# Protein sequence classification

September 22, 2020

Protein sequence classification

Miradain Atontsa Nguemo (miradain.atontsan@gmail.com)

https://miradain.github.io

## 1 Introduction

The goal of this analysis is to solve an unsupervised classification problem on a genomic data. The data consists of shuffled protein sequences of the form

GLSDGEWQQVLNVWGKVEADIAGHGQEVLIRLFTGHPETLEKFDKF...

We will perform a clustering analysis to separate them into several protein families.

## 2 Methodology

We address the problem in five major steps:

- We will extract from each protein sequence numerical information based on the biology and the configuration of amino acids present in the sequence. In other words, we build a new data set with only numerical features constructed from protein sequences;
- We will next analyse these numerical features in order to retain those features which are likely to change between families. The idea behind this approach is that, if a feature defers less over proteins, then it is less likely to differentiate them;
- Furthermore, we have to choose an algorithm able to build the protein families. This is not an easy task since all algorithms used today require some entry parameter(s) which directly or indirectly influence the number of clusters. On the other hand, we cannot plot our high dimensional data set for visualization and decide the number of clusters. To handle this part, we will use a topological data analysis tool called "persistent diagram" which gives us some insights on the configuration of points in space. This tool is robust to noises and will give the number of clusters to consider.
- The algorithm chosen for clustering is the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm. The input parameters of this algorithm will be based on the analysis of the persistent diagram.

- Finally, we will visualize a member from each protein family. For a given protein sequence, we will plot a network graph where the nodes are amino acids and edges are interaction between amino acids in the sequence. This approach will permit us to see how different families are in terms of the density (number in exchanges between amino acids).

# 3 Libraries

## 3.1 Installation

- pip install biopython
- pip install pandas
- pip install numpy
- pip install matplotlib
- pip install -U scikit-learn
- pip install ripser

- pip install networkx
- pip install persim

## 3.2 Loading the libraries

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics.pairwise import pairwise_distances
from sklearn.cluster import  DBSCAN
from Bio.SeqUtils.ProtParam import ProteinAnalysis
from sklearn.preprocessing import StandardScaler
import networkx as nx  # to draw graphs
from ripser import Rips   # to compute the topological diagrams
from persim import plot_diagrams  # to plot the topological diagrams
```

# 4 Loading the data

[3]:
```
data=pd.read_csv('/Users/miradain/Documents/Genomic analysis/
 ↪test_data_scientist_shuffled_sequences.csv')
proteins=data['original_sequence']
```

## 5 Exploratory analysis

- Plot the data

[4]:
```
print(data.head(5))
print('The data has',len(data), 'sequences.')
```

```
   id                                   original_sequence
0   0  GLSDGEWQQVLNVWGKVEADIAGHGQEVLIRLFTGHPETLEKFDKF…
1   1  QSVLTQPPSVSGAPGQRVTISCTGSRSNMGAGYDVHWYQLLPGAAP…
2   2  VQWSAEEKQLISSLWGKVNVAECGAEALARLLIVYPWTQRFFTSFG…
3   3  MHGQVDSSPGIWQLDCTHLEGKVILVAVHVASGYIEAEVIPAETGQ…
4   4  MHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPHTQRFFESFG…
The data has 411 sequences.
```

- Unambiguous letters for proteins

We know that some letters in a protein sequence might not refer to amino acids. These are 'B', 'Z', 'X', 'J'. We will check in our code if there are proteins having one of these letters. The code below describes the "unambigous_checker" which is function designed to check unambiguous letters on protein sequences in the data.

[5]:
```
#The function returns 0 is there is an unambigous letter, and 1 otherwise.
def unambigous_checker(protein):
    Unambiguous_Letters=['B', 'Z', 'X', 'J']# unvalid unambiguous letter for␣
 ↪protein
    Letters=list(protein)# get the letters from the protein
    dic={}
    checker=1
    for acid in Letters:
        if acid in Unambiguous_Letters:
            dic[acid]=0
        else:
            dic[acid]=1

        checker= checker*dic[acid]
    return checker


# Test on the data
N=len(data)
check=np.zeros(N)
for i in range(N):
    check[i]=unambigous_checker(proteins[i])

print('There is exactly ', N-sum(check), 'protein(s) with at least one␣
 ↪unambigous letter')
```

```
There is exactly  1.0 protein(s) with at least one unambigous letter
```

# 6 Feature extraction from protein parametric information

We extract features by analysing different properties of each protein of the data. These are: amino acid frequences per proteins, molecular weight, aromaticity, instability_index, flexibility, isoelectric point, secondary structure fraction, and the protein scaling. We use the methods defined in the ProtParam module of Biopython.

### 6.0.1 Feature extraction functions

- Amino acid features

Given any amino acid, we associate a numerical feature which counts its appearance on each protein sequence.

[7]:
```
def amino_acid_features(data):
    N=len(data)
    # Creating the alphabet
    Alphabet=list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
    # Create feature names
    Acid_Features=[Alphabet[i]+'_feature' for i in range(len(Alphabet))]
    # Initialize the dataframe with amino acid features
    matrix=np.zeros((N, 26))
    df=pd.DataFrame(matrix, columns=Alphabet)
    for i in range(N):
        my_seq=data[i]
        analysed_seq = ProteinAnalysis(my_seq)
        count= analysed_seq.count_amino_acids()# Count the amino acids
        Names=[name for name in count] # Get the amino acids present in the␣
↪protein
        for acid in Names:
            df[acid][i]=count[acid]
    # Renaming dataframe columns
    df.columns=Acid_Features
    return df

