Protein Secondary Structure Prediction

Frank Li

April 28, 2020

1 Abstract

We attempt to use K-Nearest-Neighbor(KNN) approaches to predict protein secondary structures from its primary amino acid sequence. A flexible and easy-to-use Python3 script was developed to achieve this goal with versatile options to run(Brute-Force approaches, kd Tree approaches, distance-weighted vote, uniform vote) and to visualize the result. As expected, KNN doesn't perform very well with the average accuracy around 50% since it is an easy and primitive learning approaches, I suppose Hidden-Markov-Model(HMM) and deep learning could have better effectiveness to solve this problem.

In the following section, I will elaborate how to use this script, choose correct options and visualization of the final result.

2 Set up the question and workflow

Initially, we downloaded the data sets (stru_benchmark_sable135.txt) which contains sequence and secondary structure information of 135 protein families. After filtering out items whose sequence contain "X", 93 protein families were retained for downstream model selection.

We randomly split the 93 protein families into 5 groups, 5-fold cross testing was performed to assess the effectiveness of the algorithm. In brief, 80% of protein families are fed into KNN model as traing set, remaining 20% of protein families will be used for testing.

For each testing protein, the accuracy is defined as

$$Accuracy = \frac{Correctly\ Predicted\ Residue}{length\ of\ protein}$$

A clear explanation is as below:

Protein A:
Sequence. :A-V-I-L-L-L-M-F-Y-K
Correct SS :H-H-S-S-S-S-C-C-C-C
Predict SS :H-H-**H**-**H**-S-S-C-C-C-C
There are two residues incorrectly predicted, and the length of this protein is 10
The accuracy = $\frac{8}{10} = 0.8$

So, suppose we have 20 testing proteins, we will have an array of accuracy with the length 20, for instance,[0.56,0.67,...,0.76,0.34]. Since we are perform 5-fold cross testing, for each set of parameter, all of these 93 protein families will be used for testing, we will get an array storing accuracy with the length 93. Mean value and 95% confidence interval will be calculated as final metrics for reporting. **For example, when choose k=3 Nearest Neighbors and sliding window length = 5, the average accuracy is 0.54, 95% confidence level is [0.47,0.56].**

3 Running options

KNN algorithm needs us to compute the distance between testing point and each training data point, when the training data sets increase, the corresponding running time will dramatically increase. I tested **brute force** approach in my local computer, when choosing k=3 and window length = 5, it took python 70 minutes to finish 5-fold cross testing. **So it is highly discouraged to use brute force options**, even if we have this options available.

Another crucial parameter when running KNN is whether to use distance-weighted vote or uniform vote. For example, testing point A is close to B(class:0),C(class:1),D(class:0), when using uniform vote, testing point A will be assigned as 0 because we have 2 votes of 0 but 1 vote of 1. When considering distance, the vote will be calculated as below:

$$vote(0) = \frac{1}{dist_{A,B}} + \frac{1}{dist_{A,D}}$$
$$vote(1) = \frac{1}{dist_{A,C}}$$

Then we take majority vote based on calculation shown above. I tested this two methods, two methods don't have pronounced differences in this setting, so you are recommended to try both votes: **"distance"** and **"uniform"**.

4 Usage of tool

Several Dependencies need to be installed at first:

- numpy
- functools

- scipy
- scikit-learn (If using kd-tree approaches)

Only one thing needs to adjust is the path of the input file, then enter your terminal window, suppose we are using k=3 nearest neighbors, sliding window length=5, using "kdTree" and distance-weighted vote, you just need to specify -l as 5(length), -k as 3(KNN), -m as "kdTree"(mode), -v as distance(vote), running as below:

```
python3 protein_secondary_structure_predictor.py -l 5 -k 3 -m kdTree -v distance
```

Listing 1: Running script

If you want to obtain all help information:

```
python3 protein_secondary_structure_predictor.py -h
```

