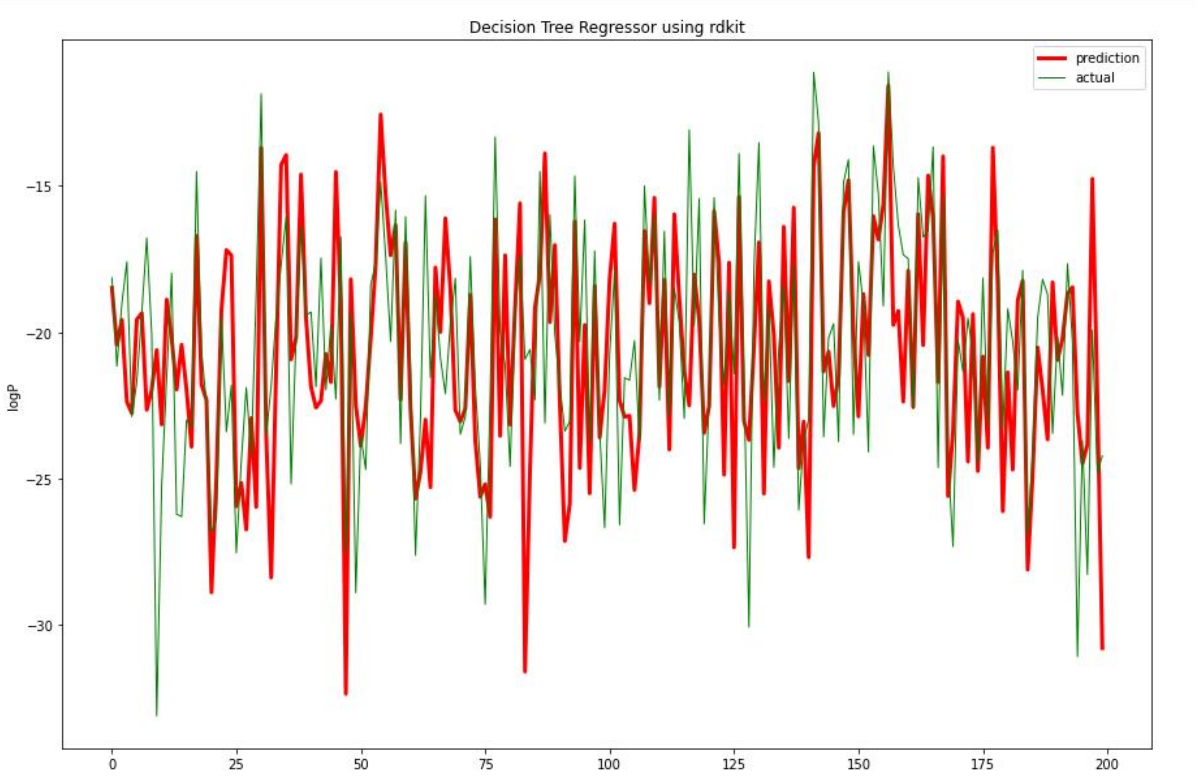
Root mean squared error – `2.585757794339119`
Mean absolute error - `1.9188634942709082`



4. **Decision Tree Regressor (random_state = 0):-**

```
classifier = DecisionTreeRegressor(random_state=0)
classifier.fit(x_train, y_train)
```