# Test on the data
df_Amino_acid=amino_acid_features(proteins)
print(df_Amino_acid)
```

```
     A_feature  B_feature  C_feature  D_feature  E_feature  F_feature  \
0         15.0        0.0        0.0        7.0       12.0        7.0
1         21.0        0.0        5.0        7.0        9.0        4.0
2         18.0        0.0        3.0        5.0        8.0        8.0
3         15.0        0.0        2.0        8.0       11.0        3.0
4         15.0        0.0        2.0        7.0        8.0        8.0
..         …        …        …        …        …        …
406       35.0        0.0        6.0       22.0       22.0        9.0
```

```
407       81.0        0.0       30.0       61.0       51.0       77.0
408       18.0        0.0        0.0        6.0       14.0        6.0
409       14.0        0.0        1.0        7.0       10.0        8.0
410       17.0        0.0        0.0        6.0       14.0        6.0

     G_feature  H_feature  I_feature  J_feature  …  Q_feature  R_feature  \
0         15.0       11.0        9.0        0.0  …        6.0        2.0
1         16.0        6.0        4.0        0.0  …       11.0        5.0
2          7.0        6.0        8.0        0.0  …        6.0        6.0
3         17.0        7.0       15.0        0.0  …       10.0        5.0
4         13.0       10.0        0.0        0.0  …        3.0        3.0
..         …          …          …          …  …         …          …
406       35.0        9.0       24.0        0.0  …       19.0       19.0
407       94.0       26.0       71.0        0.0  …       65.0       42.0
408       11.0       11.0        9.0        0.0  …        5.0        4.0
409       14.0        8.0        0.0        0.0  …        4.0        4.0
410       11.0       12.0        9.0        0.0  …        5.0        4.0

     S_feature  T_feature  U_feature  V_feature  W_feature  X_feature  \
0          5.0        7.0        0.0        7.0        2.0        0.0
1         34.0       18.0        0.0       17.0        3.0        0.0
2          9.0        5.0        0.0       12.0        4.0        0.0
3          7.0       10.0        0.0       14.0        3.0        0.0
4          5.0        7.0        0.0       17.0        1.0        0.0
..         …          …          …          …          …          …
406       19.0       17.0        0.0       27.0        7.0        0.0
407      103.0       95.0        0.0       92.0       12.0        0.0
408        6.0        5.0        0.0        8.0        2.0        0.0
409        6.0        3.0        0.0       18.0        2.0        0.0
410        6.0        5.0        0.0        8.0        3.0        0.0

     Y_feature  Z_feature
0          2.0        0.0
1          8.0        0.0
2          2.0        0.0
3          4.0        0.0
4          3.0        0.0
..         …          …
406        8.0        0.0
407       53.0        0.0
408        3.0        0.0
409        3.0        0.0
410        3.0        0.0

[411 rows x 26 columns]
```

- Molecular weight feature

This feature contains the molecular weight of each protein. That is obtained by multiplying the

subscript of each element (letter) by its atomic weight on the periodic table.

[8]:
```
# Get the molecular weight of the protein
def protein_weight_feature(data):
    N=len(data)
    weight=np.zeros(N)
    for i in range(N):
        my_seq=data[i]
        analysed_seq=ProteinAnalysis(my_seq)
        if unambigous_checker(my_seq)==1: # Check is the data has a protein with␣
 ↪unambigous letter
            weight[i]=analysed_seq.molecular_weight() # get the weight of the␣
 ↪protein
    return weight

# Test on the data
weight=protein_weight_feature(proteins)
print(pd.DataFrame(weight) )
```

```
               0
0     16963.3995
1     22767.0452
2     16299.6811
3     17746.9456
4     15850.0048
..           …
406   39836.8645
407  140697.9114
408   17263.8197
409   15870.9360
410   17402.9334

[411 rows x 1 columns]
```

- Aromaticity feature

We counts on each protein sequence the number (in percentage) of three kinds of aromatic amino acids: Phenylalanine, Tryptophan, Tyrosine. This feature is the sum of the collected values.

[9]:
```
# Get the aromaticity of the protein
def aromaticity_feature(data):
    N=len(data)
    aromaticity=np.zeros(N)
    for i in range(N):
        my_seq=data[i]
        analysed_seq=ProteinAnalysis(my_seq)
        aromaticity[i]= analysed_seq.aromaticity()
    return aromaticity
```

```
# Test on the data
aromaticity= aromaticity_feature(proteins)
print(pd.DataFrame(aromaticity))
```

```
            0
0    0.071895
1    0.069124
2    0.095890
3    0.061350
4    0.082192
..        …
406  0.065934
407  0.111460
408  0.071429
409  0.089655
410  0.077922

[411 rows x 1 columns]
```

- Instability index feature

The instability index provides an estimate of the stability of the protein in a test tube. A protein whose instability index is smaller than 40 is predicted as stable, a value above 40 predicts that the protein may be unstable.

[10]:
```
# Get the instability_index of the protein
def instability_index_Feature(data):
    N=len(data)
    instability_index=np.zeros(N)
    for i in range(N):
        my_seq=data[i]
        analysed_seq=ProteinAnalysis(my_seq)
        if unambigous_checker(my_seq)==1:# Check is the data has a protein with␣
 ↪unambigous letter
            instability_index[i]= analysed_seq.instability_index()
    return instability_index


# Test on the data
instability_index=instability_index_Feature(proteins)
print(pd.DataFrame(instability_index))
```

```
            0
0   10.119608
1   62.039677
2   18.184247
3   39.298773
```

```
4    10.765068
..         …
406  38.246154
407  31.426381
408  14.697403
409  14.311034
410  13.350649

[411 rows x 1 columns]
```

- Flexibility mean feature

Flexibility has been predicted from amino acid sequence with a sliding window averaging technique. This feature contains for each protein sequence the mean of different flexibility values.

[11]:
```
# Get the flexibility mean of the protein
def flexibility_mean_feature(data):
    N=len(data)
    flexibility_mean=np.zeros(N)
    for i in range(N):
        my_seq=data[i]
        analysed_seq=ProteinAnalysis(my_seq)
        if unambigous_checker(my_seq)==1:# Check is the data has a protein with␣
 ↪unambigous letter
            flexibility_mean[i]= np.mean(analysed_seq.flexibility())
    return flexibility_mean



