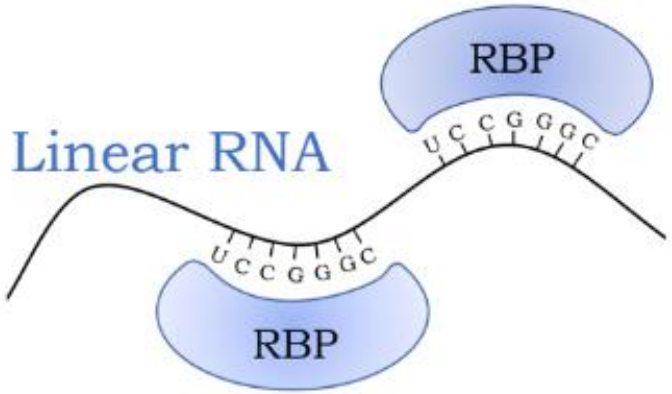
一.课题主要任务



以线型RNA举例，我们的预测目标是找出RBP上与UCCGGGC这些核苷酸结合的位点，不过核苷酸没有用在训练预测信息上，而是用于计算标签。

也就是说，根据往年论文的欧氏距离计算方法，将RNA和RBP间距离相近的残基定义为结合位点。我们的研究是找出RBP上和RNA结合的位点，所以得到的结合位点都是氨基酸，如下图的蛋白链示例，+表示【确认为结合位点】

```
MGRGKVKPNRKSTGDNSNVVTMIRAGSYPKVNPTPTWVRAIPFEVSVQSGIAFKVP
-----------+--------+--------++--++++++------+---------

VGSLFSANFRTDSFTSVTVMSVRAWTQLTPPVNEYSFVRLKPLFKTGDSTEEFEGRA
--------------------++-+----+------------------------------

SNINTRASVGYRIPTNLRQNTVAADNVCEVRSNCRQVALVISCCFN
----------+------+-----------------------+++++--
```

二.具体工作和进展（主体代码）

1. 采用biopython模块读取pdb文件，索引是从Liu（2015）论文提供的数据集PDBinter的seq名称，这一步没有锁定到链名，导致了接下来的计算标签浪费了不少时间。

```python
from Bio.PDB import *
import pandas as pd

data = pd.read_csv('PDB.seq')        # PDB.seq就是PDBinter改名
pdb = data['Seq'].tolist()       # 作为列表返回

for line in pdb:
    # 定义pdb_id
    pdb_id = line[1:5]       # 每行锁定名字

    # 开始获取蛋白质pdb文件，并保存至protein文件夹
    pdbl = PDBList()         # 通过互联网访问PDB(例如下载结构)
    pdbl.retrieve_pdb_file(pdb_id, pdir='./pdb', file_format='pdb')      # 指定了文件保存位置
    parser = PDBParser(PERMISSIVE=True, QUIET=True)      # quiet应该设置false的，能屏蔽错误结构，但是permissive已经排除了部分异常

    # 如果pdb文件不存在，则跳过该pdb
    try:
        data = parser.get_structure(pdb_id, 'pdb' + pdb_id.lower() + '.ent')        # 返回model/chain/residue/atom
    except:
        pass
```

2. 本地生成PSSM矩阵，将PDBinter分割的fasta文件单个上传，在uniref数据库进行psiblast；pssm_blast.sh文件返回了比对所得打分矩阵，保存于pssm文件夹

```python
from tqdm import tqdm
import pandas as pd

def sh_text(item):
    fasta_file = './fasta/' + item + '.fasta'
    pssm_file = './pssm/' + item + '.pssm'
    s = 'psiblast -query ' + fasta_file + ' -db ./uniref/uniprot_sprot.fasta -num_iterations 3 -out_ascii_pssm ' + pssm_file + '\n'
    with open('pssm_blast.sh', 'a') as f:
        f.write(s)


data = pd.read_csv("PDB.seq")
sequences = data['Seq'].tolist()

for seq in sequences:
    sh_text(seq[1:5])       # 链名传进了列表，变成了sh_text的item
```

```
Query_name: 1A34
Query_length: 147
TGDNSNVVTMIRAGSYPKVNPTPTWVRAIPFEVSVQSGIAFKVPVGSLFSANFRTDSFTSVTVMSVRAWTQLTPPVNEYSFVRLKPLFKTGDSTEEFEGRASNINTRASVGYRIPTNLRQNTVAADNVCEVRSNCRQVALVISCCFN
CCCCCCEEEEEECCCCCCCCCCCCCEEEEEEEEEECCEEEEECCCCEEEEEECCCEEEEEEEEEEEECCCCCCEEEEEEEECCCCCCCCEEEEEECCCCCEEEEEECCCCCEEEEEEEECCCCCCCCEECCCEEEEECCCCEEEEEEEECC
EEEEEeBBeBBeEEEbEEEEEEbEbBeBBeBeBbBEeeBBBeBeEeeBBebEBEEEEbEeBbBBbBeBbbEbEEEEEbbBBeBeEbbEEEEeEEEBEEeBEEbEeEBeBBbEBbEEBEEEeEeBEeBBeBeeeBeeBBBBBBBbbE
```

