IQB Assignment - 3 : Prediction of ATP interacting residues

Aditya Saini, 2018125, ECE

Objective

The objective of this assignment was to detect the presence of ATP interacting residues in a given protein sequence. **The main references of this assignment are the videos uploaded by Sir on Google classroom and this research paper** (https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-10-434).

Directory structure

The structure of the project directory is as follows:

- **Assignment_notebook.ipynb :** Jupyter notebook implementation of the Assignment
- **Assignment_code.py :** Python code for the assignment
- **Report.pdf :** A pdf version of *Assignment_notebook.ipynb*. Contains information regarding usage of code, implementation details, etc.
- **train.data :** Data used for creation of dataset
- **test1.txt :** Test data
- **out_new.csv :** Output data used for submission on Kaggle

Usage

Any Python 3.x version will work for the implementation.

> For running the script: `python Assignment_code.py`

Workflow

The workflow for this assignment is as follows:

In [0]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from tqdm.auto import tqdm
```

Sec-1 : Extracting data out of train.data to create a viable dataset

In [0]:

```
data = pd.read_csv('train.data')
```

In [3]:

```
data
```

Out[3]:

	Protein_ID	Amino Acid Sequence
0	>1A0I_A	VNIKTNPfkaVSFVESAIKKALDNAGYLIAeikyDGVrGNICVDNT...
1	>1A82_A	SKRYFVTGTDtevgktvASCALLQAAKAAGYRTAGYkPVASGSEKT...
2	>1ATP_E	GNAAAAKKGSEQESVKEFLAKAKEDFLKKWETPSQNTAQLDQFDRI...
3	>1AYL_A	MRVNNGLTPQELEAYGISDVHDIVYNPSYDLLYQEELDPSLTGYER...
4	>1B0U_A	MMSENKLHVIDLHKRyGGhEvLKGVSLQARAGDVISIIGssgsgks...
...	...	...
135	>3C4W_B	MDFGSLETVVANSAFIAARGSFDASSGPASRDRKYLARLKLPPLSK...
136	>3C5E_A	MGHHHHHHSSGVDLGTENLYFQSMSLQWGHQEVPAKFNFASDVLDH...
137	>3C9R_A	MGSHHHHHHDITSLYKKAGSAAAVLEENLYFQGSFTMRLKELGEFG...
138	>3R1R_C	MNQNLLvTkrDGSTerinLDkiHRvLDWAAEGLHNVSISQVELRSH...
139	>4AT1_D	MTHDNKLGveaiKRGTvIdhIPAQIGFKLLSLFKLTETDQRITIGL...

140 rows × 2 columns

Sec-2 : Creating a dataset from extracted data

In [0]:

```
sequence_data = data['Amino Acid Sequence']
```

In [5]:

```
sequence_data
```

Out[5]:

```
0      VNIKTNPfkaVSFVESAIKKALDNAGYLIAeikyDGVrGNICVDNT...
1      SKRYFVTGTDtevgktvASCALLQAAKAAGYRTAGYkPVASGSEKT...
2      GNAAAAKKGSEQESVKEFLAKAKEDFLKKWETPSQNTAQLDQFDRI...
3      MRVNNGLTPQELEAYGISDVHDIVYNPSYDLLYQEELDPSLTGYER...
4      MMSENKLHVIDLHKRyGGhEvLKGVSLQARAGDVISIIGssgsgks...
                              ...
135    MDFGSLETVVANSAFIAARGSFDASSGPASRDRKYLARLKLPPLSK...
136    MGHHHHHHSSGVDLGTENLYFQSMSLQWGHQEVPAKFNFASDVLDH...
137    MGSHHHHHHDITSLYKKAGSAAAVLEENLYFQGSFTMRLKELGEFG...
138    MNQNLLvTkrDGSTerinLDkiHRvLDWAAEGLHNVSISQVELRSH...
139    MTHDNKLGveaiKRGTvIdhIPAQIGFKLLSLFKLTETDQRITIGL...
Name: Amino Acid Sequence, Length: 140, dtype: object
```

Sec-2.1 : Generating patterns of given sequences

In [0]:

```
def generate_pattern(seq_data, window_size):
    dummy_variable_length = int((window_size-1)/2)
    ans = [] #Will hold the different patterns for a given amino acid sequence

    for sequence in seq_data:
        #Adding dummy variables to the extreme ends of the string
        string = "X" * dummy_variable_length + sequence + "X" * dummy_variable_length

        #Generating the patterns of size = "window_size"

        for idx in range(0, len(string) + 1 - window_size):
            ans.append(string[idx : idx + window_size])

    return pd.Series(ans)
```

Referring the research paper and from cross-validation, I set the window-size to 17, for achieving the best results.

In [0]:

```
size = 17
pattern_data = generate_pattern(sequence_data, size)
```

In [10]:

```
pattern_data
```

Out[10]:

