数字信号处理技术与应用（AU3318）大作业

By： 叶博文 521030910041

一. 读取txt文件

在下载好数据之后运行readtxt.m文件，得到了声音信号的所有原始数据，后缀为.mat文件。

二. 时域上分离每次敲击信号并选择合适的信号长度

首先我根据选择阈值的方法来找到信号的每一个极大值附近的点并将它们存储起来（由于阈值设置恰当导致找的点位置即使不是极大值也和极值位置接近），之后选择采样点个数为512（选择2的幂次是为了进行fft变换的时候方便且迅速），对于之前所选择的极值位置的点分别向前取128个点，向后取384个点。由于采样频率是51.2KHZ，因此时域中信号长度 $L = 512 / 51.2*10^3 = 10$ms。

而我所提交的代码中data是一些处理过的或者未处理的声音信号，比如Xraw.mat，其中 $X = [400,800,1200...4000]$,或者是声音信号进行了fft变化之后得到的一些信号，这些都是以list的形式存储在data中，后缀为.npy，存储代码如下所示：$np.save('data/fft_{}.npy'.format(j), amp_list)$。

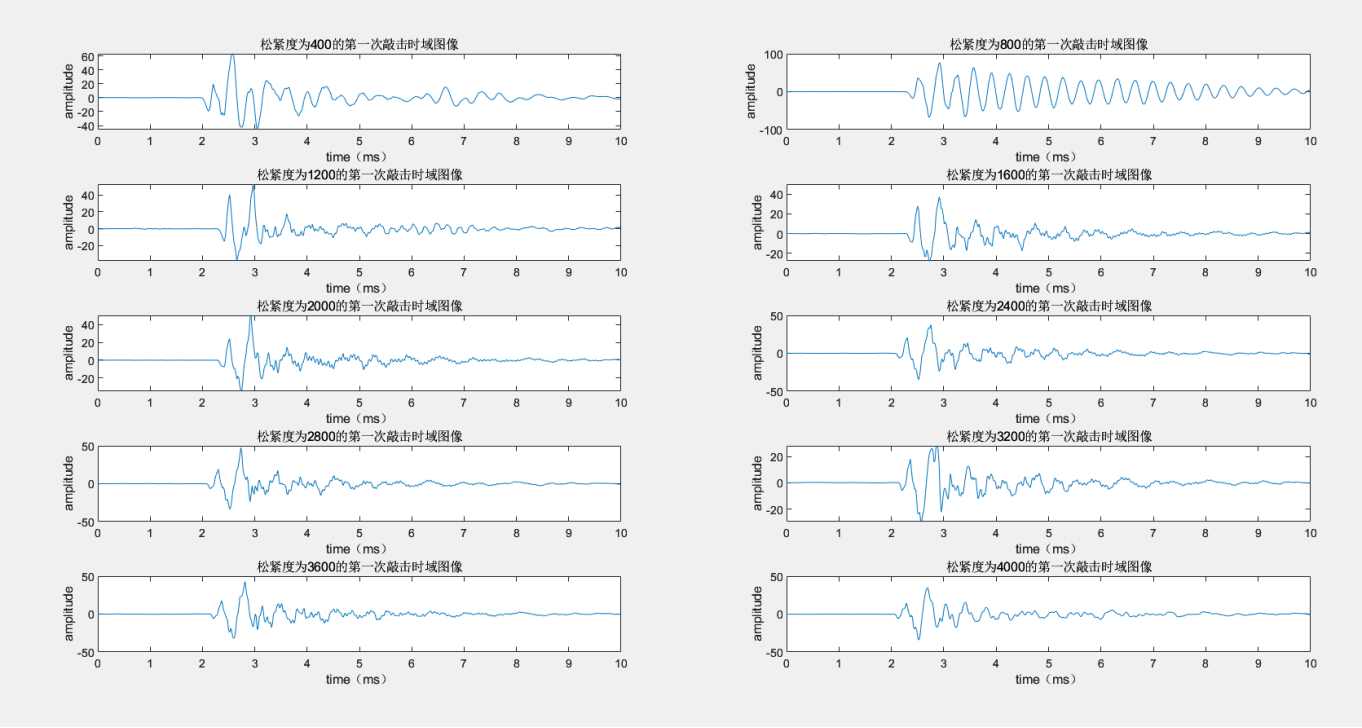
而除了.py文件之外，我还提交了run.m文件，这是一个matlab代码，其中我注释掉了一些部分，那是可以听一下每一种松紧度的声音的分割之后的敲击信号的，代码如下所示：

```
for i = 1:length(separatedSignals)
    sound(separatedSignals{i}, 4410);
    pause(4);  % 可以根据需要调整每段音频之间的暂停时间
end
sound(audioSignal, 51200);  % 采样率，你需要根据实际情况替换为你的采样率
```