Listing 2: Help Information

Parameters are as following:

-l –length : length of sliding window, you could pick 5,7,9,11
-k –k: K nearest neighbors, increasing k will result in longer runtime
-m –mode: bruteForce or kdTree, it is discouraged to use bruteForce
-v –vote: distance or uniform
-h –help: check help information

5 Result

We stick with mode = "kdTree" and vote = "distance", and choose k=3,7,15,30, in the meantime, choose sliding window length=5,7,9,11. We calculate total accuracy in each setting and the result is shown as Figure 1

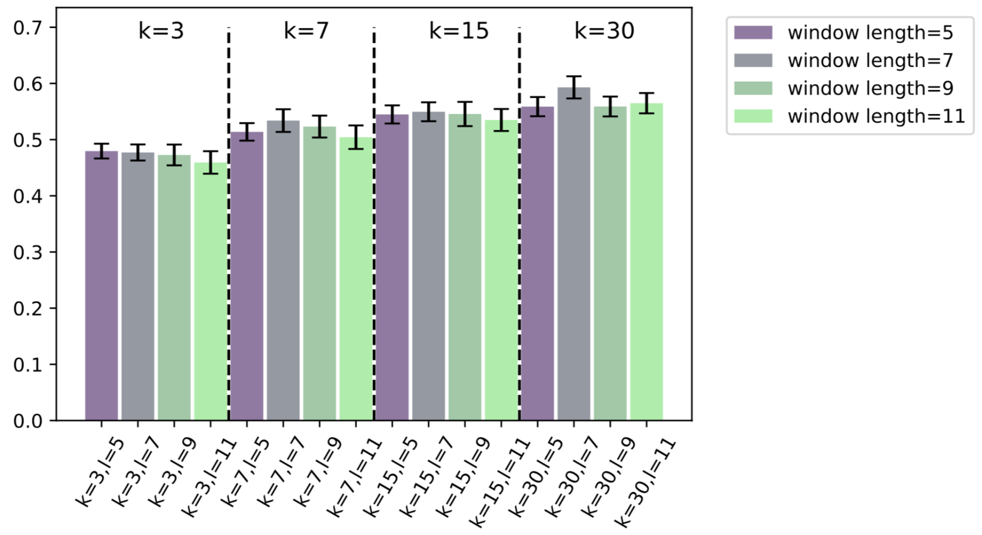

Figure 1: Accuracy under different combination of K and sliding window length

6 Discussion

The overall accuracy is around 50%, with the increase of K, the performance has slight boost but not pronounced. In principle, there is supposed to be a ”sweet spot” of K, since too large K value will incorporate lots of points from other clusters while too small K value result in biased result. Since the running time increase with the increase of K, it is not possible to exactly determine where this ”sweet spot” will be, but based on my testing, it is recommended to increase k value when $k < 100$.

Another parameter is sliding window length, in principle, with the increase of sliding window length, more effects of flanking sequences will be taken into account. In this sense, more information will be fed into KNN model, it will lead to higher accuracy. But based on the result, increase of sliding window length has no significant impact on accuracy and in some case, longer sliding window even worsen the performance. My reasoning is that, KNN is not a suitable method to fully detect the sequential information involved in the flanking sequence. Two methods that might improve its performance, one is to use position specific score matrix(PSSM) matrix instead of one hot encoding, it is able to capture and contain more information and presumably, could make the prediction more accurate. Another worthwhile endeavor would be using context-sensitive Hidden Markov Model(HMM), with the advantage of simultaneously capturing sequential information and context in which it involved, it might be a better approach to establish the probabilistic model and make predictions about protein secondary structures.