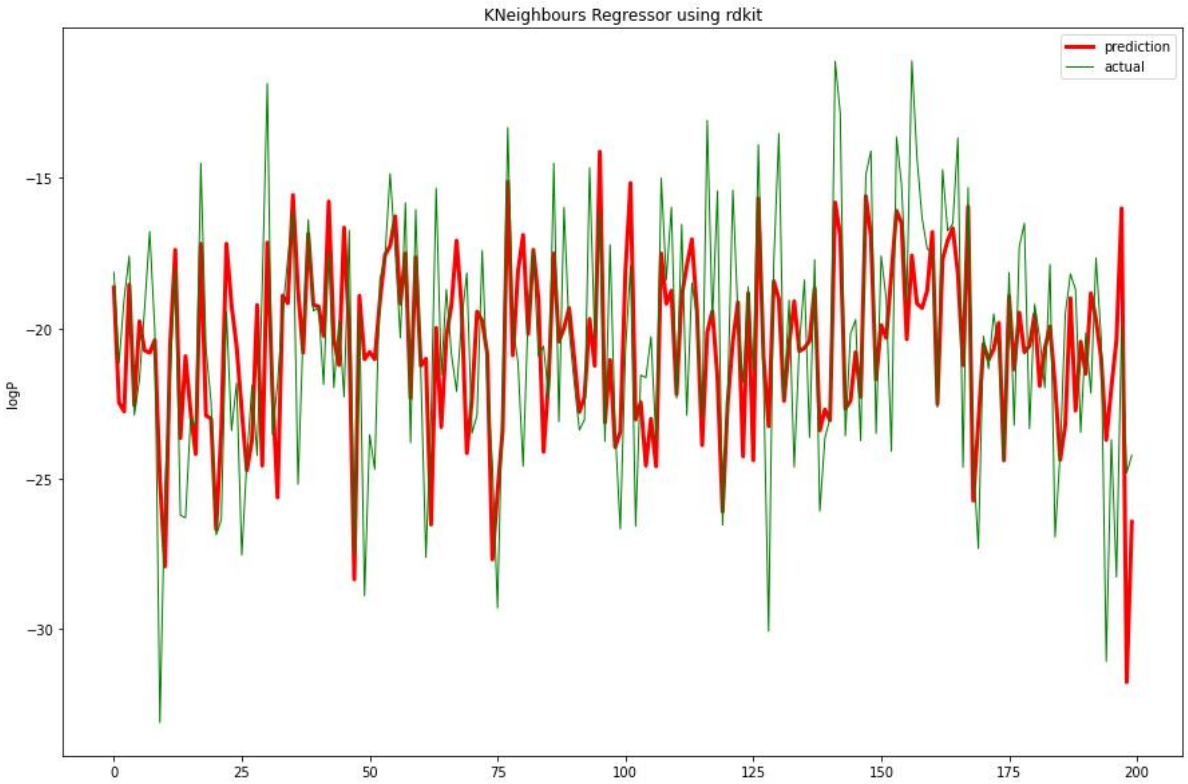
Single decision tree used as regressor is used. It's been used to experiment with simple model and to check how they work.
Root mean squared error – 3.4425048712956015
Mean absolute error - 2.4891097833333333



5. KNeighborsRegressor (Default Parameters):-

```
classifier = KNeighborsRegressor()
classifier.fit(x_train, y_train)
```

Kneighbours Regressor is used with default neighbour of 5.
Root mean squared error – `2.9958244538037317`
Mean Absolute error - `2.2513186055555554`



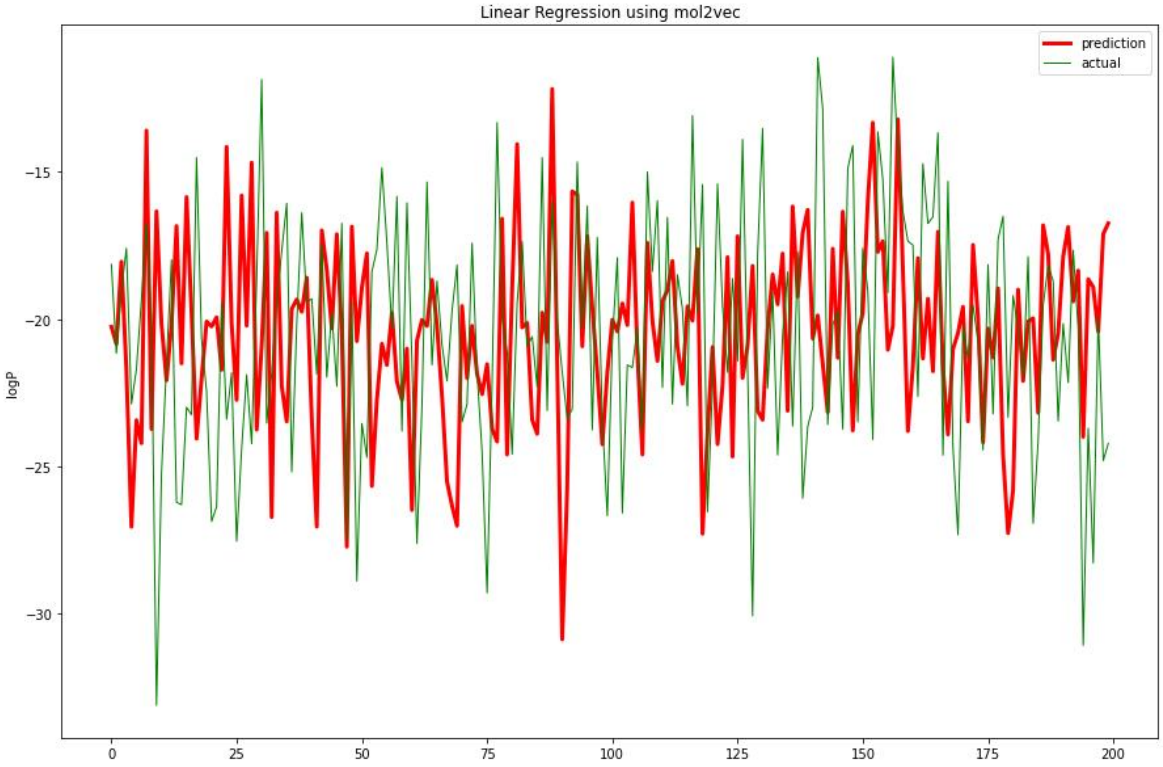
Approach 2(Using Pretrained mol2vec of 300 dimensions):-