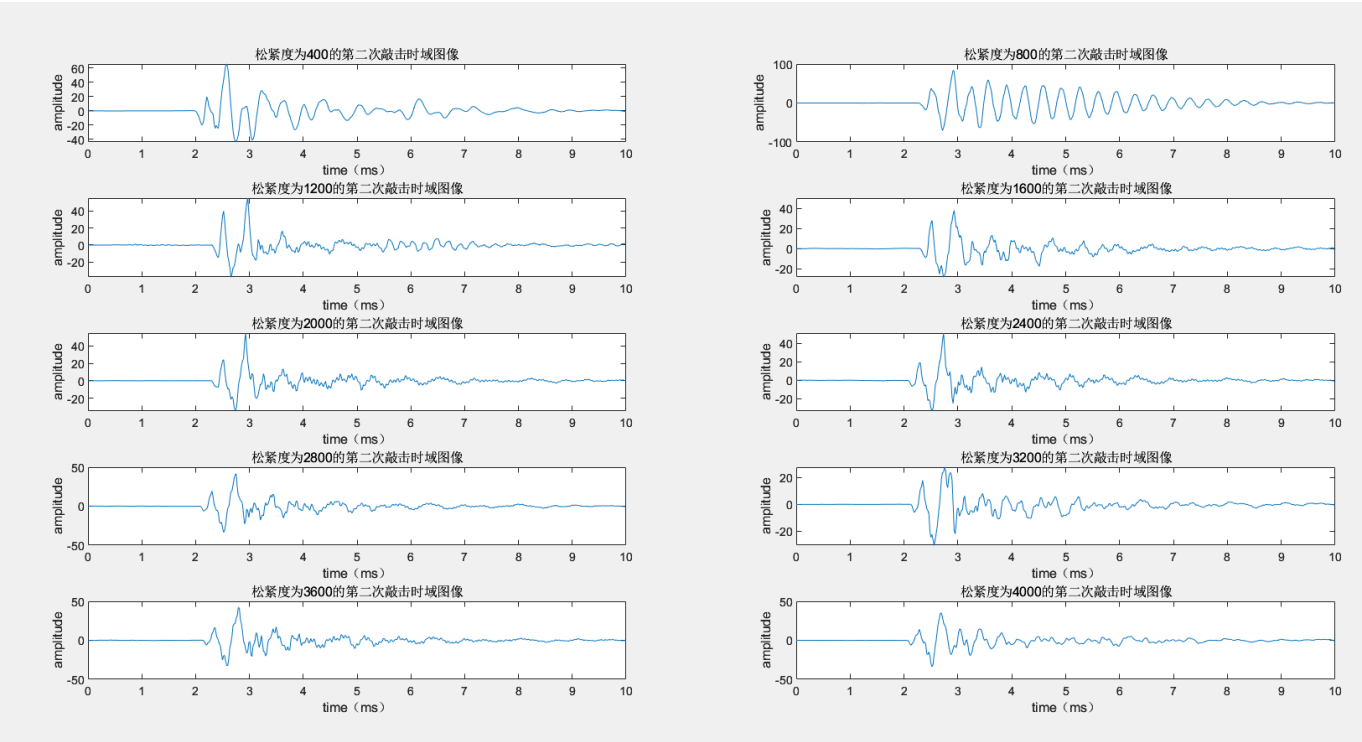
而以下代码（目前运行run.m）会显示第一次敲击时的时域图像：

```
figure;
time = (0:length(separatedSignals{1})-1) / 51200;  % 假设采样率为 44100 Hz
plot(time, separatedSignals{1});
xlabel('Time (seconds)');
ylabel('Amplitude');
title('800 Time Domain Plot of the First Impact');
```