7 Appendix I: main script

```python
#!/Users/ligk2e/opt/anaconda3/envs/python3/bin/python3
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 21 18:14:10 2020

@author: ligk2e
"""
import numpy as np

class ProteinFamily():

    def __init__(self,familyID,seq,ss):
        self.ID = familyID
        self.seq = seq
        self.ss = ss

    def ambientOneHotEncoding(self,length):   # well, assume sliding window length will be an odd number
        result = []
        dic = {'H':0,'C':1,'E':2}
        for i in range(0,len(self.seq)-length+1):
            windowSeq = self.seq[i:i+length]
            windowSS = self.ss[i:i+length]
            middle = (length-1)//2
            label = dic[windowSS[middle]]
            oneHot = ProteinFamily.oneHotEncoding(windowSeq)
            result.append((oneHot,label))   # [0,0,0.....1,0,0],'0'
        return result



    @staticmethod
    def oneHotEncoding(seq):
        '''
        A-R-N-D-C-Q-E-G-H-I-L-K-M-F-P-S-T-W-Y-V

        '''
        # Cysteine(C) will result in [0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
        result = []
        template = np.zeros(20)
        dic = {'A':0,'R':1,'N':2,'D':3,'C':4,'Q':5,'E':6,'G':7,'H':8,'I':9,'L':10,'K':11,'M':12,'F':13,'P':14,'S':15,
               'T':16,'W':17,'Y':18,'V':19}
        for letter in seq:
            template[dic[letter]] = 1   # assign the certain position to 1
            result.extend(template)     # extend it to result array
            template = np.zeros(20)     # reset the template to all zeros
        return result




    @staticmethod
    def parseFile(path):
        with open(path,'r') as file1:
            content = file1.readlines()
        familyID, seq, ss = [],[],[]
        for index,value in enumerate(content):
            if index % 4 == 0: familyID.append(value.lstrip('>').rstrip('\n'))  # >1cix
            elif index % 4 == 1: seq.append(value.rstrip('\n'))   # consensus protein sequence
            elif index % 4 == 2: ss.append(value.rstrip('\n'))   # consensus secondary structure notation
        familyInfo = [(familyID[i],seq[i],ss[i]) for i in range(len(familyID))]
        # filter items with 'X' in the sequence
        familyInfoNew = list(filter(lambda x: 'X' not in x[1],familyInfo))
        return familyInfoNew



    @staticmethod
    def kFoldSplit(lis,i):    # well, only for 5 fold
        import random
        random.Random(4).shuffle(lis)  # specify random state as 4
        training, testing = [],[]
        for index, value in enumerate(lis):
            if index % 5 == i: testing.append(value)
            else: training.append(value)
        return training,testing