# Test on the data
flexibility=flexibility_mean_feature(proteins)
print(pd.DataFrame(flexibility))
```

```
           0
0    1.005295
1    1.004945
2    0.992723
3    0.999583
4    0.996639
..        …
406  1.006678
407  0.997357
408  1.003393
409  0.997965
410  1.002945

[411 rows x 1 columns]
```

- Isoelectric point feature

We create the isoelectric point on a given protein by estimating the pH value at which the protein will have a net charge of zero, and further determine the pKa value right above and right below the estimated pH and find their average. This corresponds to the isoelectric point (pI value) of the protein.

[12]:
```
# Get the isoelectric point of the protein
def isoelectric_point_feature(data):
    N=len(data)
    isoelectric_point=np.zeros(N)
    for i in range(N):
        my_seq=data[i]
        analysed_seq=ProteinAnalysis(my_seq)
        isoelectric_point[i]= analysed_seq.isoelectric_point()
    return isoelectric_point



# Test on the data
isoelectric_point=isoelectric_point_feature(proteins)
print(pd.DataFrame(isoelectric_point))
```

```
            0
0    9.180185
1    7.149004
2    8.851396
3    6.152332
4    6.786598
..        …
406  9.025397
407  6.142669
408  9.003929
409  6.463412
410  9.004186

[411 rows x 1 columns]
```

- Secondary structure fraction feature

This method returns a list of the fraction of amino acids which tend to be in helix, turn or sheet. - Amino acids in helix: V, I, Y, F, W, L. - Amino acids in turn: N, P, G, S. - Amino acids in sheet: E, M, A, L. We create three features: Helix, Turn and Sheet.

[13]:
```
# Get the secondary structure fraction of the protein
def secondary_structure_fraction_feature(data):
    N=len(data)
    matrix=np.zeros((N, 3))
    df=pd.DataFrame(matrix, columns=['Helix', 'Turn', 'Sheet'])
    for i in range(N):
        my_seq=data[i]
```

```
        analysed_seq=ProteinAnalysis(my_seq)
        triple= analysed_seq.secondary_structure_fraction()
        df['Helix'][i]=triple[0]
        df['Turn'][i]=triple[1]
        df['Sheet'][i]=triple[2]
    return df

# Test on the data
df_secondary_structure_fraction=secondary_structure_fraction_feature(proteins)
print(df_secondary_structure_fraction)
```

```
        Helix      Turn     Sheet
0    0.287582  0.169935  0.300654
1    0.239631  0.327189  0.216590
2    0.349315  0.184932  0.301370
3    0.288344  0.208589  0.226994
4    0.321918  0.212329  0.294521
..        …       …        …
406  0.250000  0.233516  0.217033
407  0.320251  0.272370  0.192308
408  0.298701  0.149351  0.344156
409  0.344828  0.220690  0.303448
410  0.298701  0.149351  0.331169

[411 rows x 3 columns]
```

- Protein scaling feature

The method returns a list of values which can be plotted to view the change along a protein sequence. One can set several parameters that control the computation of a scale profile, such as the window size(=9) and the window edge(=0.4) relative weight value. This feature contains for each protein sequence the mean of different scaling values.

[14]:
```
# Get the protein scale mean of the protein
def protein_scale_mean_feature(data):
    N=len(data)
    protein_scale_mean=np.zeros(N)
    for i in range(N):
        my_seq=data[i]
        analysed_seq=ProteinAnalysis(my_seq)
        if unambigous_checker(my_seq)==1:# Check is the data has a protein with␣
 ↪unambigous letter
            protein_scale_mean[i]= np.mean(analysed_seq.
 ↪protein_scale(analysed_seq.get_amino_acids_percent(), window=9, edge=0.4))
    return protein_scale_mean
```

```
# Test on the data
protein_scale=protein_scale_mean_feature(proteins)
print(pd.DataFrame(protein_scale))
```

```
            0
0    0.077800
1    0.075475
2    0.067625
3    0.065106
4    0.075731
..        …
406  0.062968
407  0.059779
408  0.077081
409  0.078169
410  0.075261

[411 rows x 1 columns]
```

### 6.0.2 Transformed data

[15]:
```
New_data=df_Amino_acid.copy()
New_data['weight']=weight
New_data['aromaticity']=aromaticity
New_data['instability']=instability_index
New_data['flexibility']=flexibility
New_data['isoelectric_point']=isoelectric_point
New_data['Helix']=df_secondary_structure_fraction['Helix']
New_data['Turn']=df_secondary_structure_fraction['Turn']
New_data['Sheet']=df_secondary_structure_fraction['Sheet']
New_data['protein_scale']=protein_scale