```
0        XXXXXXXXVNIKTNPfk
1        XXXXXXXVNIKTNPfka
2        XXXXXXVNIKTNPfkaV
3        XXXXXVNIKTNPfkaVS
4        XXXXVNIKTNPfkaVSF
               ...
49302    CEKEFSHNVVLANXXXX
49303    EKEFSHNVVLANXXXXX
49304    KEFSHNVVLANXXXXXX
49305    EFSHNVVLANXXXXXXX
49306    FSHNVVLANXXXXXXXX
Length: 49307, dtype: object
```

Sec-2.2 : Creating labels for our data

- If the middle element was **lower case**, a **+1** was assigned to the sequence, denoting that the given **residue is an ATP interacting residue**
- If the middle element was **upper case**, a **-1** was assigned to the sequence, denoting that the given **residue is NOT an ATP interacting residue**

In [0]:

```
def label_data(pat_data):
    ans = []
    for protein_seq in pat_data:
        target = protein_seq[int(len(protein_seq)/2)]
        if target.islower():
            ans.append(1)
        elif target.isupper():
            ans.append(-1)
    return pd.Series(ans)
```

In [0]:

```
Y_data = label_data(pattern_data)
```

In [13]:

```
Y_data
```

Out[13]:

```
0        -1
1        -1
2        -1
3        -1
4        -1
         ..
49302    -1
49303    -1
49304    -1
49305    -1
49306    -1
Length: 49307, dtype: int64
```

Sec-2.3 : Creating a binary profile for our generated patterns

Each pattern sequence will be matched to a vector sequence, consisting of amino acids and, a 17*21 length vector will be generated. This sequence is a ***reshaped (or flattened)*** binary matrix which will help us in representing our patterns quantitatively.

In [0]:

```
def generate_binaryProfile(pat_data):
    amino_acid = ['A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V','
    ans = []
    for series in tqdm(pat_data):
        ans.append([])
        for acid_1 in series:
            for acid_2 in amino_acid:
                if(acid_1.upper() == acid_2):
                    ans[-1].append(1)
                else:
                    ans[-1].append(0)

    return pd.Series(ans)
```

In [15]:

```
X_data = generate_binaryProfile(pattern_data)
```

```
HBox(children=(IntProgress(value=0, max=49307), HTML(value='')))
```

In [16]:

```
X_data
```

Out[16]:

```
0        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
2        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
3        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
                                ...
49302    [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
49303    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
49304    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, ...
49305    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
49306    [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
Length: 49307, dtype: object
```

Sec-3 : Applying Machine Learning techniques to dataset

We chose SVC (Support Vector Classifier) as our training algorithm as it is best suited for binary class classification problem on huge datasets.

In [0]:

```
from sklearn.svm import SVC
model = SVC(kernel = 'rbf', C = 2, gamma = 0.1)
```

Sec-3.1 : Splitting into training and testing sets

In [0]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X_data.tolist(), Y_data.tolist(), test_
```

Sec-3.2 : Fitting our data on the training sets

In [19]:

```
model.fit(X_train, Y_train)
```

Out[19]:

```
SVC(C=2, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Sec-3.3 : Predicting the test data

In [0]:

```
Y_test_predict = model.predict(X_test)
```

Sec-3.4 : Evaluating our model by calculating the area under ROC curve

In [21]:

```
from sklearn.metrics import roc_auc_score
roc_auc_score(Y_test, Y_test_predict.tolist())
```

Out[21]:

```
0.5315259522502983
```

In [23]:

```
Y_test_predict
```

Out[23]:

```
array([-1, -1, -1, ..., -1, -1, -1])
```

Sec-4 : Generating output for submission

In [0]:

```
pd.read_csv('test1.txt')
X_predict_data = pd.read_csv('test1.txt')['Lable'].tolist()
X_predict_string = ''.join(map(str, X_predict_data))
```

In [0]:

```
X_predict_pattern_data = generate_pattern([X_predict_string], size)
```

In [29]:

```
X_predict_pattern_data
```

Out[29]:

```
0        XXXXXXXXAASSLDELV
1        XXXXXXXAASSLDELVA
2        XXXXXXAASSLDELVAL
3        XXXXXAASSLDELVALC
4        XXXXAASSLDELVALCK
               ...
13018    IQWKYREPKDRSEXXXX
13019    QWKYREPKDRSEXXXXX
13020    WKYREPKDRSEXXXXXX
13021    KYREPKDRSEXXXXXXX
13022    YREPKDRSEXXXXXXXX
Length: 13023, dtype: object
```

In [30]:

```
X_predict = generate_binaryProfile(X_predict_pattern_data)
```

```
HBox(children=(IntProgress(value=0, max=13023), HTML(value='')))
```

In [0]:

```
Y_predict = model.predict(X_predict.tolist())
```

In [32]:

```
pd.Series(Y_predict)
```

Out[32]:

```
0       -1
1       -1
2       -1
3       -1
4       -1
        ..
13018   -1
13019   -1
13020   -1
13021   -1
13022   -1
Length: 13023, dtype: int64
```

```
In [33]:
```

```
output = { 'ID': pd.read_csv('test1.txt')['ID'], 'Lable': pd.Series(Y_predict) }
output = pd.DataFrame(output)
output
```

```
Out[33]:
```

	ID	Lable
0	10001	-1
1	10002	-1
2	10003	-1
3	10004	-1
4	10005	-1
...	...	...
13018	23019	-1
13019	23020	-1
13020	23021	-1
13021	23022	-1
13022	23023	-1

13023 rows × 2 columns

```
In [0]:
```

```
output.to_csv('out_new.csv', index=False)
```