3. 将氨基酸fasta文件上传到PaleAle进行二级结构和溶剂可及性预测，返回如上图所示的表格，序列下第一行是二级结构，第二行是溶剂可及性，C E H分别代表环，折叠和螺旋，B b e E分别表示该残基<=4%可及，>4%但<25%可及，>25%但<50%可及，>50%可及；

下图的代码将这些性质编码为特征向量，如C以（1,0,0）表示，共返回7维向量

```python
"""
    功能：生成氨基酸的子结构特征
    输入：一个氨基酸的子结构序列
    返回：该氨基酸的特征向量
"""
def sub_encoding(sub_seq):
    result = []
    for i in sub_seq:
        if i == 'C':
            result.append([1, 0, 0])
        elif i == 'E':
            result.append([0, 1, 0])
        elif i == 'H':
            result.append([0, 0, 1])

    return result
```

```python
"""
    功能：生成氨基酸的溶剂可及性特征
    输入：一个氨基酸的名称
    返回：该氨基酸的特征向量
"""
def rsa_encoding(rsa_seq):
    result = []
    for i in rsa_seq:
        if i == 'B':
            result.append([1, 0, 0, 0])
        elif i == 'b':
            result.append([0, 1, 0, 0])
        elif i == 'e':
            result.append([0, 0, 1, 0])
        elif i == 'E':
            result.append([0, 0, 0, 1])

    return result
```

4. 氨基酸理化性质和类别编码，由（1,0,0,0,0,0）表示ALA，7是其侧链pKa值，1.8是其疏水性参数，共返回11维向量

```python
"""
    功能：存放氨基酸的特征向量字典，1-6为氨基酸类别编码，7-11为理化性质编码
    输入：一个氨基酸名称
    返回：该氨基酸的特征向量
"""
def amino_encoding(amino_name):
    amino = {'ALA': [1, 0, 0, 0, 0, 0, 5, 0, 2, 7.00, 1.8],
             'CYS': [0, 0, 1, 0, 0, 0, 6, 0, 2, 7.00, 2.5],
             'ASP': [0, 0, 0, 0, 0, 1, 8, -1, 4, 3.65, -3.5],
             'GLU': [0, 0, 0, 0, 0, 1, 9, -1, 4, 3.22, -3.5],
             'PHE': [0, 1, 0, 0, 0, 0, 11, 0, 2, 7.00, 2.8],
             'GLY': [1, 0, 0, 0, 0, 0, 4, 0, 2, 7.00, -0.4],
             'HIS': [0, 0, 0, 1, 0, 0, 10, 1, 4, 6.00, -3.2],
             'ILE': [0, 1, 0, 0, 0, 0, 8, 0, 2, 7.00, 4.5],
             'LYS': [0, 0, 0, 0, 1, 0, 9, 1, 2, 10.53, 3.9],
             'LEU': [0, 1, 0, 0, 0, 0, 8, 0, 2, 7.00, 3.8],
             'MET': [0, 0, 1, 0, 0, 0, 8, 0, 2, 7.00, 1.9],
             'ASN': [0, 0, 0, 1, 0, 0, 8, 0, 4, 8.18, -3.5],
             'PRO': [0, 1, 0, 0, 0, 0, 7, 0, 2, 7.00, -1.6],
             'GLN': [0, 0, 0, 1, 0, 0, 9, 0, 4, 7.00, -3.5],
             'ARG': [0, 0, 0, 0, 1, 0, 11, 1, 4, 12.48, -4.5],
             'SER': [0, 0, 1, 0, 0, 0, 6, 0, 4, 7.00, -0.8],
             'THR': [0, 0, 1, 0, 0, 0, 7, 0, 4, 7.00, -0.7],
             'VAL': [1, 0, 0, 0, 0, 0, 7, 0, 2, 7.00, 4.2],
             'TRP': [0, 0, 0, 1, 0, 0, 14, 0, 3, 7.00, -0.9],
             'TYR': [0, 0, 1, 0, 0, 0, 12, 0, 3, 10.07, -1.3]}

    return amino[amino_name]
```

5. PSSM矩阵的打分列直接转换为20维向量，返回给氨基酸

1A34.pssm - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

Last position-specific scoring matrix computed, weighted observed percentages rounded down, information per position, and relative weight of gapless real matches to pseudocounts

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V		
1 T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00
2 G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-3	-3	-3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00
3 D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-4	-1	0	-1	-4	-3	-3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00
4 N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00
5 S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00
6 N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00
7 V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00
8 V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00
9 T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00
10 M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00
11 I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00
12 R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00
13 A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00

```
139    def gen_pssm(pdb_id, num):
140        reader = open("./pssm/" + pdb_id + '.pssm').readlines()
141        for line_num, line in enumerate(reader):
142            if line_num - 3 == num and line[:2] == '  ' and line[4:5] != ' ':        # 锁定到AA所在的打分列
143                data = line[7:].split(' ')
144                for num, i in enumerate(data):
145                    if i == '':
146                        data.remove(i)      # 可能是剪切的意思
147
148                feat_pssm = [int(i) for i in data[:20]]
149
150        return feat_pssm
```

