pIC50-model

November 5, 2022

1 Model to Predict pIC50 based on SMILES

Aim: Create machine learning model to predict pIC50 values of SMILES generated by the DrugEx Demo

[460]:
```
from pathlib import Path
import tensorflow as tf
import keras
from sklearn.model_selection import train_test_split
from keras.layers import Input, Dense
import pandas as pd
from rdkit import Chem
from rdkit.Chem import MACCSkeys
from rdkit.Chem.AllChem import GetMorganFingerprintAsBitVect
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
```

[461]:
```
HERE = Path(_dh[-1])
DATA = HERE / "data"
```

2 Load Data

The data of Lipinski drugs will be used in order to have more data available

[462]:
```
data = pd.read_csv(DATA / "PTP1B_compounds_lipinski.csv")

#inactive & active classification
data['active'] = (data['pIC50']>7)
data
```

[462]:
```
   Unnamed: 0.1  Unnamed: 0 molecule_chembl_id        IC50 units  \
0            14          14       CHEMBL411295         7.0    nM
1            16          16       CHEMBL377141        10.0    nM
2            18          18      CHEMBL1938829        11.0    nM
3            22          22       CHEMBL604457        12.6    nM
```

```
4                23           23       CHEMBL592290        14.0    nM
…             …         …                 …         …    …
1987          2977         2977         CHEMBL99971  12000000.0    nM
1988          2978         2978        CHEMBL324473  12200000.0    nM
1989          2980         2980        CHEMBL100267  20000000.0    nM
1990          2981         2981         CHEMBL95668  28000000.0    nM
1991          2982         2982        CHEMBL330395  29000000.0    nM

                                                 smiles     pIC50  \
0     O=C(O)c1cccc(/C=C/c2ccc3cc(Br)c(C(F)(F)P(=O)(O…  8.154902
1     O=C1CC(c2ccc(C[C@H](NS(=O)(=O)c3cccc(C(F)(F)F)…  8.000000
2     O=C(O)COc1ccc(S(=O)(=O)N(Cc2ccc(-c3csnn3)cc2)C…  7.958607
3     CS(=O)(=O)OC1C(Cl)CCCC1Cc1ccc(N2CC(=O)CS2(=O)=…  7.899629
4            O=C1CN(c2c(O)cc(CC3CCCC3)cc2F)S(=O)(=O)C1  7.853872
…                                                   …       …
1987                      O=C(O)c1ccc2c(C(=O)O)c(O)ccc2c1  1.920819
1988      C=CCOC(=O)/C=C\c1cccc(C(F)(F)P(=O)(O)O)c1.N.N  1.913640
1989                       COc1ccc2cc(C(=O)O)ccc2c1C(=O)O  1.698970
1990  O=S(=O)([O-])c1cccc2c(S(=O)(=O)[O-])cccc12.[Na…  1.552842
1991              CS(=O)(=O)Nc1cccc2cc(S(=O)(=O)O)ccc12  1.537602

      molecular_weight  n_hba  n_hbd    logp  molecular_weight.1  n_hba.1  \
0           482.968278      3      3  5.0929          482.968278        3
1           652.024097      6      2  2.3154          652.024097        6
2           694.056877      9      3  4.9739          694.056877        9
3           451.052622      7      1  1.4059          451.052622        7
4           341.109707      4      1  2.3730          341.109707        4
…                  …    …    …     …                  …      …
1987        232.037173      3      3  1.9418          232.037173        3
1988        352.099965      5      4  2.9800          352.099965        5
1989        246.052823      3      2  2.2448          246.052823        3
1990        331.940118      6      0 -5.3440          331.940118        6
1991        301.007864      4      2  1.4580          301.007864        4

      n_hbd.1  logp.1  ro5_fulfilled  active
0           3  5.0929           True    True
1           2  2.3154           True    True
2           3  4.9739           True    True
3           1  1.4059           True    True
4           1  2.3730           True    True
…         …     …              …      …
1987        3  1.9418           True   False
1988        4  2.9800           True   False
1989        2  2.2448           True   False
1990        0 -5.3440           True   False
1991        2  1.4580           True   False
```

```
[1992 rows x 17 columns]
```