```

```python
class KNNmachine():


    def __init__(self,X1,Y1,testingData):
        self.X1 = X1
        self.Y1 = Y1
        self.testingData = testingData

    def bruteForce(self,k):
        finalResult = []
        for eachProtein in self.testingData:
            window = len(eachProtein)
            stat = []
            for eachWindow in eachProtein:
                temp = np.array(eachWindow[0])
                #print(k)
                prediction = KNNmachine.distance(self.X1,self.Y1,temp,k)  # deploy instance
    function
                stat.append(1) if prediction == eachWindow[1] else stat.append(0)
            from functools import reduce
            percentage = reduce(lambda a,b:a+b,stat)/window
            finalResult.append(round(percentage,2))
        self.prediction = finalResult   #[80%,45%,34%,98%...]
        from statistics import mean
        print(finalResult)
        print('This round yield {0} average accuracy'.format(mean(finalResult)))




    @staticmethod
    def distance(X1,Y1,temp,k):   # hamming distance between a testing point and a training point
        from scipy.spatial.distance import hamming
        allDist = []
        for i in range(len(Y1)):
            ref = X1[i,:]
            dist = hamming(ref,temp)  # hamming function only accept 1D array
            allDist.append(dist)
        '''
        Y1:         0,1,0,0,2,0....
        allDist:    4,5,3,5,........   (distance)
        take the maximum k number neighbors
        '''
        kNneighbors =  sorted(zip(allDist,Y1),key=lambda x:x[0],reverse=True)[:k]
        labels = [neighbor[1] for neighbor in kNneighbors]
        from collections import Counter
        count = Counter(labels)  # [0,1,1,2,2,2,1,1] will be {0:1,1:4,2:3}
        prediction = max(count,key=lambda x:count[x])  # smart solution
        return prediction



    def constructModel(self,k,mode='distance'):  # mode = 'uniform' so don't account for distance
        from sklearn.neighbors import KNeighborsClassifier
        clf = KNeighborsClassifier(k,weights=mode,algorithm='kd_tree')
        clf.fit(self.X1,self.Y1)
        self.clf = clf


    def predict(self):  #[   [   ([],0)  ,  ([],1)   ],[],[]]
        testing = len(self.testingData)   # how many protein in testing set
        finalResult = []
        for eachProtein in self.testingData:   # [  (  [],0     ),(),()   ]
            window = len(eachProtein)   # how many windows in a protein
            stat = []
            for eachWindow in eachProtein:   # (   [],0  )
                temp = np.array(eachWindow[0]).reshape(1,-1)   # from i-d array to 1*n 2d row vector
                #print(temp.shape)
                prediction = list(self.clf.predict(temp))[0]
                #print(prediction,type(prediction))

                stat.append(1) if prediction == eachWindow[1] else stat.append(0)
            from functools import reduce
            percentage = reduce(lambda a,b:a+b,stat)/window   # 80% of the residue' secondary structure are correctly predicted
            #if percentage >= 0.6: print(eachProtein,stat)
            finalResult.append(round(percentage,2))
        self.prediction = finalResult   #[80%,45%,34%,98%...]
        from statistics import mean
        print(finalResult)
        print('This round yield {0} average accuracy'.format(mean(finalResult)))
        return finalResult


    @staticmethod
    def decoder(oneHotEncodingProtein,stat):
        dic1= {0:'A',1:'R',2:'N',3:'D',4:'C',5:'Q',6:'E',7:'G',8:'H',9:'I',10:'L',11:'K',12:'M',13:'F',14:'P',15:'S',
                16:'T',17:'W',18:'Y',19:'V'}
        dic2= {0:'H',1:'C',2:'E'}
        protein,secondStructure = [],[]
        for eachWindow in oneHotEncodingProtein:
            ss = dic2[eachWindow[1]]
            secondStructure.append(ss)
            seqOri = eachWindow[0]
            seqDeco = ''
            for i in range(0,len(seq),20):
                aa = dic[seq[i,i+20].index(1.0)]
                seqDeco += aa
            protein.append(seqDeco)
        # protein: [RTYDY,TYDYD,...,FETGD]
        # secondStructure: [H,C,C,E...E]
        reconstruct = ''
        for each in protein:
            if protein.index(each) == len(protein) - 1: remain = each
            else: remain = each[0]
            reconstruct += remain
        print(reconstruct,secondStructure,stat)