6. 最繁琐的任务：标签计算，虽然只是给氨基酸返回要么1要么0两种标签，但计算过程涉及氨基酸原子及坐标定位，残基标识码和链名定位（否则生成的标签会相互覆盖），RNA原子及坐标定位，以及缩进问题（直接影响循环效率），调整缩进之前150个蛋白链需要计算3天，调整后（相当于算法优化后）只需要1小时。

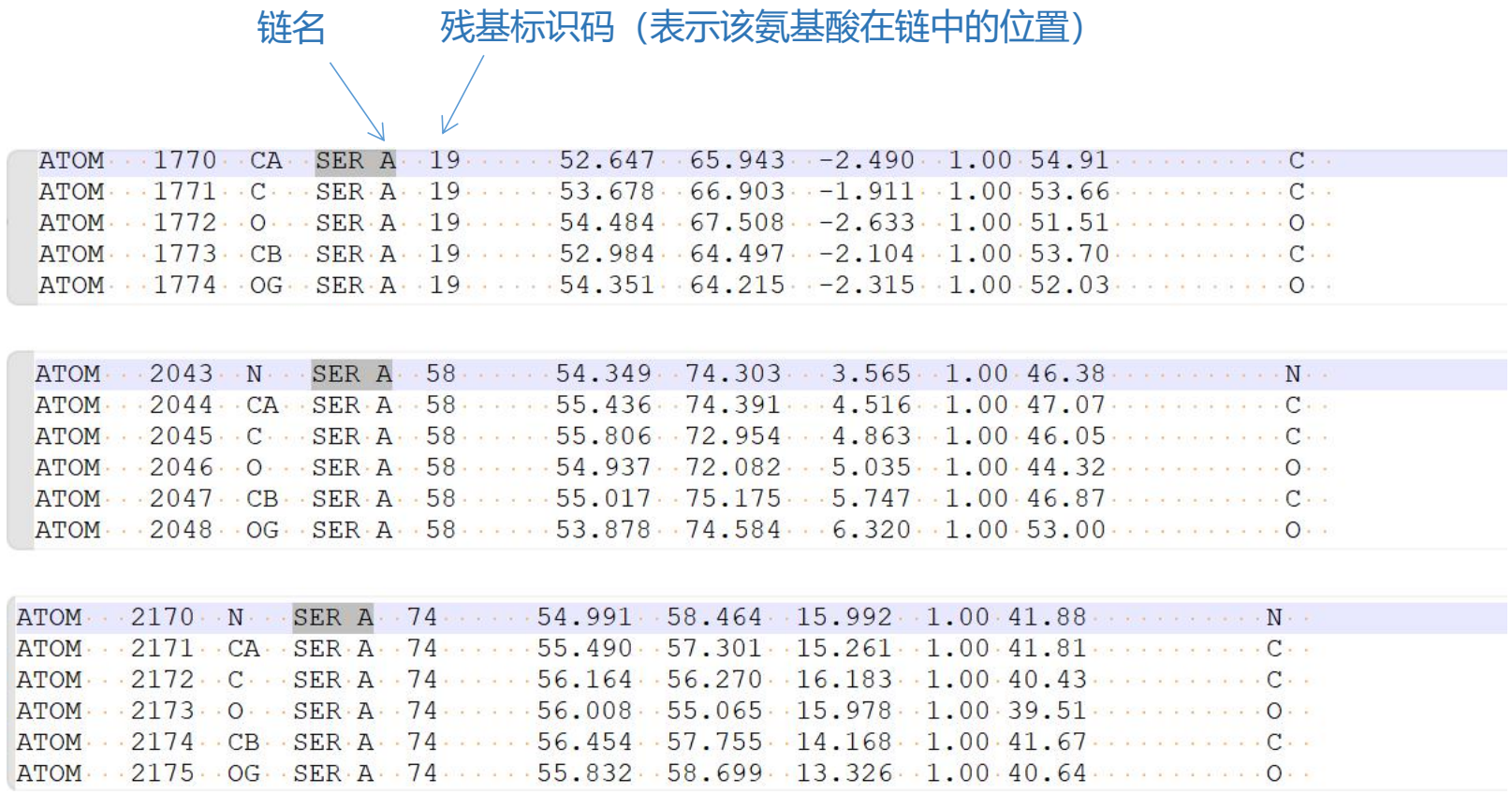
```python
def gen_label(pdb, pdb2, amino_name, protein_name, index, rna_name):
    # print(amino_name + ":" + protein_name + ":" + str(index))
    pdb1 = []
    for line_amino in pdb:
        if line_amino[:4] == 'ATOM' and line_amino[17:20] == amino_name and protein_name == line_amino[21:22]: # 此时为氨基酸的原子
            index2 = int(line_amino[22:26])     #残基标识码
            if index == index2:
                pdb1.append(line_amino)

    for line_amino in pdb1:
        # print(protein_name) # 此时为氨基酸的原子
        line2 = [i for i in line_amino.strip()[28:].split(' ') if i != '']
        amino_cord = np.array([float(i) for i in line2[0:3]])  # 得氨基酸原子的坐标
        for line_rna in pdb2:
            # print(rna_name) # 此时为RNA的原子
            line1 = [i for i in line_rna.strip()[28:].split(' ') if i != '']
            rna_cord = np.array([float(i) for i in line1[0:3]])  # 得RNA原子的坐标

            if np.sqrt(np.sum(np.square(amino_cord - rna_cord))) < 6:
                return 1

    return 0
```