print('The new data has', New_data.shape[1], 'features.')
```

```
The new data has 35 features.
```

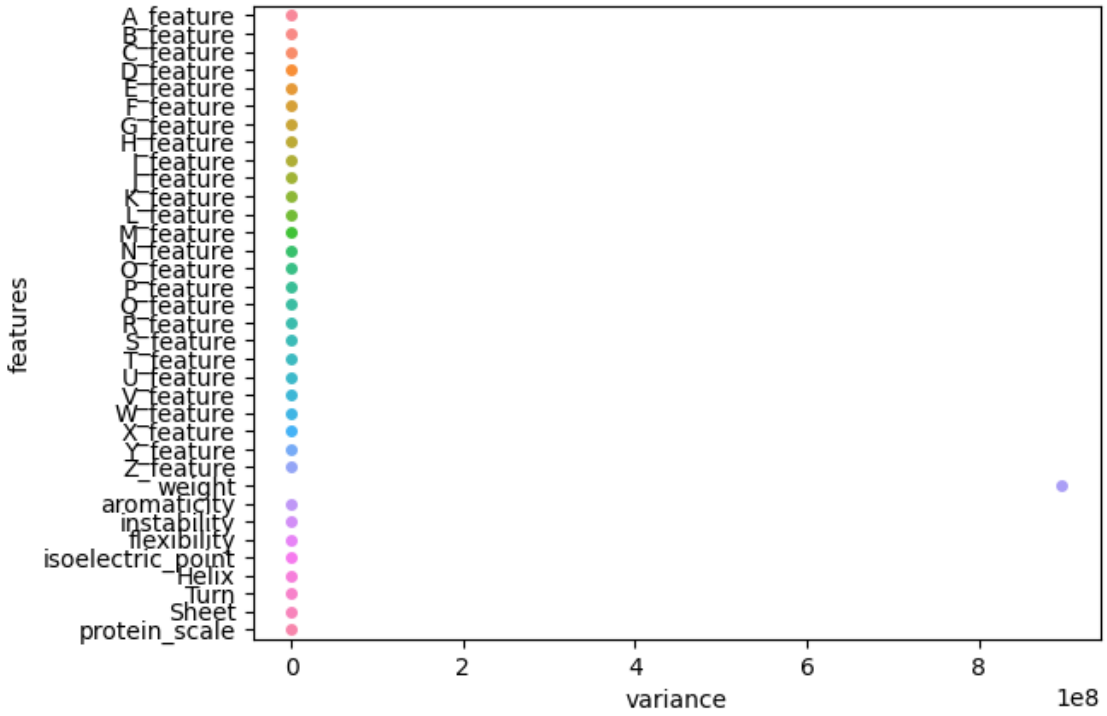
### 6.0.3 Feature selection

In this part, we will remove features with very low variances.

[65]:
```
Covariances=New_data.cov() # Computing the covariance matrix
# Creating a dataframe with feature variances
variances=pd.DataFrame(np.diag(Covariances), index=Covariances.index,␣
 ↪columns=['variance'])
```

```
features=list(variances.index) # Creating a feature list
variances['features']=features # Creating a feature column
sns.swarmplot(x="variance", y="features", data=variances)# Plotting the␣
 ↪variances
```

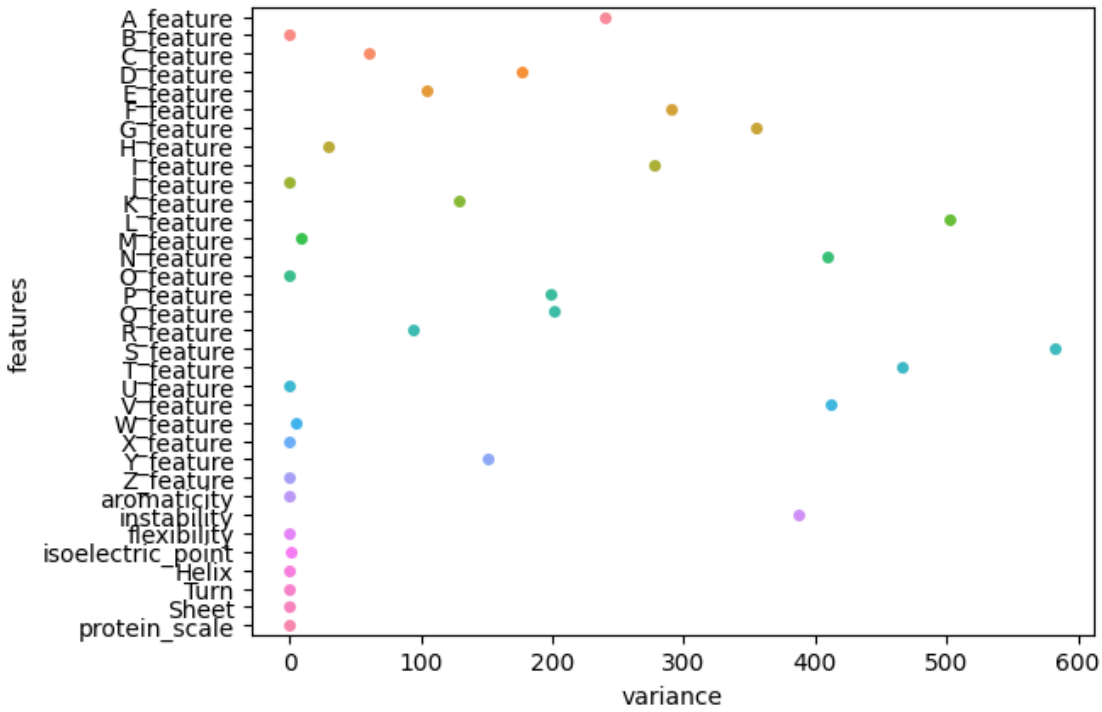
[65]: <matplotlib.axes._subplots.AxesSubplot at 0x120e68ac8>




In this plot, the variable "weight" has the highest variance and his hight value is influencing the visibility of the other features. Therefore, we will drop this feature and plot again the rest of the data.

[69]:
```
df=variances.drop(index='weight')# drop the feature "weight"
sns.swarmplot(x="variance", y="features", data=df)
```

[69]: <matplotlib.axes._subplots.AxesSubplot at 0x12131d2b0>




Many features have very low variances (closed to 0). These are: protein_scale, Sheet, Turn, Aromacity, flexibility, some amino acid features such as: U_feature, O_feature, and finaly the obvious unambigous letters: B_feature, Z_feature, X_feature and J_feature.

We decide from this plot to consider the treshold=0.05. In other words, we will drop features with variances less that 0.05.

[70]:
```
Var=variances[variances.variance>0.05]
New_features=Var.index.values.tolist()
print('The features to consider in our analysis are:',New_features)