第一次敲击的图像如下所示：



而第二次敲击的图像如下图所示：



三. 对选定的信号进行频域变化

在代码中我实现了三种方法(其中 $X = [400,800,1200...4000]$)，分别是分割后的时间信号，命名为sound_X.npy,对时间信号进行fft变换之后的信号，命名为fft_X.npy，和对信号进行mfcc变换之后得到的结果，命名为mfcc_X.npy。其中前二者的代码在store.py中（store.py中还有读取raw信号并进行分离的过程），最后的mfcc信号的变换和存储在mfcc.py中。进行变化的代码分别如下所示：

```
for i, segment in enumerate(separated_signals):
    fft_result = fftn(segment)
    amplitude = np.abs(fft_result)
```

```
    total_emerge = np.sum(amplitude**2)
    tmp.append(np.max(amplitude))
    train_set.append(tmp)
    train_target.append(label)
    amp = amp + amplitude
    amp_list.append(amplitude)
np.save('data/fft_{}.npy'.format(j), amp_list)
```

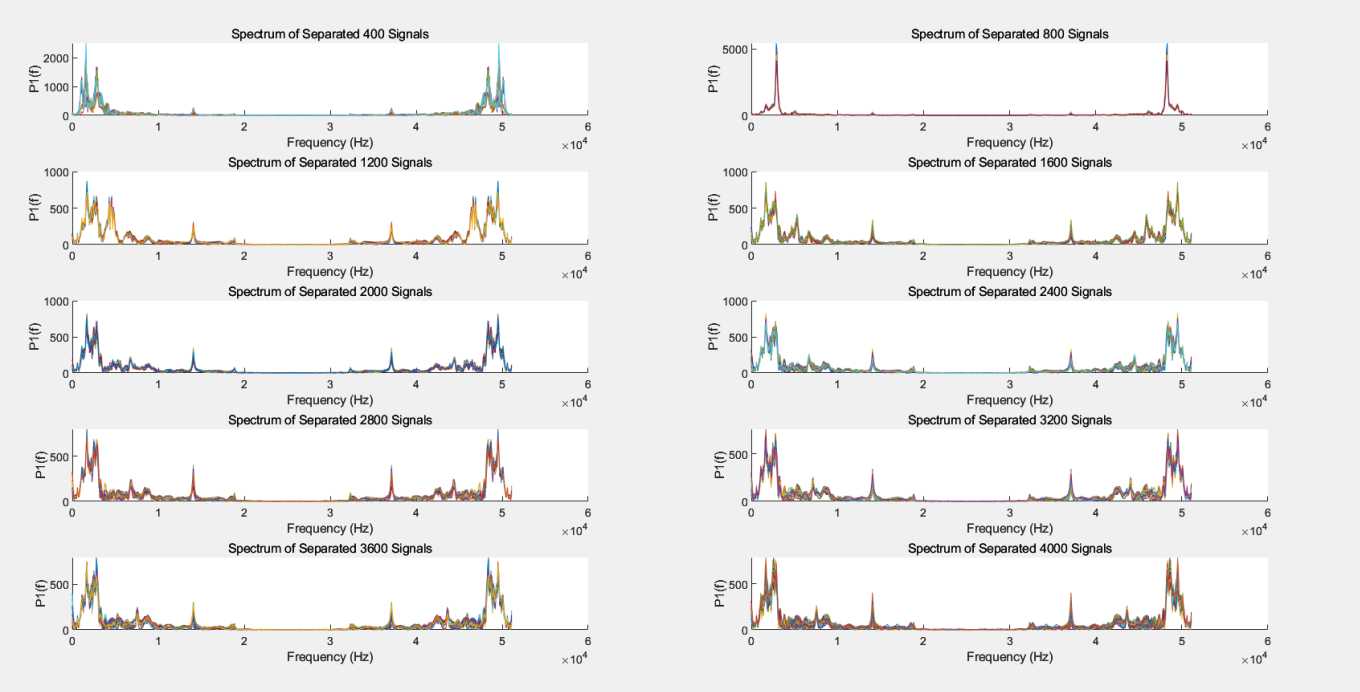
和如下所示：

```
mfcc_lis = []
for i in range(len(data)):
    pre_emphasis_data = librosa.effects.preemphasis (data[i], coef=0.97,
zi=np.zeros((len(data[i]), 1), dtype=data[i].dtype))

    pre_emphasis_data = pre_emphasis_data.flatten()
    frames = librosa.util.frame(pre_emphasis_data, frame_length=400,
hop_length=160)

    mfcc = librosa.feature.mfcc(y=pre_emphasis_data, sr=16000, n_mfcc=13,
hop_length=160)
    mfcc_lis.append(mfcc.tolist())
np.save('data/mfcc_{}.npy'.format(m), mfcc_lis)
```