由于ATOM部分信息较多，一开始只考虑了用氨基酸名字参与索引，计算；后来补上链名和残基标识码，这些细节都是过往论文没有提到的



7. 合并所有特征，依次从seq文件提取氨基酸，计算特征，合并后保存到CSV表格中，val,train,test三个表格代表后续模型训练用到的验证集，训练集和测试集；

所生成38维度特征向量的示例见最下方

```
            label = gen_label(pdb, pdb2, amino_lower2upper(amino_name), protein_name, num+start, rna_name)        # 获得氨基酸的标签

            amino_feats.append(feat_11 + feat_sub_ras + feat_pssm)          # 将氨基酸的所有特征合并
            amino_names.append(amino_name)                                  # 将氨基酸的名称合并
            amino_labels.append(label)                                      # 将氨基酸的标签合并
            all_feat = {"name": amino_names, "feat": amino_feats, "label": amino_labels}

            df_all_feat = pd.DataFrame(all_feat)
            df_all_feat.to_csv(save_path)
```

```
val_count_amino, val_count_1, val_count_0 = gen_all_feature("dataset/val_seq.csv", "dataset/val.csv")

train_count_amino, train_count_1, train_count_0 = gen_all_feature("dataset/train_seq.csv", "dataset/train.csv")

test_count_amino, test_count_1, test_count_0 = gen_all_feature("dataset/test_seq.csv", "dataset/test.csv")
```

	name	feat	label
0	L	[0, 1, 0, 0, 0, 0, 8, 0, 2, 7.0, 3.8, 1, 0, 0, 1, 0, 0, 0, -1, 0, -1, 2, -6, 3, 5, -4, -1, -4, -4, -1, -2, -4, -4, -2, -1, -6, -1, 0]	0
1	P	[0, 1, 0, 0, 0, 0, 7, 0, 2, 7.0, -1.6, 1, 0, 0, 0, 0, 0, 1, -1, -6, -3, -7, -1, -6, -6, -5, -2, 0, -2, -6, 1, 6, -4, -3, -5, 5, 7, -3]	0

8. 投入网络训练之前，需要构造样本生成器，也就是将数据都投入到Dataset这个类中初始化，每次产生一个样本（38维向量）用于训练，最后同时返回样本和标签；

每次调用都获取一个样本的索引，其上界由 __len__ 方法确定，也就是在训练集，测试集和验证集划定好上限

```python
class Dataset(data.Dataset):

    def __init__(self, list_IDs, labels, df_feat):
        'Initialization'
        self.labels = labels
        self.list_IDs = list_IDs
        self.df = df_feat

    def __len__(self):
        'Denotes the total number of samples'
        return len(self.list_IDs)

    def __getitem__(self, index):
        'Generates one sample of data'
        # Select sample
        # Load data and get label
        # index = self.list_IDs[index]
        #print(self.df.iloc[index]['feat'])
        feat = eval(self.df.iloc[index]['feat'])        # 特征索引

        y = self.labels[index]

        return np.array(feat), y
```

9. 模型训练方面，挑选展示表现最好的CNN-DNN模型，设计得较为简单，但1维卷积层的使用已经足以提取特征，右侧是最佳测试结果

```python
class CNN(nn.Sequential):
    def __init__(self, **config):
        super(CNN, self).__init__()

        self.conv = nn.Conv1d(38, 16, 1)

        self.decoder = nn.Sequential(
            nn.Linear(config['flatten_dim'], 4),
            nn.ReLU(True),

            nn.BatchNorm1d(4),
            nn.Linear(4, 1),

            nn.Sigmoid()
        )

    def forward(self, feat):
        feat = feat.view(config['batch_size'], 38, 1)
        # print("init_feat: ", feat.shape)

        feat = self.conv(feat)
        # print("conv_feat: ", feat.shape)

        feat = feat.view(config['batch_size'], -1)
        # print("view_feat: ", feat.shape)
        feat = self.decoder(feat)
        # print("result_feat: ", feat.shape)
        return feat
```

```
--- Go for Testing ---
[Testing metrics]: auc:0.6939, auprc:0.6875, f1:0
.6257, accuracy:0.6410, recall:0.7739, precision:
0.5981
(base) aita@aita-CVN-Z590-GAMING-PRO:~/4444/east/
protein-rnn/model$
```