2.1 Smiles to Fingerprint

[463]:
```python
def smiles_to_fp(smiles, method="maccs", n_bits=2048):
    """
    Encode a molecule from a SMILES string into a fingerprint.

    Parameters
    ----------
    smiles : str
        The SMILES string defining the molecule.

    method : str
        The type of fingerprint to use. Default is MACCS keys.

    n_bits : int
        The length of the fingerprint.

    Returns
    -------
    array
        The fingerprint array.

    """

    # convert smiles to RDKit mol object
    mol = Chem.MolFromSmiles(smiles)

    if method == "maccs":
        return np.array(MACCSkeys.GenMACCSKeys(mol))
    if method == "morgan2":
        return np.array(GetMorganFingerprintAsBitVect(mol, 2, nBits=n_bits))
    if method == "morgan3":
        return np.array(GetMorganFingerprintAsBitVect(mol, 3, nBits=n_bits))
    else:
        # NBVAL_CHECK_OUTPUT
        print(f"Warning: Wrong method specified: {method}. Default will be used␣
↪instead.")
        return np.array(MACCSkeys.GenMACCSKeys(mol))
```

2.2 Splitting Data into Train and Test

[464]:
```python
X_train, X_test, y_train, y_test = train_test_split(data['smiles'].values,␣
↪data['active'].values, test_size=0.3)
```

[465]:
```
fp_X_train = []
for i in range(len(X_train)):
    fp_X_train.append(smiles_to_fp(X_train[i]))
```

[466]:
```
fp_X_test = []
for i in range(len(X_test)):
    fp_X_test.append(smiles_to_fp(X_test[i]))
```

[467]:
```
fp_X_train = np.array(fp_X_train)
fp_X_test = np.array(fp_X_test)
fp_X_train.shape
y_test.shape
```

[467]:
```
(598,)
```

2.3 Neural Network

[509]:
```
#Build model
inputLayer = Input(167,)
Hidden = Dense(32)(inputLayer)
Hidden = Dense(32, activation='relu')(Hidden)
Output = Dense(1, activation='sigmoid')(Hidden)
```

[510]:
```
Model = tf.keras.Model(inputs=inputLayer, outputs=Output)
Model.summary()
```

```
Model: "model_40"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_37 (InputLayer)       [(None, 167)]             0

 dense_88 (Dense)            (None, 32)                5376

 dense_89 (Dense)            (None, 32)                1056

 dense_90 (Dense)            (None, 1)                 33

=================================================================
Total params: 6,465
Trainable params: 6,465
Non-trainable params: 0
_________________________________________________________________
```