Final_data=New_data[New_features]
```

```
The features to consider in our analysis are: ['A_feature', 'C_feature',
'D_feature', 'E_feature', 'F_feature', 'G_feature', 'H_feature', 'I_feature',
'K_feature', 'L_feature', 'M_feature', 'N_feature', 'P_feature', 'Q_feature',
'R_feature', 'S_feature', 'T_feature', 'V_feature', 'W_feature', 'Y_feature',
'weight', 'instability', 'isoelectric_point']
```

### 6.0.4 Protein data clustering

In this part, we will group the proteins using the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm. The idea behind this approach consists of thikening the data points to balls and then explore the density of the point cloud. The main parameters of this

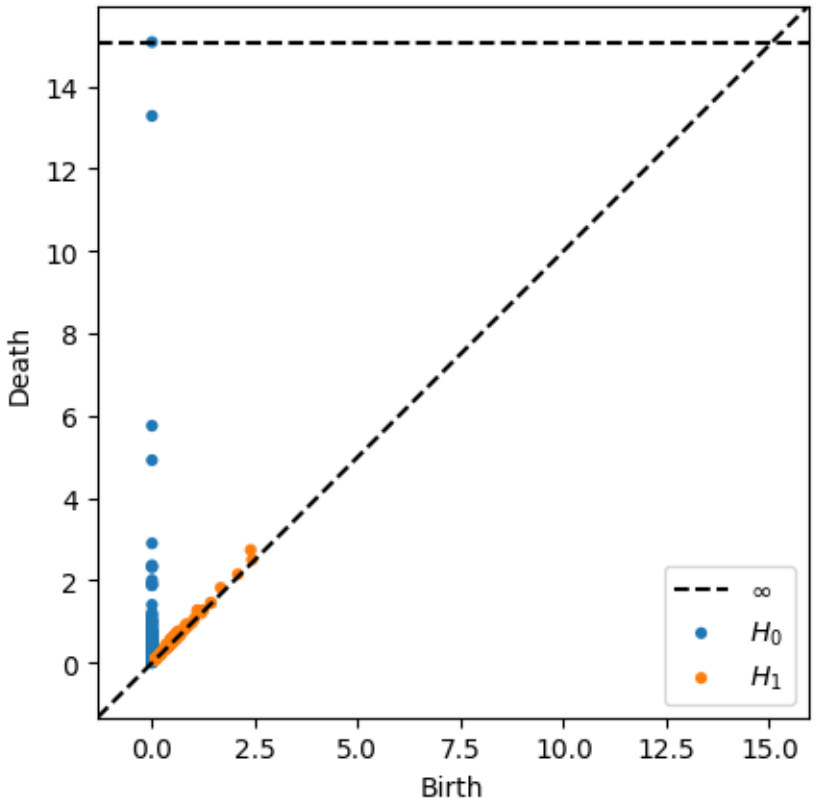
algorithm are: the radius $\varepsilon$ of disks (or sphere) around each point and the minimum number of points required by cluster.

To make the DBSCAN algorithm more efficient, we will use a topological trick which consists of varying the radius $\varepsilon$. This approach is robust to noises and we can observe (through a filtration) when a cluster is created (its birth) and when it merges with another cluster (its death). This will permit to decide (with less bias) the number of cluster to consider.

**Topological filtering** To implement this topological part, we will use the "ripser" library to compute the diagrams of (birth, death) and the "persim" library to plot the diagrams.

[19]:
```
rips=Rips()
scaled_data=StandardScaler().fit_transform(Final_data)
diagrams=rips.fit_transform(scaled_data)
rips.plot(diagrams)
```

```
Rips(maxdim=1, thresh=inf, coeff=2, do_cocycles=False, n_perm = None,
verbose=True)
```




The points in blue represents the clusters and those in orange, that we will not interpret here, represents the holes in the space. The blue points far from the diagonal are clusters which persists

the most over filtration. In other words, we are interested by these points and our treshold will be to keep clusters which persist over Dearth=4 (on the plot). There are exactly 4 clusters.

**Clustering**

[20]:
```
scaled_data=StandardScaler().fit_transform(Final_data)# Scaling the data
db=DBSCAN(eps=4, min_samples=1).fit(scaled_data) # Creating the DBSCAN object␣
 ↪and fitting the data
labels=np.array(db.labels_) # Getting the label out of the model
print('The clusters are labeled in the group:',set(labels))
```

```
The clusters are labeled in the group: {0, 1, 2, 3}
```

**Classified data**

[21]:
```
classified_data=data.copy()
classified_data['labels']=labels
print(classified_data)
```

```
      id                                   original_sequence  labels
0      0  GLSDGEWQQVLNVWGKVEADIAGHGQEVLIRLFTGHPETLEKFDKF…       0
1      1  QSVLTQPPSVSGAPGQRVTISCTGSRSNMGAGYDVHWYQLLPGAAP…       0
2      2  VQWSAEEKQLISSLWGKVNVAECGAEALARLLIVYPWTQRFFTSFG…       0
3      3  MHGQVDSSPGIWQLDCTHLEGKVILVAVHVASGYIEAEVIPAETGQ…       0
4      4  MHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPHTQRFFESFG…       0
..   …                                                …     …
406  406  GSHMPKRGRPAATEVKIPKPRGRPPLPAGTNSKGPPDFSSDEEREP…       0
407  407  CVNLTTRTQLPPAYTNSFTRGVYYPDKVFRSSVLHSTQDLFLPFFS…       1
408  408  MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR…       0
409  409  VQLSGEEKAAVLALWDKVNEEEVGGEALGRLLVVYPWTQRFFDSFG…       0
410  410  MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR…       0