10. 指标计算的实现方法

```python
def test(data_generator, model):
    y_pred = []
    y_label = []
    model.eval()
    loss_accumulate = 0.0
    count = 0.0

    for i, (feat, label) in enumerate(tqdm(data_generator)):

        score = model(feat.to(device).float())

        logits = torch.squeeze(score)
        loss_fct = torch.nn.BCELoss()

        label = Variable(torch.from_numpy(np.array(label)).to(device).float())

        loss = loss_fct(logits, label)

        loss_accumulate += loss
        count += 1

        logits = logits.detach().cpu().numpy()

        label_ids = label.to('cpu').numpy()

        y_label = y_label + label_ids.flatten().tolist()
        y_pred = y_pred + logits.flatten().tolist()

    loss = loss_accumulate / count

    fpr, tpr, thresholds = roc_curve(y_label, y_pred)

    precision = tpr / (tpr + fpr + 0.00001)
```

交叉熵损失函数是二分类常用的loss函数

根据公式实现的指标计算：

F1score = 2 * precision * recall precision + recall

ACC = (TP+TN)/ (TP+FP+FN+TN)

TPR = TP/(TP+FN)

FPR = FP/(FP+TN)

而roc_curve()可以自动作图，求出AUC（TPR的求和/FPR的求和）

```python
    f1 = 2 * precision * tpr / (tpr + precision + 0.00001)
    thred_optim = thresholds[1:][np.argmax(f1[1:])]
    y_pred_s = [1 if i else 0 for i in (y_pred >= thred_optim)]
    cm1 = confusion_matrix(y_label, y_pred_s)
    total1 = sum(sum(cm1))
    accuracy1 = (cm1[0, 0] + cm1[1, 1]) / total1

    outputs = np.asarray([1 if i else 0 for i in (np.asarray(y_pred) >= 0.5)])
    return roc_auc_score(y_label, y_pred), average_precision_score(y_label, y_pred), f1_score(y_label, outputs), \
           accuracy1, recall_score(y_label, y_pred_s), precision_score(y_label, y_pred_s)
```

11. AUC得到最大值的模型将得到保存，这样就不用人工翻看记录了
最后清空一下cache，准备下一次训练

```
123     with torch.set_grad_enabled(False):
124         auc, auprc, f1, acc, rec, pre = test(validation_generator, model)
125         print("[Validation metrics]: auc:{:.4f}, auprc:{:.4f}, f1:{:.4f}, accuracy:{:.4f}, recall:{:.4f}, precision:{:.4f}".format(auc, auprc, f1, acc, rec, pre))
126 
127         if auc > max_auc:
128             torch.save(model, 'save_model/model.pth')
129             model_max = copy.deepcopy(model)
130             max_auc = auc
131             print("*" * 30 + " save best model " + "*" * 30)
132 
133     torch.cuda.empty_cache()
```

12. 此处可给模型调参和切换训练模型，后期写入新的模型时也可以保留该框架，不过要注意学习率和批大小的调整，学习率太大和太小都会使梯度崩溃；批大小过大会爆显存，过小影响训练效果