[515]:
```
Model.compile(optimizer="Adam", loss="binary_crossentropy",␣
 ↪metrics=['accuracy'])
```

```python
History = Model.fit(fp_X_train, y_train, validation_split=0.3, epochs=100)
```

```
Epoch 1/100
31/31 [==============================] - 1s 11ms/step - loss: 0.0303 - accuracy:
0.9856 - val_loss: 0.2003 - val_accuracy: 0.9618
Epoch 2/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0353 - accuracy:
0.9815 - val_loss: 0.2103 - val_accuracy: 0.9642
Epoch 3/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0312 - accuracy:
0.9836 - val_loss: 0.2472 - val_accuracy: 0.9642
Epoch 4/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0325 - accuracy:
0.9815 - val_loss: 0.2139 - val_accuracy: 0.9690
Epoch 5/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0321 - accuracy:
0.9826 - val_loss: 0.1968 - val_accuracy: 0.9618
Epoch 6/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0318 - accuracy:
0.9815 - val_loss: 0.2080 - val_accuracy: 0.9618
Epoch 7/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0311 - accuracy:
0.9846 - val_loss: 0.2153 - val_accuracy: 0.9666
Epoch 8/100
31/31 [==============================] - 0s 7ms/step - loss: 0.0285 - accuracy:
0.9877 - val_loss: 0.2216 - val_accuracy: 0.9666
Epoch 9/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0304 - accuracy:
0.9846 - val_loss: 0.2225 - val_accuracy: 0.9666
Epoch 10/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0265 - accuracy:
0.9877 - val_loss: 0.2286 - val_accuracy: 0.9690
Epoch 11/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0292 - accuracy:
0.9856 - val_loss: 0.2735 - val_accuracy: 0.9618
Epoch 12/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0304 - accuracy:
0.9867 - val_loss: 0.2393 - val_accuracy: 0.9666
Epoch 13/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0264 - accuracy:
0.9887 - val_loss: 0.2490 - val_accuracy: 0.9666
Epoch 14/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0279 - accuracy:
0.9846 - val_loss: 0.2439 - val_accuracy: 0.9690
Epoch 15/100
31/31 [==============================] - 0s 6ms/step - loss: 0.0248 - accuracy:
0.9867 - val_loss: 0.2559 - val_accuracy: 0.9666
```

```
Epoch 16/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0316 - accuracy:
0.9826 - val_loss: 0.2003 - val_accuracy: 0.9570
Epoch 17/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0300 - accuracy:
0.9836 - val_loss: 0.2420 - val_accuracy: 0.9666
Epoch 18/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0258 - accuracy:
0.9877 - val_loss: 0.2400 - val_accuracy: 0.9666
Epoch 19/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0283 - accuracy:
0.9856 - val_loss: 0.2468 - val_accuracy: 0.9666
Epoch 20/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0242 - accuracy:
0.9877 - val_loss: 0.2402 - val_accuracy: 0.9690
Epoch 21/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0308 - accuracy:
0.9846 - val_loss: 0.2201 - val_accuracy: 0.9666
Epoch 22/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0268 - accuracy:
0.9887 - val_loss: 0.2608 - val_accuracy: 0.9690
Epoch 23/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0267 - accuracy:
0.9867 - val_loss: 0.2439 - val_accuracy: 0.9666
Epoch 24/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0240 - accuracy:
0.9867 - val_loss: 0.2633 - val_accuracy: 0.9666
Epoch 25/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0237 - accuracy:
0.9877 - val_loss: 0.2675 - val_accuracy: 0.9666
Epoch 26/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0229 - accuracy:
0.9887 - val_loss: 0.2608 - val_accuracy: 0.9642
Epoch 27/100
31/31 [==============================] - 0s 6ms/step - loss: 0.0251 - accuracy:
0.9856 - val_loss: 0.2821 - val_accuracy: 0.9690
Epoch 28/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0234 - accuracy:
0.9877 - val_loss: 0.2675 - val_accuracy: 0.9690
Epoch 29/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0272 - accuracy:
0.9846 - val_loss: 0.2792 - val_accuracy: 0.9666
Epoch 30/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0249 - accuracy:
0.9887 - val_loss: 0.2494 - val_accuracy: 0.9523
Epoch 31/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0232 - accuracy:
0.9918 - val_loss: 0.3203 - val_accuracy: 0.9666
```

```
Epoch 32/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0243 - accuracy:
0.9846 - val_loss: 0.2565 - val_accuracy: 0.9618
Epoch 33/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0271 - accuracy:
0.9867 - val_loss: 0.2653 - val_accuracy: 0.9618
Epoch 34/100
31/31 [==============================] - 0s 6ms/step - loss: 0.0241 - accuracy:
0.9867 - val_loss: 0.2680 - val_accuracy: 0.9594
Epoch 35/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0206 - accuracy:
0.9908 - val_loss: 0.3308 - val_accuracy: 0.9690
Epoch 36/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0253 - accuracy:
0.9856 - val_loss: 0.2882 - val_accuracy: 0.9690
Epoch 37/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0224 - accuracy:
0.9867 - val_loss: 0.2841 - val_accuracy: 0.9666
Epoch 38/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0215 - accuracy:
0.9908 - val_loss: 0.2728 - val_accuracy: 0.9594
Epoch 39/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0230 - accuracy:
0.9867 - val_loss: 0.2840 - val_accuracy: 0.9690
Epoch 40/100
31/31 [==============================] - 0s 6ms/step - loss: 0.0231 - accuracy:
0.9877 - val_loss: 0.2886 - val_accuracy: 0.9666
Epoch 41/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0236 - accuracy:
0.9887 - val_loss: 0.3070 - val_accuracy: 0.9690
Epoch 42/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0222 - accuracy:
0.9867 - val_loss: 0.2882 - val_accuracy: 0.9690
Epoch 43/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0285 - accuracy:
0.9846 - val_loss: 0.2410 - val_accuracy: 0.9618
Epoch 44/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0357 - accuracy:
0.9826 - val_loss: 0.2875 - val_accuracy: 0.9690
Epoch 45/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0298 - accuracy:
0.9836 - val_loss: 0.3200 - val_accuracy: 0.9690
Epoch 46/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0254 - accuracy:
0.9887 - val_loss: 0.2797 - val_accuracy: 0.9690
Epoch 47/100
31/31 [==============================] - 0s 6ms/step - loss: 0.0257 - accuracy:
0.9877 - val_loss: 0.2732 - val_accuracy: 0.9690
```