```

```


def main(length,k,mode='kdTree',vote='distance'):
    accuracyCollect = []
    for i in range(5):
        trainingData,testingData = [],[]
        training,testing = ProteinFamily.kFoldSplit(familyInfoNew,i)
        for item in training:
            member = ProteinFamily(item[0],item[1],item[2])
            result = member.ambientOneHotEncoding(length)
            trainingData.extend(result)    #[([],0),([],1)...]
        for item in testing:
            member = ProteinFamily(item[0],item[1],item[2])
            result = member.ambientOneHotEncoding(length)
            testingData.append(result)    # for testing set, I preserve where each sliding window
    comes from
        X1,Y1= [],[]
        for j in trainingData:
            X1.append(j[0])
            Y1.append(j[1])
        X1,Y1 = np.array(X1),np.array(Y1)  # [[0,0,0,1,0,0,1,....],[0,1,0,1,0,0,0,...]],
    [0,1,2,1,2,1,...]
        #print(X1,Y1,type(X1),type(Y1),X1.shape,Y1.shape)
        machine = KNNmachine(X1,Y1,testingData)

        if mode == "bruteForce": machine.bruteForce(k)
        elif mode == "kdTree":
            machine.constructModel(k,vote)
            accuracy = machine.predict()
            accuracyCollect.extend(accuracy)
    return accuracyCollect


def usage():
    print('Usage:')
    print('python3 protein_secondary_structure_predictor.py -l 5 -k 3 -m kdTree -v distance')
    print('Options:')
    print('-l --length : length of sliding window, you could pick 5,7,9,11')
    print('-k --k: K nearest neighbors, increasing k will result in longer runtime')
    print('-m --mode: bruteForce or kdTree, it is discouraged to use bruteForce')
    print('-v --vote: distance will use distance-weighted measure when assigning label to each
    testing point, uniform will not consider that')
    print('-h --help: check help information ')
    print('Author: Guangyuan(Frank) Li <li2g2@mail.uc.edu>, PhD Student, University of Cincinnati,
     2020')


def confidenceInterval(lis):
    import numpy as np, scipy.stats as st
    a = np.array(lis)
    me = np.mean(a)
    confInt = st.t.interval(0.95, len(a)-1, loc=me, scale=st.sem(a))  # will return a tuple (lower
    ,upper)
    errorBar1 = np.std(a)   # using standard deviation as errorbar, yerr=errorBar1, lower error =
    upper error = errorBar1
```

```python
    errorBar2 = st.sem(a)   # using standard error of mean as errorbar, same as above
    errorBar3 = [me-confInt[0],confInt[1]-me]  # using confidence interval as errorbar, yerr=errorBar3, lower error = errorBar3[0], upper error = errorBar3[1]
    return me, confInt, errorBar3   # these three will be combined as tuple automatically, if function return multiple values


if __name__ == '__main__':
    familyInfoNew = ProteinFamily.parseFile('/Users/ligk2e/Desktop/ssprotein/sec_stru_benchmark_sable135.txt')

    import getopt
    import sys
    try:
        options, remainder = getopt.getopt(sys.argv[1:],'hl:k:m:v:',['help','length=','k=','mode=','vote='])
    except getopt.GetoptError as err:
        print('ERROR:', err)
        usage()
        sys.exit(1)
    for opt, arg in options:
        if opt in ('-l','--length'):
            length = int(arg)
            print('Sliding Window Length:', arg)
        elif opt in ('-k','--k'):
            k = int(arg)
            print('K value for KNN:',arg)
        elif opt in ('-m','--mode'):
            mode = arg
            print('mode of KNN:', arg)
        elif opt in ('-v','--vote'):
            vote = arg
            print('vote measure when using KNN:',arg)
        elif opt in ('--help','-h'):
            usage()
            sys.exit()


    accuracy = main(length,k,mode,vote)
```