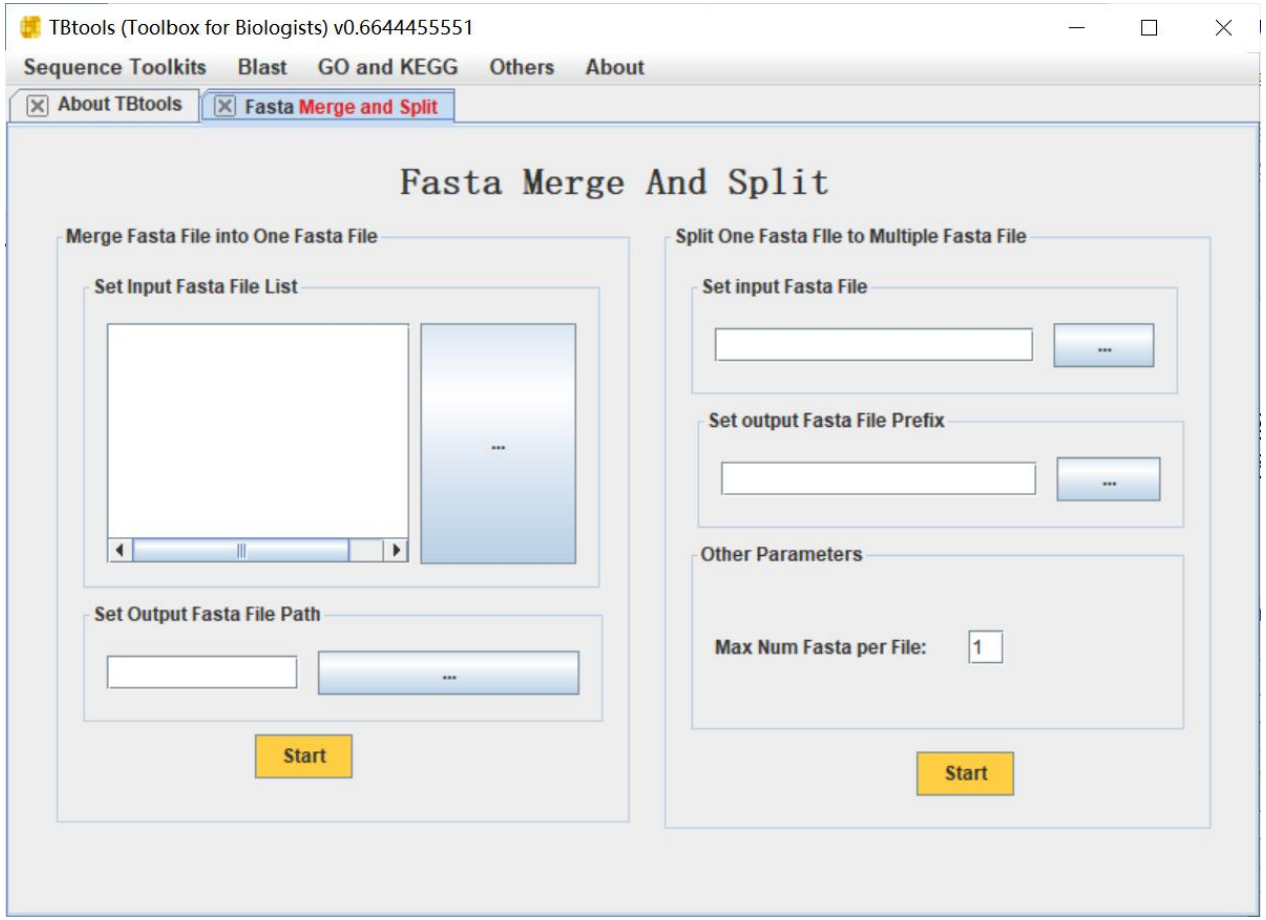
```python
def Bin_config():
    config = {}

    config['batch_size'] = 64
    config['num_workers'] = 0
    config['epochs'] = 30
    config['lr'] = 0.01

    config['model'] = 'cnn_maxpool'        # cnn, lstm, gru, transformer, cnn_maxpool


    if config['model'] == 'lstm' or config['model'] == 'gru':
        config['flatten_dim'] = 1216
    elif config['model'] == 'cnn' or config['model'] == 'cnn_maxpool':
        config['flatten_dim'] = 16
    elif config['model'] == 'transformer':
        config['flatten_dim'] = 38
    return config
```

13. 其他技术细节：fasta文件的拆分，由于我们得到的数据集是150个fasta在同一个文件，这里采用了TBtools工具拆分fasta为150个文件，方便送进uniref数据库单个（与库中其他序列）比对



14. 其他技术细节：值得注意的是，PaleAle返回的二级结构和溶剂可及性结果由于文本框的限制，各个蛋白链并不是严格的成一行，下方有二级结构和可及性对齐（如下图，长的链有3-4行序列，短的才是严格的1行），我们需要另外编写脚本，另起三行来分别存放序列，二级结构和可及性，见下页代码