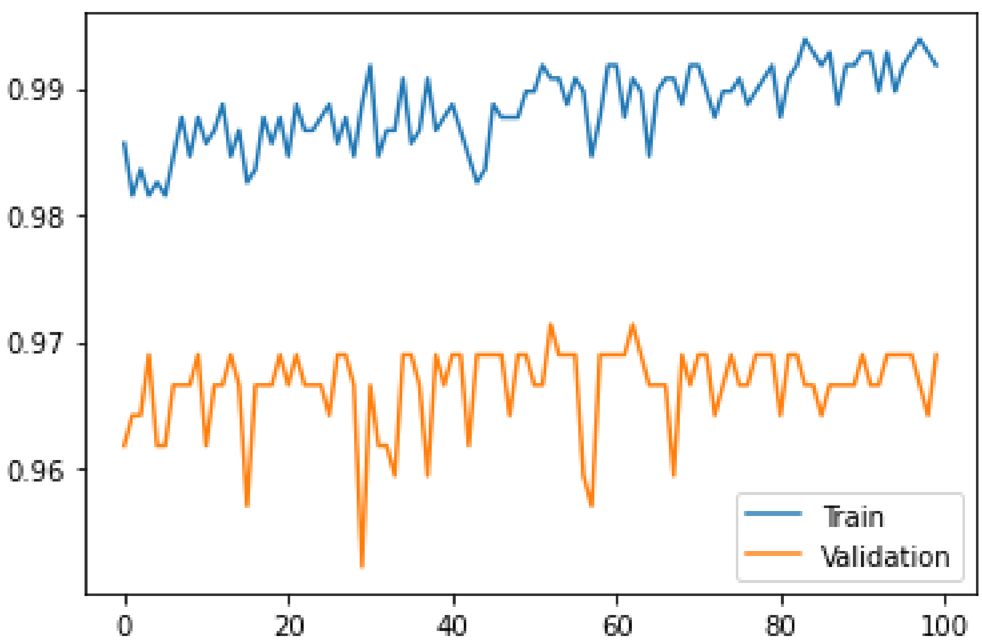
```
Epoch 48/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0219 - accuracy:
0.9877 - val_loss: 0.3509 - val_accuracy: 0.9642
Epoch 49/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0232 - accuracy:
0.9877 - val_loss: 0.2977 - val_accuracy: 0.9690
Epoch 50/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0198 - accuracy:
0.9897 - val_loss: 0.2973 - val_accuracy: 0.9690
Epoch 51/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0218 - accuracy:
0.9897 - val_loss: 0.2868 - val_accuracy: 0.9666
Epoch 52/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0211 - accuracy:
0.9918 - val_loss: 0.3129 - val_accuracy: 0.9666
Epoch 53/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0203 - accuracy:
0.9908 - val_loss: 0.2816 - val_accuracy: 0.9714
Epoch 54/100
31/31 [==============================] - 0s 6ms/step - loss: 0.0201 - accuracy:
0.9908 - val_loss: 0.3311 - val_accuracy: 0.9690
Epoch 55/100
31/31 [==============================] - 0s 6ms/step - loss: 0.0206 - accuracy:
0.9887 - val_loss: 0.3163 - val_accuracy: 0.9690
Epoch 56/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0212 - accuracy:
0.9908 - val_loss: 0.3511 - val_accuracy: 0.9690
Epoch 57/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0251 - accuracy:
0.9897 - val_loss: 0.2776 - val_accuracy: 0.9594
Epoch 58/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0263 - accuracy:
0.9846 - val_loss: 0.2654 - val_accuracy: 0.9570
Epoch 59/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0231 - accuracy:
0.9877 - val_loss: 0.3123 - val_accuracy: 0.9690
Epoch 60/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0201 - accuracy:
0.9918 - val_loss: 0.3024 - val_accuracy: 0.9690
Epoch 61/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0187 - accuracy:
0.9918 - val_loss: 0.3010 - val_accuracy: 0.9690
Epoch 62/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0199 - accuracy:
0.9877 - val_loss: 0.3410 - val_accuracy: 0.9690
Epoch 63/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0179 - accuracy:
0.9908 - val_loss: 0.3004 - val_accuracy: 0.9714
```

```
Epoch 64/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0207 - accuracy:
0.9897 - val_loss: 0.3271 - val_accuracy: 0.9690
Epoch 65/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0256 - accuracy:
0.9846 - val_loss: 0.3503 - val_accuracy: 0.9666
Epoch 66/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0189 - accuracy:
0.9897 - val_loss: 0.2881 - val_accuracy: 0.9666
Epoch 67/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0201 - accuracy:
0.9908 - val_loss: 0.3036 - val_accuracy: 0.9666
Epoch 68/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0196 - accuracy:
0.9908 - val_loss: 0.3107 - val_accuracy: 0.9594
Epoch 69/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0218 - accuracy:
0.9887 - val_loss: 0.3200 - val_accuracy: 0.9690
Epoch 70/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0188 - accuracy:
0.9918 - val_loss: 0.3189 - val_accuracy: 0.9666
Epoch 71/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0176 - accuracy:
0.9918 - val_loss: 0.3121 - val_accuracy: 0.9690
Epoch 72/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0204 - accuracy:
0.9897 - val_loss: 0.3131 - val_accuracy: 0.9690
Epoch 73/100
31/31 [==============================] - 0s 6ms/step - loss: 0.0199 - accuracy:
0.9877 - val_loss: 0.3036 - val_accuracy: 0.9642
Epoch 74/100