Listing 3: Python Code

8 Appendix II: Bar Plot

```python
    accuracy_l5_k3 = main(5,3,'kdTree','distance')
    accuracy_l7_k3 = main(7,3,'kdTree','distance')
    accuracy_l9_k3 = main(9,3,'kdTree','distance')
    accuracy_l11_k3 = main(11,3,'kdTree','distance')

    accuracy_l5_k5 = main(5,7,'kdTree','distance')
    accuracy_l7_k5 = main(7,7,'kdTree','distance')
    accuracy_l9_k5 = main(9,7,'kdTree','distance')
    accuracy_l11_k5 = main(11,7,'kdTree','distance')

    accuracy_l5_k7 = main(5,15,'kdTree','distance')
    accuracy_l7_k7 = main(7,15,'kdTree','distance')
    accuracy_l9_k7 = main(9,15,'kdTree','distance')
    accuracy_l11_k7 = main(11,15,'kdTree','distance')

    accuracy_l5_k10 = main(5,30,'kdTree','distance')
    accuracy_l7_k10 = main(7,30,'kdTree','distance')
    accuracy_l9_k10 = main(9,30,'kdTree','distance')
    accuracy_l11_k10 = main(11,30,'kdTree','distance')

    import matplotlib.pyplot as plt

    fig = plt.figure()

    barWidth = 0.9
    # in following: 1 means l=5, 2 means l=7, 3 means l=9, 4 means l=11
    r1 = [1,5,9,13]
    r2 = [2,6,10,14]
    r3 = [3,7,11,15]
    r4 = [4,8,12,16]
    r5 = sorted(r1 + r2 + r3 + r4)

    bar1 = [confidenceInterval(item)[0] for item in [accuracy_l5_k3,accuracy_l5_k5,accuracy_l5_k7,
    accuracy_l5_k10]]
    bar2 = [confidenceInterval(item)[0] for item in [accuracy_l7_k3,accuracy_l7_k5,accuracy_l7_k7,
    accuracy_l7_k10]]
    bar3 = [confidenceInterval(item)[0] for item in [accuracy_l9_k3,accuracy_l9_k5,accuracy_l9_k7,
    accuracy_l9_k10]]
    bar4 = [confidenceInterval(item)[0] for item in [accuracy_l11_k3,accuracy_l11_k5,
    accuracy_l11_k7,accuracy_l11_k10]]

    yer1 = np.transpose(np.array([confidenceInterval(item)[2] for item in [accuracy_l5_k3,
    accuracy_l5_k5,accuracy_l5_k7,accuracy_l5_k10]]))
    yer2 = np.transpose(np.array([confidenceInterval(item)[2] for item in [accuracy_l7_k3,
    accuracy_l7_k5,accuracy_l7_k7,accuracy_l7_k10]]))
    yer3 = np.transpose(np.array([confidenceInterval(item)[2] for item in [accuracy_l9_k3,
    accuracy_l9_k5,accuracy_l9_k7,accuracy_l9_k10]]))
    yer4 = np.transpose(np.array([confidenceInterval(item)[2] for item in [accuracy_l11_k3,
    accuracy_l11_k5,accuracy_l11_k7,accuracy_l11_k10]]))

    plt.bar(r1,bar1,width=barWidth,color=(0.3,0.1,0.4,0.6),yerr = yer1,capsize=4,label='window
    length=5')
    plt.bar(r2,bar2,width=barWidth,color=(0.3,0.33,0.4,0.6),yerr = yer2,capsize=4,label='window
    length=7')
    plt.bar(r3,bar3,width=barWidth,color=(0.3,0.65,0.4,0.6),yerr = yer3,capsize=4,label='window
    length=9')
    plt.bar(r4,bar4,width=barWidth,color=(0.3,0.9,0.4,0.6),yerr = yer4,capsize=4,label='window
    length=11')

    plt.vlines(4.50,0.0,0.70,linestyles='dashed')
    plt.vlines(8.50,0.0,0.70,linestyles='dashed')
    plt.vlines(12.50,0.0,0.70,linestyles='dashed')
    text = ['k=3','k=7','k=15','k=30']
    for i in range(4):
        plt.text(x=i*4+2.0,y=0.68,s=text[i],size=12)

    plt.legend(bbox_to_anchor=(1.04,1),fontsize=10)
    plt.xticks([r for r in r5],['k=3,l=5','k=3,l=7','k=3,l=9','k=3,l=11',
                                               'k=7,l=5','k=7,l=7','k=7,l=9','k=7,l=11',
                                               'k=15,l=5','k=15,l=7','k=15,l=9','k=15,l=11',
                                               'k=30,l=5','k=30,l=7','k=30,l=9','k=30,l=11'],rotation
    =60)
    plt.savefig('fig1.pdf',bbox_inches='tight')
    plt.show()
```

Listing 4: Bar chart coding