Pretrained mol2vec of 300 dimensions is used to form dataset of 300 dimensions.

1. **Linear Regression(Default parameters) :-**

```
mreg = LinearRegression().fit(mx_train, my_train)
```

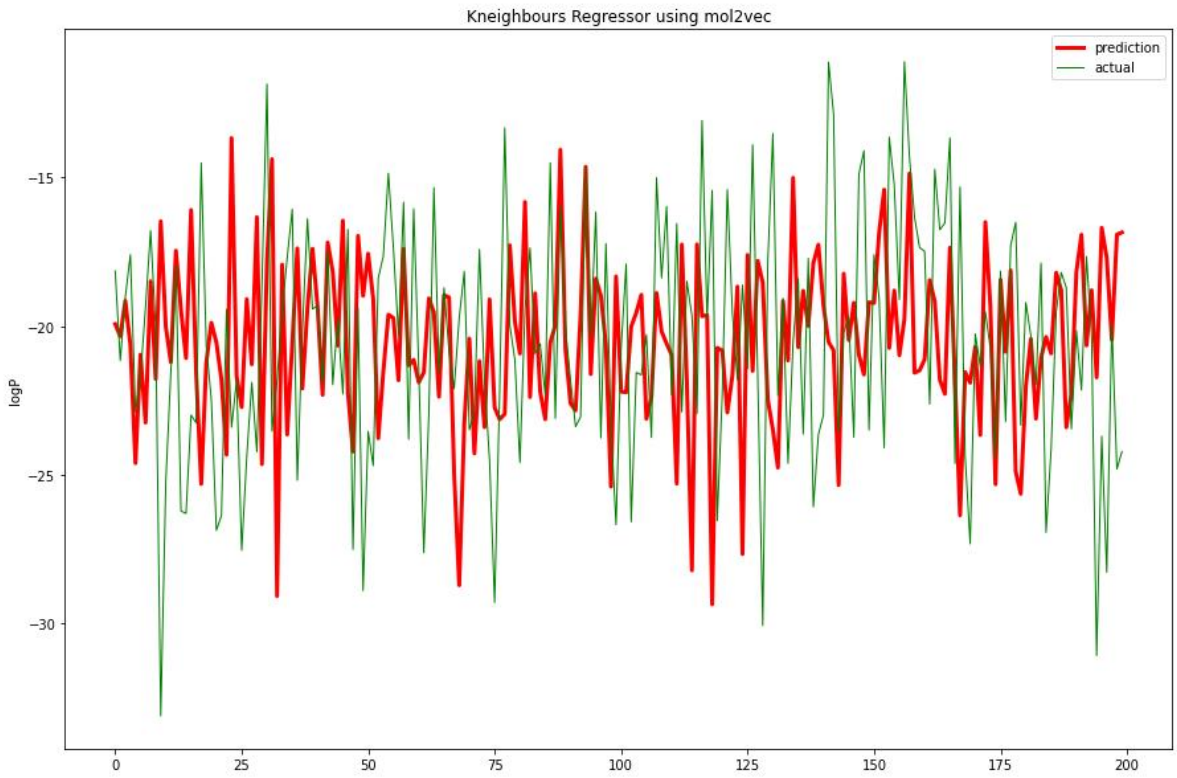
Linear regression is used with all the default parameters. It's been observed that it works quite good the default parameters.
Root mean squared error – 2.4541342918568545
Mean absolute error- 1.7645788399556477