[411 rows x 3 columns]
```

# 7 Visualization of protein families

- Network graph function

To plot a protein sequence, we consider each letter (amino acid) as a node. Then when reading the sequence from the left to the right, we create an edge between every consecutive letters(nodes). In that sense, the graph represents the interaction between each amino acid with the others.

[22]:
```
def Protein_Network_graph(protein):
    G=nx.Graph()# Create the network
    Letters=list(protein)# Extract the letters from the protein
    # set a color to each letter of the alphabet
```

```
    color_map = {'A':'silver', 'B':'rosybrown', 'C':'red','D': 'sienna', 'E':
↪'peru', 'F':'gold', 'G':'orange',
                 'H':'green','I': 'turquoise', 'J':'steelblue','K':'slategray',␣
↪'L':'lightsteelblue', 'M':'blue',
                'N':'navy', 'O':'plum', 'P':'orchid','Q': 'pink', 'R':'violet',␣
↪'S':'salmon','T': 'sienna',
                 'U':'palegreen','V': 'slategray','W': 'blueviolet', 'X':
↪'ivory', 'Y':'lime', 'Z':'cyan'}

    color_letters=[color_map[letter] for letter in set(Letters)] # get the color␣
↪to each letter of the protein
    # adding nodes to the network
    Nodes=list(set(Letters))
    G.add_nodes_from(Nodes)
    # adding edges
    for i in range(len(Letters)-1):
        edge = (Letters[i], Letters[i+1])
        G.add_edge(*edge)
    nx.draw(G, node_color=color_letters, with_labels=True )
    plt.show() # display
```

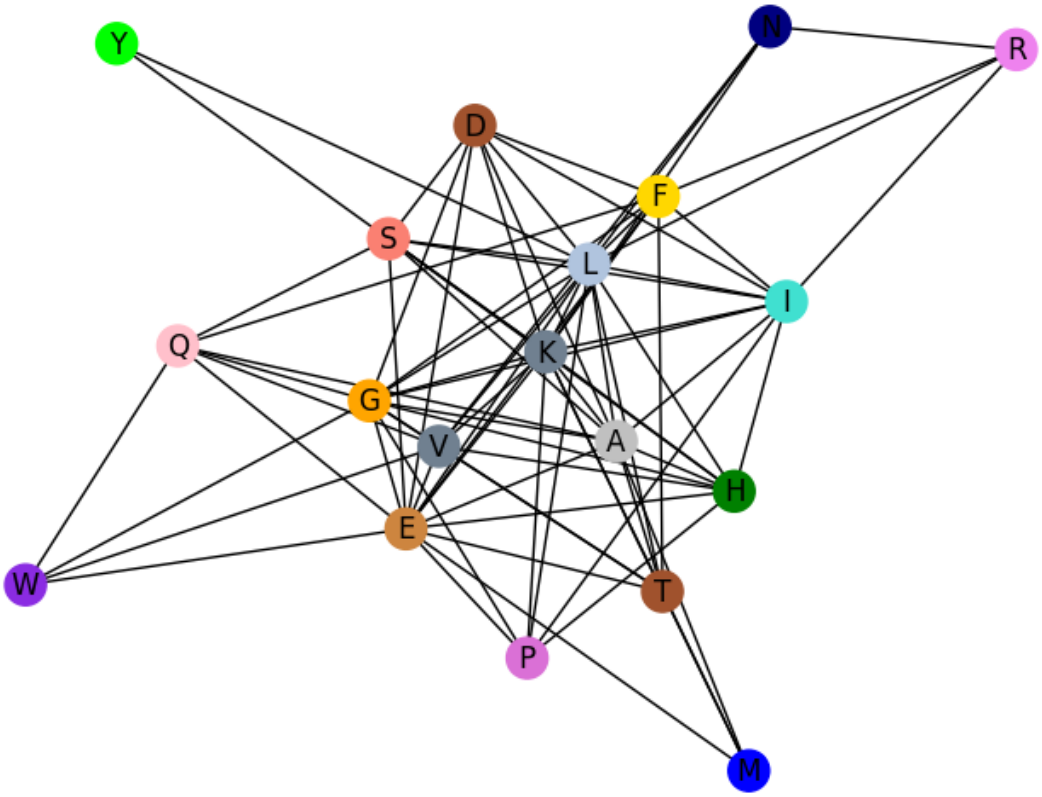
- Visualizing the first group

We extract a protein sequence from the first family and plot the corresponding graph.

[33]:
```
# Fist group setting
Seq0_data=classified_data[classified_data['labels']==0]
Seq_0=Seq0_data['original_sequence'][0]

# Plot the first group
Protein_Network_graph(Seq_0)

# Graph description
Nodes=set(list(Seq_0))
print('Nodes:',list(Nodes))
print('This graph has',len(list(Nodes)), 'nodes.')
```




