BIOINFORMATIKA:
PREDIKSI STRUKTUR SEKUNDER PROTEIN MENGGUNAKAN SUPPORT VECTOR MACHINE PADA DATASET RS126

Muhammad Ilham Ibadurrohman

14/366093/PA/16211

PROGRAM STUDI ILMU KOMPUTER
DEPARTEMEN ILMU KOMPUTER DAN ELEKTRONIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS GADJAH MADA
2019

GitHub Link: https://github.com/nopedude/SVM_PSSP/tree/master/SVM_PSSP

PENGOLAHAN DATA :

Dataset RS126 dibuka dan diolah dengan menggunakan bahasa pemrograman python. dari dataset tersebut, line ganjil merujuk pada struktur primer protein dan line genap merujuk pada struktur sekunder dari struktur primer.

```
In [1]: import numpy as np

In [2]: raw_primer = []
        raw_sekunder = []

        with open('RS126.txt', 'r') as f:
            for count, line in enumerate(f, start=1):
                if count % 2 == 0:
                    raw_sekunder.append(line.strip())
                else:
                    raw_primer.append(line.strip())
```

Contoh pembacaan data struktur primer:

```
In [3]: raw_primer

Out[3]: ['APAFSVSPASGASDGQSVSVSVAAAGETYYIAQCAPVGGQDACNPATATSFTTDASGAASFSFTVRKSYAGQTPS
         GTPVGSVDCATDACNLGAGNSGLNLGHVALTFG',
          'CSVDIQGNDQMQFNTNAITVDKSCKQFTVNLSHPGNLPKNVMGHNWVLSTAADMQGVVTDGMASGLDKDYLKPDD
         SRVIAHTKLIGSGEKDSVTFDVSKLKEGEQYMFFCTFPGHSALMKGTLTLK',
          'NVYHDGACPEVKPVDNFDWSNYHGKWWEVAKYPNSVEKYGKCGWAEYTPEGKSVKVSNYHVIHGKEYFIEGTAYP
         VGDSKIGKIYHKLTYGGVTKENVFNVLSTDNKNYIIGYYCKYDEDKKGHQDFVWVLSRSKVLTGEAKTAVENYLIGS
         PVVDSQKLVYSDFSEAACKVN',
          'AAPCFCSGKPGRGDLWILRGTCPGGYGYTSNCYKWPNICCYPH',
          'SISQQTVWNQMATVRTPLNFDSSKQSFCQFSVDLLGGGISVDKTGDWITLVQNSPISNLLRVAAWKKGCLMVKVV
         MSGNAAVKRSDWASLVQVFLTNSNSTEHFDACRWTKSEPHSWELIFPIEVCGPNNGFEMWSSEWANQTSWHLSFLVD
         NPKQSTTFDVLLGISQNFEIAGNTLMPAFSVPQ',
          'METNLFKLSLDDVETPKGSMLDLKISQSKIALPKNTVGGTILRSDLLANFLTEGNFRASVDLQRTHRIKGMIKMV
```

Contoh pembacaan data struktur sekunder:

```
In [4]: raw_sekunder

Out[4]: ['CCEEEEECCCCCCCCCCEEEEEECCCCEEEEEEECEECCEECCCCCCCCCEEECCCCCCCCEEEEECCCEEEECCC
         CCEEEEEECCCCCCEEEEECCCCCCCCCCCCCC',
          'CCEEEECCCCCCCCCCCEECCCCCCEEEEEEECCCCCCCCCCCECCEEEECCCCHHHHHHHHHHCHHHHCCCCCC
         CCCCECCCCECCCCCEEEEEECCCCCCCCCEEEECCCCCCCCCCCEEEEECC',
          'CEEEECCCCCCCCCCCCCHHHCCEEEEEEEECCCCCCCCCEEEEEEEEECCCCEEEEEEEECCEEEEEEEEEE
         CCCCCCCEEEEEEEECCEEEEEEEEEECCCCEEEEEEEEECCCCEEEEEEEEECCCCCCHHHHHHHHHHHHC
         CCCCHHHCEECCCCHHHHCCC',
          'CCCCCCCCCCCCCCEEECCCCCCCCCCCCCCEEEECCEEEECC',
          'CCCCCCCCEEEEEEECCCCCCCCCCCEEEEEEECCCCEEECCCCCCCEEEECCHHHHHHHHECCEEEEEEEEEE
         EECCCCCCHHHCCCCEEEEECCCCCCCCCCEEEEECCCCEEEEEEEECCCCCCCECCCCCCCCCCCEEEEEE
         CCCCCCEEEEEEECCCCEECCEEECCCEECC',
          'CCCCCCCCCCCCCCCCCCCCCCCEEEEEEEECCCCCCCEEEEEEHHHHHCCCCCCHHHHHCCCCECCCEEEE
         EECCCCCCCCCEEEEEEECEECCCCCCCCCCCCCEEEECCCCCCCCEEEECCCCCCCCECHHHHHCCCCEEEEEE
         EECCCCCCCCCEEEEEEEECCCCCCCCCEECCCCCCCEEEEEEEEEECCECCCCCEEEECCCCCCEEECCEEEC
         CHHHHCHHHCCEEEEEEEEEECCCCCCECCEEEEECCCCCCCCHHHHHCCEEECCCCCCCEEEEECHHHCCC
```