2. KNeighbours Regressor(Default Parameters) :-

```
mclassifier = KNeighborsRegressor()
mclassifier.fit(mx_train, my_train)
```

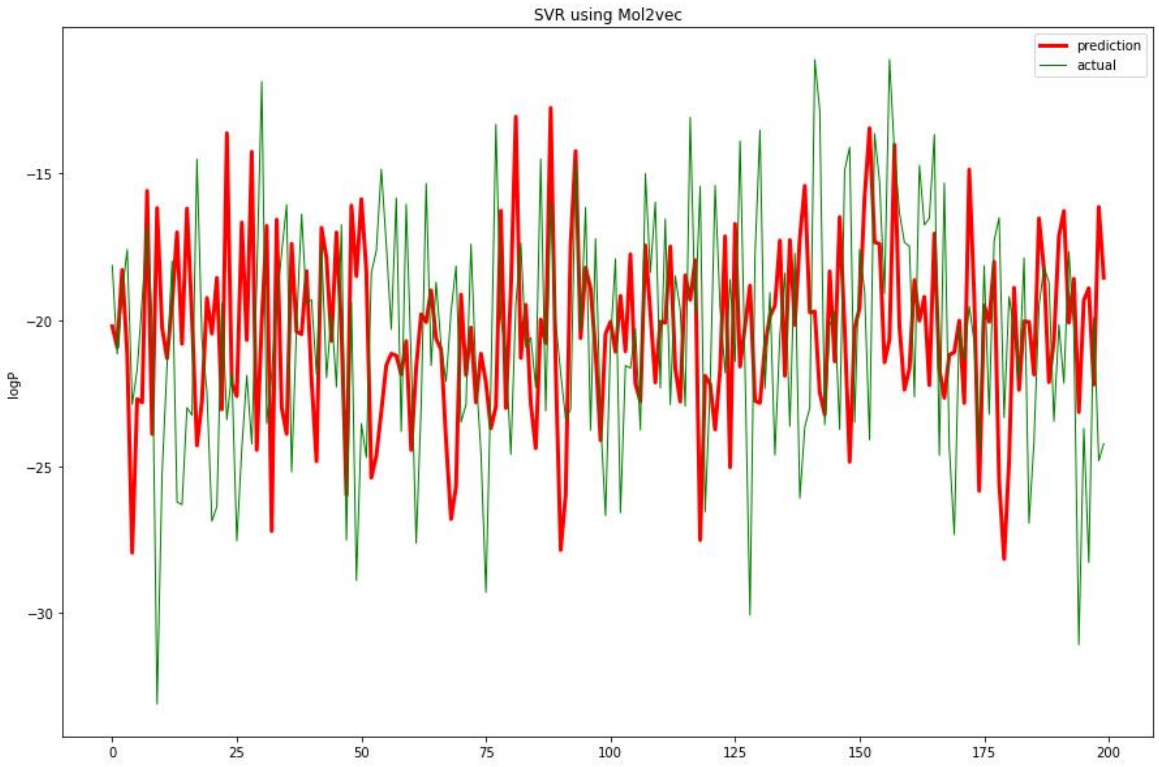
Kneighbours Regressor is used with default neighbour of 5.
Root mean squared error – 2.8592695955481378
Mean Absolute error - 2.0805104644444445



3. **Support Vector Regressor with RBF kernel (`kernel='rbf',C=100,epsilon=1.5,gamma='scale'`):-**

```
svr = SVR(kernel='rbf',C=100,epsilon=1.5,gamma='scale')
svr.fit(mx_train, my_train)
```