```
Nodes: ['S', 'L', 'F', 'M', 'V', 'E', 'R', 'Y', 'P', 'W', 'K', 'N', 'T', 'I',
'Q', 'A', 'G', 'D', 'H']
This graph has 19 nodes.
```

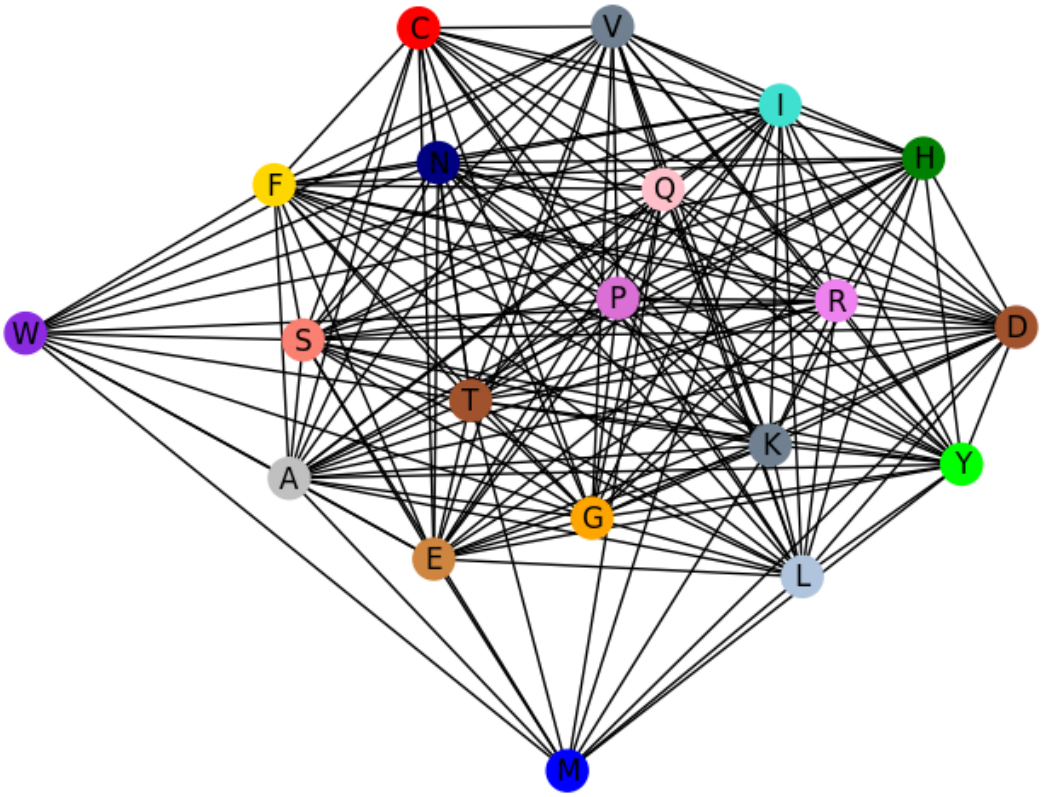
- Visualizing the second group

We extract a protein sequence from the second family and plot the corresponding graph.

[37]:
```
# Second group setting
Seq1_data=classified_data[classified_data['labels']==1]
Seq_1=Seq1_data['original_sequence'][58]

# Plot the second group
Protein_Network_graph(Seq_1)


# Graph description
Nodes=set(list(Seq_1))
print('Nodes:',list(Nodes))
print('This graph has',len(list(Nodes)), 'nodes.')
```




```
Nodes: ['S', 'L', 'F', 'M', 'V', 'E', 'R', 'Y', 'P', 'W', 'K', 'N', 'T', 'I',
'Q', 'A', 'G', 'C', 'D', 'H']
This graph has 20 nodes.
```

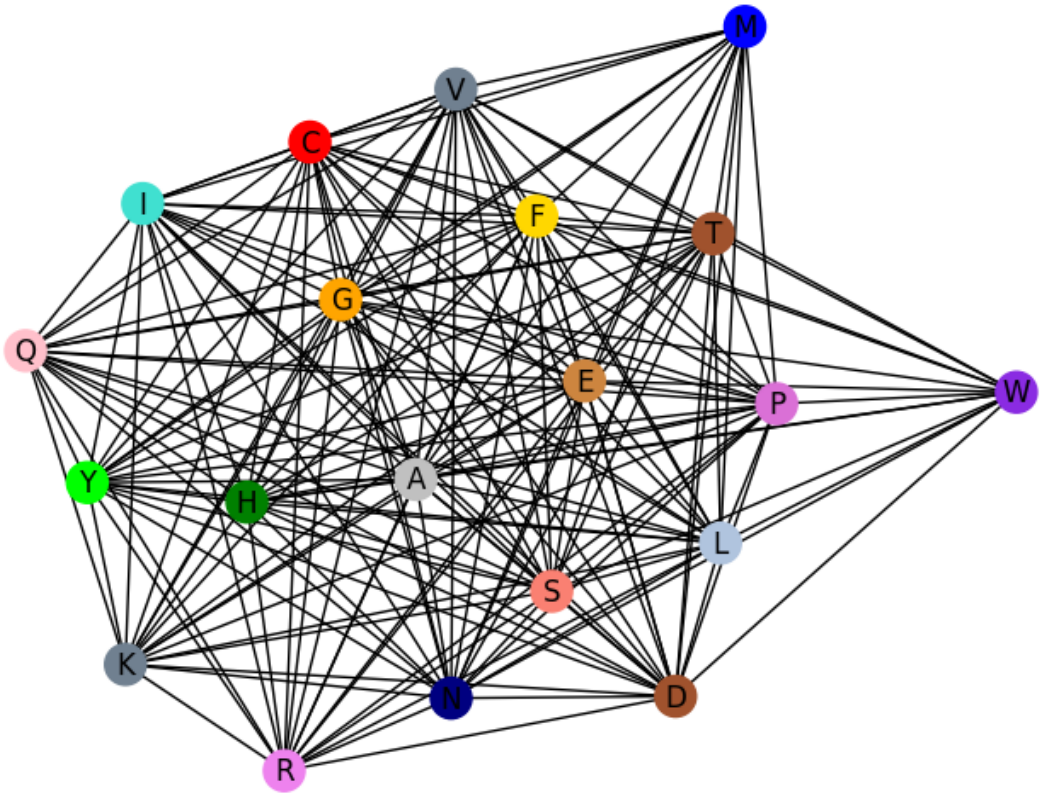
- Visualizing the third group

We extract a protein sequence from the third family and plot the corresponding graph.

[43]:
```
# Third group setting
Seq2_data=classified_data[classified_data['labels']==2]
Seq_2=Seq2_data['original_sequence'][275]

# Plot the third group
Protein_Network_graph(Seq_2)

# Graph description
Nodes=set(list(Seq_2))
print('Nodes:',list(Nodes))
print('This graph has',len(list(Nodes)), 'nodes.')
```