PENCARIAN DATA YANG TIDAK LENGKAP:

Pengecekan terhadap data dilakukan untuk melihat apakah terdapat perbedaan jumlah karakter struktur primer dan sekunder (karena prediksi struktur sekunder termasuk ke dalam permasalahan sequence labelling, maka dipastikan jumlah karakter dari struktur primer dan struktur sekunder selalu sama).

```
In [5]: for i in range(len(raw_sekunder)):
            len1 = len(raw_sekunder[i])
            len2 = len(raw_primer[i])

            if(len1 != len2):
                print(i," ", raw_sekunder[i]," ",raw_primer[i])
```

```
109   CCCCCECCCCCCCCECCHHHHHHHHHHHHHHHCCCCCHHHHHHHCCECCCCCCCCCCCCCCEEEECCCEEEEEH
HHHHHHHHHHHHHCCCCCCCCEEEECCCCCCECCCHHHHHHHHHHHHHHHHHHHHHHHHHHHCCEEECCCCECHHHHHHH
HCCCCCCEEEECCCCHHHHHHHCCEEEECCHHHHHHHHHCCCEEEECCEEEEEEEECCCCCCCCHHHHHHHHHHHHHHH
HCCCEEEEEECHHHHCCHHHHHHHHHHHHHHCCCEEECCCCHHHHHHHHHCCCHHHHHHHHCCHHHCCCCCCCCCCEE
EEEECCCCCCECECCCCCCCCCEHHHHHHHHHHHCCCCEEEEEEEECCCECCHHHHHHHHHHHHHHHCCCCCCCCEEEE
ECCEHHHHHHHHHCCHHHHHHCCEEECCCCCHHHCCCECCCCCCCCCCEEEEECCCCCCCCCCCCCCCEEEEECCHHH
HHHHHHCECCCCCCCCCEECCCCCEECCCCCCCCCCCCCCCCCCCCCEECCCCCCCCCCCCCCCCCCCECCCCCCCCC
CCCCEEEEEEEEECCCCEHHHHECCHHHHHHCECHHHHHHHECCCCEECCCCEECCEECCCCCCEECHHHHHHHHCC
CCCEEEECCCCECECCCCCHHHHHHHHCCEEEEEECCECHHHHHHHHHCCCEEEEECCHHHHCCCCCCEEEEECHHHC
CCCCCEEEEEECCCCCEEEEEEECCCCHHHHHHHHHCCHHHHHHCCCC   RAKVAMSHFEPHEYIRYDLLEKNIDIV
RKRLNRPLTLSEKIVYGHLDDPANQEIERGKTYLRLRPDRVAMQDATAQMAMLQFISSGLPKVAVPSTIHCDHLIEAQL
```

Setelah dilakukan pengecekan, ditemukan bahwa data pada urutan urutan 109 memiliki panjang struktur primer dan sekunder yang berbeda. Karena hanya ada 1 data maka data struktur primer dan struktur sekunder dari urutan ke 109 tersebut kita hapus dengan cara melakukan operasi pop.

```
In [6]: raw_primer.pop(109)
        raw_sekunder.pop(109)

        count_sekunder = 0
        count_primer = 0
        for i in range(len(raw_sekunder)):
            len1 = len(raw_sekunder[i])
            len2 = len(raw_primer[i])
            count_sekunder = count_sekunder + len1
            count_primer = count_primer + len2
            if(len1 != len2):
                print(i," ", raw_sekunder[i]," ",raw_primer[i])

        print("count struktur sekunder : ",count_sekunder)
        print("count struktur primer : ",count_primer)
```

```
count struktur sekunder :  22594
count struktur primer :  22594
```