而run.m代码下面部分，即以下所示：

```
subplot(5, 2, k);
for i = 1:length(separatedSignals)
    segment = separatedSignals{i};
    fftResult = fft(segment);

    % 计算频谱的振幅
    amplitude = abs(fftResult);

    % 计算对应的频率
    fs = 51200;  % 采样率，根据实际情况调整
    f = (0:length(segment)-1) * fs / length(segment);

    % 绘制频谱图，使用 hold on 保持当前图形
    hold on;
    plot(f, amplitude, 'DisplayName', ['Signal ' num2str(i)]);
end
xlabel('Frequency (Hz)');
ylabel('P1(f)');
title(sprintf('Spectrum of Separated %d Signals', pos));
```

运行(显示频谱图像的时候需要将时域的图像显示注释并解注释频谱的图像显示)之后显示出所有松紧度下的所有敲击信号的幅值频谱迭加图如下所示：



四. 比较并分析不同松紧度信号的特征

首先提取一些物理意义上的特征进行分类（其提取特征和拟合直线的代码在fit.py当中）。

频谱宽度

运行之后显示如下所示：

	feature	Y
0	0.058594	400
1	0.113281	800
2	0.062500	1200
3	0.062500	1600
4	0.042969	2000
5	0.042969	2400
6	0.042969	2800
7	0.042969	3200
8	0.042969	3600
9	0.042969	4000

可以发现对于松紧度小于1200的其频谱宽度还是有一些差异的，二对于2000-4000的松紧度的信号，其频谱宽度几乎相同。因此这个物理意义上的特征是不恰当的。因此我又做了以下尝试。

熵

我又提取了信号的熵作为特征(需要注意的是当计算熵的时候需要将读取的信号改为fft.npy)，结果如下所示：

	feature	Y
0	7.495522	400
1	7.000595	800
2	7.979904	1200
3	8.085309	1600
4	8.129133	2000
5	8.138310	2400
6	8.130649	2800
7	8.114822	3200
8	8.106713	3600
9	8.107373	4000

可以看到熵作为特征的时候其效果较好，但是对于2800-4000的松紧度的信号，其效果略差，可能会和1600-2400信号混淆。

能量的加权集中度

然后我尝试了能量的加权集中度，其结果如下：

	feature	Y
0	-3.391586e-09	400
1	-1.669656e-09	800
2	-3.210190e-09	1200
3	-5.487420e-09	1600
4	-4.766006e-09	2000
5	-4.547979e-09	2400
6	-4.370261e-09	2800
7	-4.738336e-09	3200
8	-5.471656e-09	3600
9	-5.515402e-09	4000

可以发现其特征分类效果较好，最后的拟合结果也较好，具体看下文。

总能量

最后我尝试了能量，由于时域和频域结果几乎差不多（指的是辨别度），因此以下我仅展示时域的结果：

	feature	Y
0	36376.380072	400
1	180602.463048	800
2	23872.662911	1200
3	9660.431955	1600
4	14814.884224	2000
5	15418.786091	2400
6	15521.118815	2800
7	14181.295743	3200
8	14253.014501	3600
9	14764.998077	4000