output.txt

```
Query_name: 1A34
Query_length: 147
TGDNSNVVTMIRAGSYPKVNPTPTWVRAIPFEVSVQSGIAFKVPVGSLFSANFRTDSFTSVTVMSVRAWTQLTPPVNEYSFVRLKPLFKTGDSTEEFEGRASNINTRASVGYRIPTNLRQNTVAADNVCEVRSNCRQVA
LVISCCFN
CCCCCCEEEEEECCCCCCCCCCCCCEEEEEEEEEECCEEEEECCCCEEEEEECCCEEEEEEEEEEEEECCCCCCEEEEEEEECCCCCCCCEEEEEECCCCCCEEEEEEECCCCCCCCEECCCEEEECCCCEEE
EEEEEECC
EEEEEeBBeBBeEEEbEEEEEEbEbBeBBeBeBbBEeeBBBeBeEeeBBebEBEEEEbEeBbBBbBeBbbEbEEEEEEbbBBeBeEbbEEEEeEEEBEEeBEEbEeEBeBBbEBbEEBEEEeEeBEeBBeBeeeBeeBB
BBBBBbbE
Query_name: 1A9N
Query_length: 94
IRPNHTIYINNMNDKIKKEELKRSLYALFSQFGHVVDIVALKTMKMRGQAFVIFKELGSSTNALRQLQGFPFYGKPMRIQYAKTDSDIISKMRG
CCCCCEEEEECCCCCCCHHHHHHHHHHHHHHCCCEEEEEECCCCCCCCCEEEEEECCHHHHHHHHHHHCCCEECCEEEEEEECCCCCHHHHHCCC
EEEEeeBbBbEbeEEbeeEeBeEbBeEbEbeeBebeEEEEeEbbBbBbBEeeEbBeeBbEebEEeEbEEeeBeBebbEEEEEEEEEEEE
Query_name: 1ASY
Query_length: 490
EDTAKDNYGKLPLIQSRDSDRTGQKRVKFVDLDEAKDSDKEVLFRARVHNTRQQGATLAFLTLRQQASLIQGLVKANKEGTISKNMVKWAGSLNLESIVLVRGIVKKVDEPIKSATVQNLEIHITKIYTISETPEALPI
LLEDASRSEAEAEAAGLPVVNLDTRLDYRVIDLRTVTNQAIFRIQAGVCELFREYLATKKFTEVHTPKLLGAPSEGGSSVFEVTYFKGKAYLAQSPQFNKQQLIVADFERVYEIGPVFRAENSNTHRHMTEFTGLDMEM
AFEEHYHEVLDTLSELFVFIFSELPKRFAHEIELVRKQYPVEEFKLPKDGKMVRLTYKEGIEMLRAAGKEIGDFEDLSTENEKFLGKLVRDKYDTDFYILDKFPLEIRPFYTMPDPANPKYSNSYDFFMRGEEILSGAQ
RIHDHALLQERMKAHGLSPEDPGLKDYCDGFSYGCPPHAGGGIGLERVVMFYLDLKNIRRASLFPRDPKRLRP
CCCCCCCCCCCCCCCCCCCCCHHHHHHHHHCCCCCHCCCCEEEEEEEEEEEECCCCEEEEEEEECCCEEEEEECCCCCCCCCCHHHHHHHHCCCCCCEEEEEEEEEECCCCCCCCCCCEEEEEEEEEEECCCCCCCCCC
CCCCCCCCCHHHHCCCCCCCCCCHHHHHHHCCCCCCCCHHHHHHHHHHHHHHHHHHHHHHHHCCCEEEECCEEECCCCCCCCCCCCCECCCCCCEEECCCHHHHHHHHHHCCCCCCEEEEECCCCCCCCCCCCCCCHHHHHHHHH
HHHHHHHHHHHHHHHHHHHHHHHHHHHHCCCHHHHHCCCCCCHHHCCCCCCCCCCCCCCCHHHHHHHHHHCCCCCCCCCCCCCCHHHHHHHHHHHHHHHCCCCCEEECCCCCCCCCHHHCCCCCCCCCCCEEEEEEECCCCCEEECCCH
HCCCHHHHHHHHHHCCCCHHHHHHHHHHHHHHHCCCCCCCCCCCCHHHHHHHHHCCCCHHHHCCCCCCCCCCCCC
EEEEEEEbEEbEbbeEEEEEebEebbbebbEbEEEEeeEeeBbBBBbBbbbbEEEbbBBBBBBbbeeBbBBBBBbeEeEEEbeEEBbEbBEEBebBBBBBBbBeBeebEEEEEEbEeeeBbBbBeeBeBBBeBbEEbbb
ebEEEEEEEEEEEEEEbEEbeeEbbbebBBBbBbbEEBeebBebBBeBBebBbebBeEeEBBbBbBBbBbebEBbBBbEeBeBebbEeEBbBbBBBBBBBbBBBBBBBebBBBBBbBBbbbEbEbebBBBBBBBBBBBBB
BBbebBeeBBebBbeBBbbBBeeBbEEbEEEbEeBeeEbEbEebEbbEEbEbeebebEeBbEbBeEbEeEbEEEbEbbEeBbEbBbebBeEEbbEeBBBBbebBeeBeBBbbeEbEEeEeBBbBBBBBBbBbBBBBBBB
bBbebEeBbEbBEEeEbEEEeEbBebBBeBBebBBBbBBBBBBBBBBBBBBBBEbEbBbbBBBBBebeEebEE
Query_name: 1AV6
Query_length: 289
```

```python
if __name__ == '__main__':

    outputs = open('output.txt', 'w', encoding='utf-8')

    line1 = ''
    line2 = ''
    line3 = ''
    linenum = 0
    for line in load_file('二级结构+可及性.txt'):

        if (len(line)>0) :
            if (line.startswith('Query_name')):
                if (linenum > 0):
                    print(line1)
                    outputs.write(line1 + '\n')
                    print(line2)
                    outputs.write(line2 + '\n')
                    print(line3)
                    outputs.write(line3 + '\n')

                print(line)
                outputs.write(line + '\n')
                continue
            if (line.startswith('Query_length')):
                print(line)
                outputs.write(line + '\n')
                line1 = ''
                line2 = ''
                line3 = ''
                linenum = 0
                continue
            if (linenum%3)==0:
                linenum = linenum+1
                line1 = line1+line
            elif (linenum%3)==1:
                linenum = linenum + 1
                line2 = line2+line
            elif (linenum%3)==2:
                linenum = linenum + 1
                line3 = line3+line

    if (linenum > 0):
        print(line1)
        outputs.write(line1 + '\n')
        print(line2)
        outputs.write(line2 + '\n')
        print(line3)
```