ORTHOGONAL ENCODING:

Target Labeling Untuk setiap data struktur primer dan sekunder dilakukan split sehingga dapat diencode kedalam bentuk orthogonal.

```
In [7]: def split(sequence):
            return [char for char in sequence]

In [8]: split_primer = []
        split_sekunder = []
        for i in range(len(raw_primer)):
            split_primer.append(split(raw_primer[i]))
            split_sekunder.append(split(raw_sekunder[i]))
```

Hasil operasi split struktur primer dan struktur sekunder protein kemudian diubah kedalam bentuk Orthogonal Encoding dan Target Labeling. Cuplikan switch case untuk setiap asam amino pada struktur primer protein ditampilkan sebagai berikut:

```
In [9]: def orthogonal_primer(arg):
            switch = {
                'A' : np.array([1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]),
                'C' : np.array([0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]),
                'E' : np.array([0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]),
                'D' : np.array([0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]),
                'G' : np.array([0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]),
                'F' : np.array([0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]),
                'I' : np.array([0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0]),
                'H' : np.array([0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0]),
                'K' : np.array([0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0]),
                'M' : np.array([0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0]),
                'L' : np.array([0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0]),
```

```
        'L' : np.array([0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0]),
        'N' : np.array([0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0]),
        'Q' : np.array([0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0]),
        'P' : np.array([0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0]),
        'S' : np.array([0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0]),
        'R' : np.array([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0]),
        'T' : np.array([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0]),
        'W' : np.array([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0]),
        'V' : np.array([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]),
        'Y' : np.array([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1])
    }

    return switch.get(arg)

def orthogonal_sekunder(arg):
    switch = {
        'H' : 0,
        'C' : 1,
        'E' : 2
    }

    return switch.get(arg)
```

In [10]:
```
for i in range(len(split_primer)):
    seq = split_primer[i]
    for j in range(len(seq)):
        seq[j] = orthogonal_primer(seq[j])
```

In [11]:
```
split_primer
```

Out[11]:
```
[[array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
  array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]),
  array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
  array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
  array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]),
  array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]),
  array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]),
  array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]),
  array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
  array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]),
```

In [12]:
```
for i in range(len(split_sekunder)):
    seq2 = split_sekunder[i]
    for j in range(len(seq2)):
        seq2[j] = orthogonal_sekunder(seq2[j])
```

In [13]:
```
split_sekunder
```

Out[13]:
```
[[1,
  1,
  2,
  2,
  2,
  2,
  2,
  1,
  1,
  1,
  1,
```

```
In [14]: def target(lis):
             Y = []
             for i in range(len(lis)):
                 for j  in range(len(lis[i])):
                     Y.append(lis[i][j])
             return Y

In [15]: y_label = target(split_sekunder)
         y_label

Out[15]: [1,
          1,
          2,
          2,
          2,
          2,
          2,
          1,
          1,
```

DATASET

Label dan Fitur Setiap hasil split dari struktur sekunder protein akan dijadikan label/target class. Dari hasil operasi tersebut didapatkan total keseluruhan label berjumlah 22.594. Angka ini akan digunakan untuk memastikan bahwa dataset juga memiliki jumlah fitur yang sama.

```
In [16]: len(y_label)

Out[16]: 22594
```

Pembentukan fitur data dilakukan dengan menggunakan fungsi window_padding_data. Fungsi ini akan menerima ukuran dari sliding window dan sekuens struktur primer protein. Pada fungsi ini, fitur akan diolah seperti penambahan padding 0 di awal dan diakhir dan pengambilan fitur-fitur hasil windowing sehingga data keluaran bisa langsung dilatih pada model SVM.

```
In [17]: def window_padding_data(size, sequence):
             num = int(size/2)
             zeros = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
             for i in range(len(sequence)):
                 for j in range(num):
                     sequence[i].append(zeros)
                     sequence[i].insert(0, zeros)

             X = []
             temp = []

             for k in range(len(sequence)):
                 for l in range(len(sequence[k])-(size-1)):
                     temp = sequence[k][l:l+size]
                     X.append(temp)
                     temp = []

             return X
```