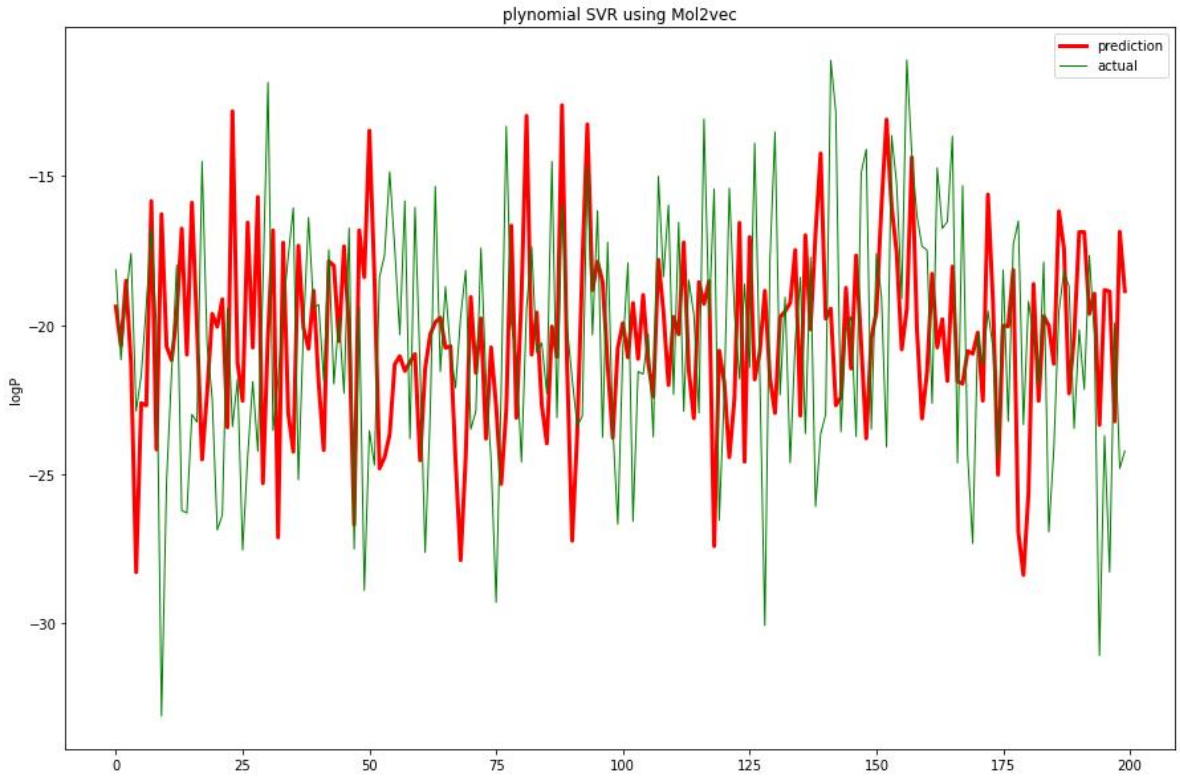
It's been noticed that C is the regularization parameter and regularization is inversely proportional to regularization. So if we increase the C the then regularization will be reduced. Keeping that on mind, I have increased the C until I have found the desired result. And I have found the C value as 100, with that I have varied epsilon value to 1.5 to get the maximized result. Epsilon basically does not penalize loss function if the predicted value is within epsilon-distance from actual value.
Root mean squared error - `2.371103408306654`
Mean absolute error - `1.6704595158708813`



4. Support Vector Regressor with Polynomial kernel (`kernel=poly,C=100,epsilon=1.5,gamma='scale'`):-

```
svr = SVR(kernel='poly',C=100,epsilon=1.5,gamma='scale')
svr.fit(mx_train, my_train)
```

As explained above, The C and epsilon is tuned.
Root mean squared error – 2.430427449541101
Mean absolute error - 1.7301253437800772



Final Model :- So from all the observations it's been noticed that the support vector regressor with C as 100 and epsilon as 1.5 suing the pretrained mol2vec of 300 dimensions gives the least root mean squared error.

```
svr = SVR(kernel='rbf',C=100,epsilon=1.5,gamma='scale')
svr.fit(mx_train, my_train)
```