拟合曲线见五。

五. 提取特征，分类并计算分类精度

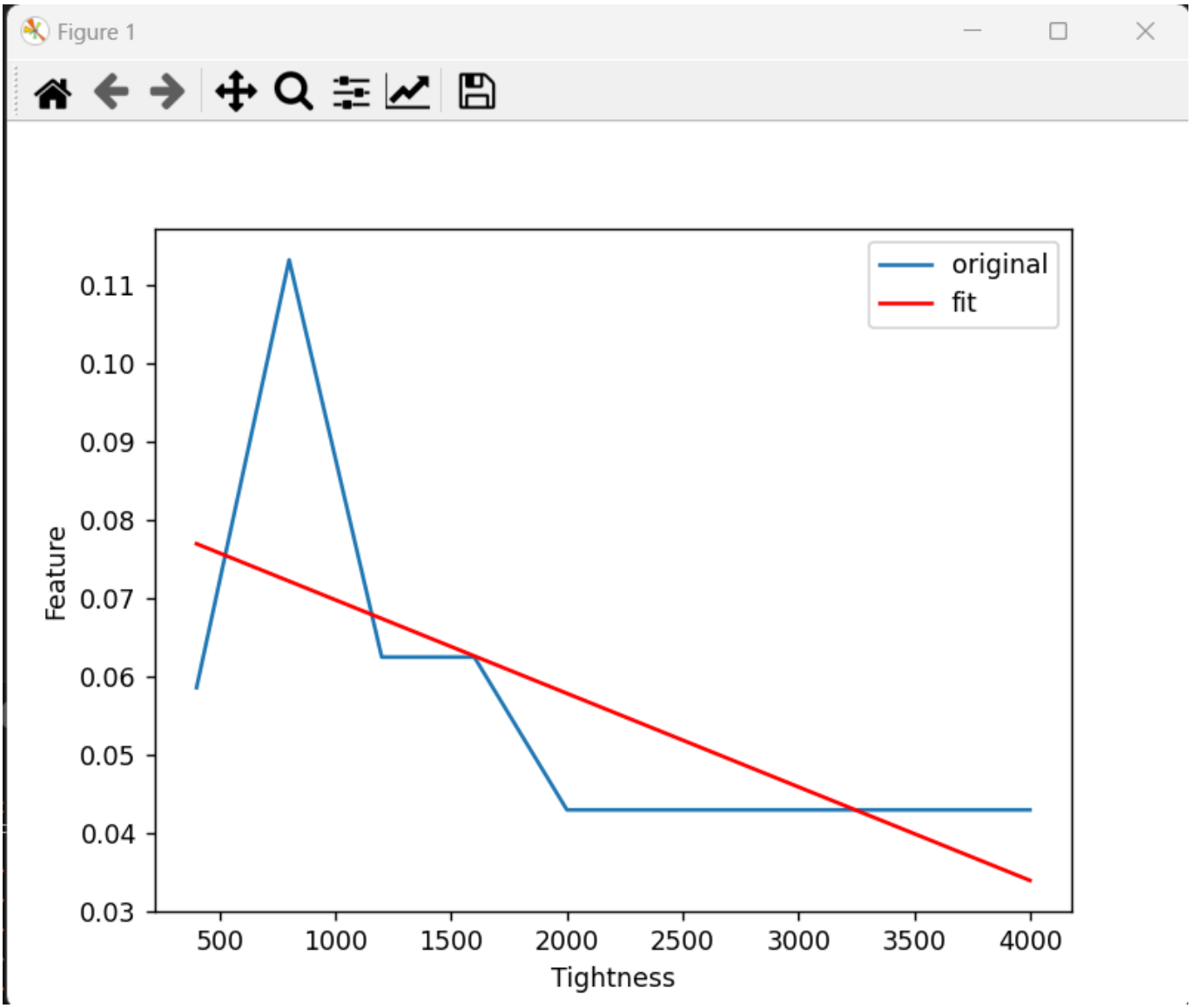
在这里我采用传统方法和机器学习方法。

传统方法