Contoh penggunaan fungsi padding :

```
In [18]: X = window_padding_data(13,split_primer)

In [19]: len(X)

Out[19]: 22594
```

Dimana data struktur primer protein akan diolah dengan variabel ukuran sliding window SVM. Sebelum dimasukkan ke model SVM Scikit-Learn, data di reshape untuk mengikuti ukuran input dari model.

```
In [20]: X = np.array(X)
         y_label = np.array(y_label)
         X = X.reshape(22594, 13*20)
```

Sebelum data dimasukkan ke model SVM Scikit-Learn, data di reshape untuk mengikuti ukuran input dari model. Data direshape menjadi ukuran 22.594 data dikali dengan 100 (x ukuran window dan 20 ukuran orthogonal encoding.)

```
In [20]: X = np.array(X)
         y_label = np.array(y_label)
         X = X.reshape(22594, 13*20)

In [26]: #Data di split menjadi 2, training dan testing data. Kemudian dihitung dengan SV

In [22]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y_label, test_size = 0.20

In [23]: from sklearn.svm import SVC
         from sklearn.metrics import classification_report

In [24]: svc = SVC(kernel='rbf', gamma = 0.1, C=1.5)
         svc.fit(X_train, y_train)
         y_pred = svc.predict(X_test)
         y_true = y_test

         print(classification_report(y_true,y_pred))
```

HASIL:

Ukuran Window	Classification Report
3	precision recall f1-score support 0 0.54 0.48 0.51 1458 1 0.60 0.76 0.67 2050 2 0.49 0.30 0.37 1011 accuracy 0.57 4519 macro avg 0.54 0.51 0.52 4519 weighted avg 0.56 0.57 0.55 4519
5	precision recall f1-score support 0 0.57 0.53 0.55 1477 1 0.63 0.75 0.68 2047 2 0.51 0.37 0.43 995 accuracy 0.59 4519 macro avg 0.57 0.55 0.55 4519 weighted avg 0.58 0.59 0.58 4519
7	precision recall f1-score support 0 0.59 0.58 0.59 1389 1 0.63 0.73 0.68 2025 2 0.58 0.41 0.48 1105 accuracy 0.61 4519 macro avg 0.60 0.58 0.58 4519 weighted avg 0.61 0.61 0.60 4519
9	precision recall f1-score support 0 0.62 0.57 0.59 1479 1 0.63 0.75 0.68 2006 2 0.54 0.41 0.47 1034 accuracy 0.61 4519 macro avg 0.60 0.57 0.58 4519 weighted avg 0.61 0.61 0.60 4519

		precision	recall	f1-score	support
11	0	0.61	0.57	0.59	1465
	1	0.63	0.75	0.69	2007
	2	0.56	0.41	0.48	1047
	accuracy			0.62	4519
	macro avg	0.60	0.58	0.59	4519
	weighted avg	0.61	0.62	0.61	4519
13		precision	recall	f1-score	support
	0	0.63	0.59	0.61	1492
	1	0.62	0.76	0.69	1974
	2	0.59	0.41	0.48	1053
	accuracy			0.62	4519
	macro avg	0.62	0.59	0.59	4519
	weighted avg	0.62	0.62	0.61	4519
15		precision	recall	f1-score	support
	0	0.60	0.60	0.60	1420
	1	0.65	0.75	0.69	2037
	2	0.57	0.40	0.47	1062
	accuracy			0.62	4519
	macro avg	0.61	0.58	0.59	4519
	weighted avg	0.62	0.62	0.61	4519
17 (Terbaik)		precision	recall	f1-score	support
	0	0.62	0.62	0.62	1421
	1	0.66	0.75	0.70	2052
	2	0.58	0.42	0.49	1046
	accuracy			0.63	4519
	macro avg	0.62	0.60	0.60	4519
	weighted avg	0.63	0.63	0.62	4519
19		precision	recall	f1-score	support
	0	0.63	0.59	0.61	1455
	1	0.64	0.77	0.70	2005
	2	0.61	0.41	0.49	1059
	accuracy			0.63	4519
	macro avg	0.62	0.59	0.60	4519
	weighted avg	0.63	0.63	0.62	4519