```
Nodes: ['S', 'L', 'F', 'M', 'V', 'E', 'R', 'Y', 'P', 'W', 'K', 'N', 'T', 'I',
'Q', 'A', 'G', 'C', 'D', 'H']
This graph has 20 nodes.
```

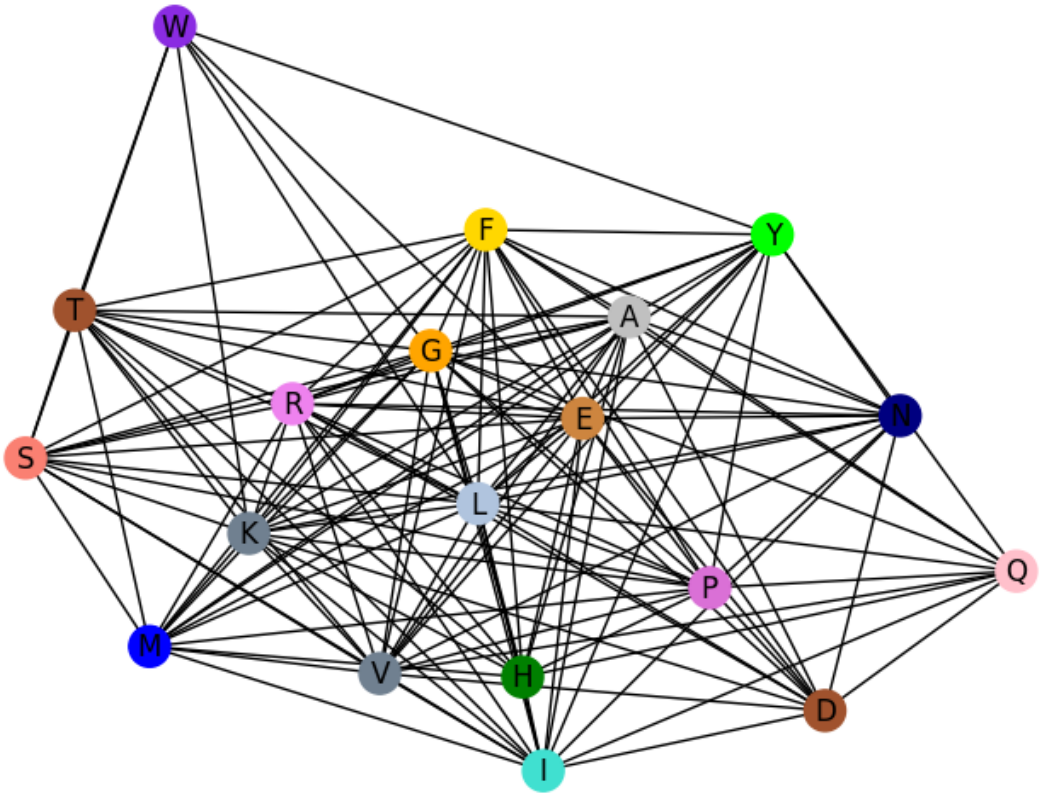
- Visualizing the fourth group

We extract a protein sequence from the fourth family and plot the corresponding graph.

[49]:
```
# Fourth group setting
Seq3_data=classified_data[classified_data['labels']==3]
Seq_3=Seq3_data['original_sequence'][303]

# Plot the fourth group
Protein_Network_graph(Seq_3)

# Graph description
Nodes=set(list(Seq_3))
print('Nodes:',list(Nodes))
print('This graph has',len(list(Nodes)), 'nodes.')
```




```
Nodes: ['S', 'L', 'F', 'M', 'V', 'E', 'R', 'Y', 'P', 'W', 'K', 'N', 'T', 'I',
'Q', 'A', 'G', 'D', 'H']
This graph has 19 nodes.
```

## 8 Conclusion

Our analysis suggests that our dataset contains 4 major families of proteins. We have drawn a graph for each member of the family. This gives insight in terms of protein density which refers to the interaction between amino acids. We then see that they all have different densities.

To perform this cluster analysis, we relied on some of the literature on this subject. This made it possible to obtain numerical features. However, we believe that some improvements can still be made in this analysis:

- We are not sure that we have included all of the defining features to differentiate proteins. Therefore, as an improvement of this work, one could make a state of the art in order to take into account all the important features necessary in the process of classification of proteins.
- One question caught our attention during this work: have we sufficiently exploited the connection between amino acids in the protein sequence? Even though we have used the "Protein scaling" and "Flexibility" features which are based on the sliding window approach, we are

not sure that they are sufficient from a generalization perspective on protein classification. One could for example add features (categorical) which take the letter of protein sequence corresponding to each given position.

- To decide on the number of clusters, we relied heavily on the topological (spartial) configuration of the points in a Euclidean space. We could further study the space of the protein points that we created (by transformation) and choose a more appropriate metric (for example: the pseudometric which comes from the alignment by pairs of sequences). We believe that the study of such a space is more adequate in a supervised classification context.

To make our codes scalable to a larger dataset, in addition to considering adding more functionality, we believe that we need to evaluate our results with real families of proteins and adjust meta parameters such as: the size of the sliding window on "Protein scaling" and "Flexibility".

# 9 References

[1] Kunchur Guruprasad, B.V.Bhasker Reddy, Madhusudan W. Pandit, Correlation between stability of a protein and its dipeptide composition: a novel approach for predicting in vivo stability of a protein from its primary sequence, Protein Engineering, Design and Selection, Volume 4, Issue 2, December 1990, Pages 155–161, https://doi.org/10.1093/protein/4.2.155

[2] Lobry JR, Gautier C. Hydrophobicity, expressivity and aromaticity are the major trends of amino-acid usage in 999 Escherichia coli chromosome-encoded genes. Nucleic Acids Res. 1994;22:3174–3180.

[3] Vihinen M, Torkkila E, Riikonen P. Accuracy of protein flexibility predictions. Proteins. 1994 Jun;19(2):141-149. DOI: 10.1002/prot.340190207.