31/31 [==============================] - 0s 6ms/step - loss: 0.0188 - accuracy:
0.9897 - val_loss: 0.3031 - val_accuracy: 0.9666
Epoch 75/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0182 - accuracy:
0.9897 - val_loss: 0.3478 - val_accuracy: 0.9690
Epoch 76/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0189 - accuracy:
0.9908 - val_loss: 0.3545 - val_accuracy: 0.9666
Epoch 77/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0194 - accuracy:
0.9887 - val_loss: 0.3314 - val_accuracy: 0.9666
Epoch 78/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0210 - accuracy:
0.9897 - val_loss: 0.3538 - val_accuracy: 0.9690
Epoch 79/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0185 - accuracy:
0.9908 - val_loss: 0.3034 - val_accuracy: 0.9690
```

```
Epoch 80/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0175 - accuracy:
0.9918 - val_loss: 0.3411 - val_accuracy: 0.9690
Epoch 81/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0211 - accuracy:
0.9877 - val_loss: 0.3141 - val_accuracy: 0.9642
Epoch 82/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0185 - accuracy:
0.9908 - val_loss: 0.3459 - val_accuracy: 0.9690
Epoch 83/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0217 - accuracy:
0.9918 - val_loss: 0.2822 - val_accuracy: 0.9690
Epoch 84/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0175 - accuracy:
0.9938 - val_loss: 0.3360 - val_accuracy: 0.9666
Epoch 85/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0168 - accuracy:
0.9928 - val_loss: 0.3429 - val_accuracy: 0.9666
Epoch 86/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0174 - accuracy:
0.9918 - val_loss: 0.3456 - val_accuracy: 0.9642
Epoch 87/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0170 - accuracy:
0.9928 - val_loss: 0.3429 - val_accuracy: 0.9666
Epoch 88/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0193 - accuracy:
0.9887 - val_loss: 0.3526 - val_accuracy: 0.9666
Epoch 89/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0169 - accuracy:
0.9918 - val_loss: 0.3787 - val_accuracy: 0.9666
Epoch 90/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0164 - accuracy:
0.9918 - val_loss: 0.3451 - val_accuracy: 0.9666
Epoch 91/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0187 - accuracy:
0.9928 - val_loss: 0.3388 - val_accuracy: 0.9690
Epoch 92/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0171 - accuracy:
0.9928 - val_loss: 0.3367 - val_accuracy: 0.9666
Epoch 93/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0172 - accuracy:
0.9897 - val_loss: 0.3487 - val_accuracy: 0.9666
Epoch 94/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0161 - accuracy:
0.9928 - val_loss: 0.3567 - val_accuracy: 0.9690
Epoch 95/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0196 - accuracy:
0.9897 - val_loss: 0.3668 - val_accuracy: 0.9690
```

```
Epoch 96/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0150 - accuracy:
0.9918 - val_loss: 0.3185 - val_accuracy: 0.9690
Epoch 97/100
31/31 [==============================] - 0s 5ms/step - loss: 0.0189 - accuracy:
0.9928 - val_loss: 0.3432 - val_accuracy: 0.9690
Epoch 98/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0169 - accuracy:
0.9938 - val_loss: 0.3599 - val_accuracy: 0.9666
Epoch 99/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0165 - accuracy:
0.9928 - val_loss: 0.3555 - val_accuracy: 0.9642
Epoch 100/100
31/31 [==============================] - 0s 4ms/step - loss: 0.0172 - accuracy:
0.9918 - val_loss: 0.3813 - val_accuracy: 0.9690
```