新建文件，另起三行等待存放序列，二级结构和可及性

把上述三个空行写入output.txt，进行占位

仔细观察发现，序列所在的行正好是3的倍数，而二级结构和可及性分别被3除余1,2，所以我们用load_file的line读行，就可以把序列，二级结构和可及性分行打出来，方便后续转化成独热编码

Query_name: 1ASY
Query_length: 490

EDTAKDNYGKLPLIQSRDSDRTGQKRVKFVDLDEAKDSDKEVLFRARVHNTRQQGATLAF
CCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHCCCCCHCCCCEEEEEEEEEEECCCCEEEE
EEEEEEEbEEbEbbeEEEEEebEebbbebbEbEEEEeeEeeBbBBBbBbbbbEEEbbBBB

LTLRQQASLIQGLVKANKEGTISKNMVKWAGSLNLESIVLVRGIVKKVDEPIKSATVQNL
EEEECCCEEEEEEECCCCCCCCCHHHHHHHHHCCCCCCEEEEEEEEECCCCCCCCCCCEE
BBBbbeeBbBBBBBbeEeEEEbeEEBbEbBEEBebBBBBBBbBeBeebEEEEEEbEeeeB

EIHITKIYTISETPEALPILLEDASRSEAEAEAAGLPVVNLDTRLDYRVIDLRTVTNQAI
EEEEEEEEEECCCCCCCCCCCCCCCCHHHHCCCCCCCCCHHHHHHHHCCCCCCCHHHHHH
bBbBeeBeBBBeBbEEbbbebEEEEEEEEEEEEEbEEbeeEbbbebBBBbBbbEEBeeb

FRIQAGVCELFREYLATKKFTEVHTPKLLGAPSEGGSSVFEVTYFKGKAYLAQSPQFNKQ
HHHHHHHHHHHHHHHHHHCCCEEEECCEEECCCCCCCCCCCCECCCCCCEEECCCHHHHHH
BebBBeBBebBbebBeEeEBBbBbBBbBbebEBbBBbEeBeBebbEeEBbBbBBBBBBBb

15. 其他信息：1）氨基酸的理化性质编码

	原子数	静电荷值	氢键数	侧链pKa值	疏水性参数
Ala	5	0	2	7.00	1.8
Cys	6	0	2	7.00	2.5
Asp	8	-1	4	3.65	-3.5
Glu	9	-1	4	3.22	-3.5
Phe	11	0	2	7.00	2.8
Gly	4	0	2	7.00	-0.4
His	10	1	4	6.00	-3.2
Ile	8	0	2	7.00	4.5
Lys	9	1	2	10.53	3.9
Leu	8	0	2	7.00	3.8
Met	8	0	2	7.00	1.9
Asu	8	0	4	8.18	-3.5
Pro	7	0	2	7.00	-1.6
Gln	9	0	4	7.00	-3.5
Arg	11	1	4	12.48	-4.5
Ser	6	0	4	7.00	-0.8
Thr	7	0	4	7.00	-0.7
Val	7	0	2	7.00	4.2
Trp	14	0	3	7.00	-0.9
Tyr	12	0	3	10.07	-1.3

2）氨基酸类别信息编码

类别	所含的氨基酸	偶极矩和侧链体积特征
1	Ala,Gly, Val	偶极矩小于1.0德拜，侧链体积大于50 $Å^3$
2	Ile, Leu, Phe, Pro	偶极矩小于1.0德拜，侧链体积小于50 $Å^3$
3	Tyr, Met, Thr, Ser, Cys	偶极矩在1.0~2.0德拜之间，侧链体积小于50 $Å^3$
4	His, Asn, Gln, Trp	偶极矩在2.0~3.0德拜之间，侧链体积小于50 $Å^3$
5	Arg, Lys	偶极矩大于3.0德拜，侧链体积小于50 $Å^3$
6	Asp, Glu	偶极矩大于3.0德拜且具有不同方向，侧链体积小于50 $Å^3$

3）38维向量信息总览（下方以苯丙氨酸为例，这是它的38维度特征向量）

序号	特征
1-6	氨基酸所属类别
7	氨基酸原子数
8	氨基酸静电荷值
9	氨基酸所含氢键数
10	氨基酸侧链pKa值
11	氨基酸疏水性参数
12-14	氨基酸所属的蛋白质二级结构
15-18	氨基酸的相对溶剂可及性指标（0-9）
19-38	氨基酸进化信息
36	氨基酸结合残基判定指标（**数值为1表示该氨基酸与RNA相互作用，为0则不与RNA发生相互作用**）

L [0, 1, 0, 0, 0, 0, 8, 0, 2, 7.0, 3.8, 1, 0, 0, 1, 0, 0, 0, -1, 0, -1, 2, -6, 3, 5, -4, -1, -4, -4, -1, -2, -4, -4, -2, -1, -6, -1, 0]