[516]:
```
plt.plot(History.history['accuracy'], label='Train')
plt.plot(History.history['val_accuracy'], label='Validation')
plt.legend()
plt.show()
```



2.4 Evaluate performance on train and validation set combined

[517]:
```
evaluate = []
for i in range(len(data)):
    evaluate.append(smiles_to_fp(data['smiles'][i]))

evaluate = np.array(evaluate)
Model.evaluate(evaluate,data['active'].values)

Preds = Model(evaluate).numpy().round()
print("Predicted amount of active comps:",Model(evaluate).numpy().round().sum())
print("Actual amount of active comps:", data['active'].values.sum())
Preds = Preds.astype(bool)
Pos = data[Preds]
Neg = data[~Preds]
```

```
63/63 [==============================] - 0s 2ms/step - loss: 0.1845 - accuracy:
0.9829
Predicted amount of active comps: 47.0
Actual amount of active comps: 71
```

[518]:
```
print("True Positives: ", Pos['active'].sum())
print("False Positives: ", len(Pos)-Pos['active'].sum())
print("True Negatives: ", len(Neg)-Neg['active'].sum())
print("False Negatives: ", Neg['active'].sum())
```

```
True Positives:  42
False Positives:  5
True Negatives:  1916
False Negatives:  29
```

2.5 Save Model

[519]:
```
Model.save(DATA/"Activity_prediction.h5")
```