由于传统方法得到的特征并不是很恰当，因此对于传统方法我只进行特征提取和拟合曲线，并不进行分类和计算分类精度。

频谱宽度

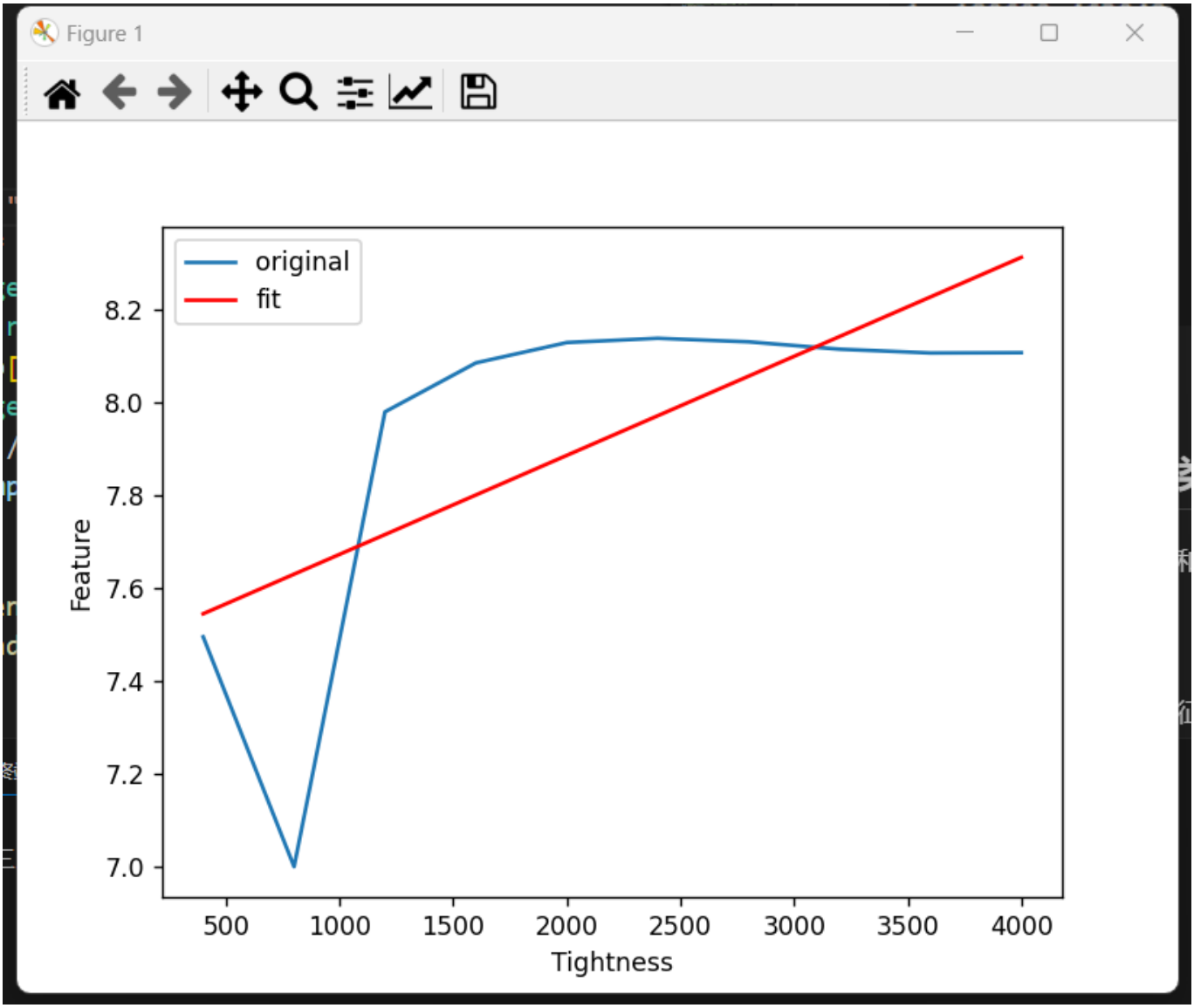
一阶拟合曲线如下所示：

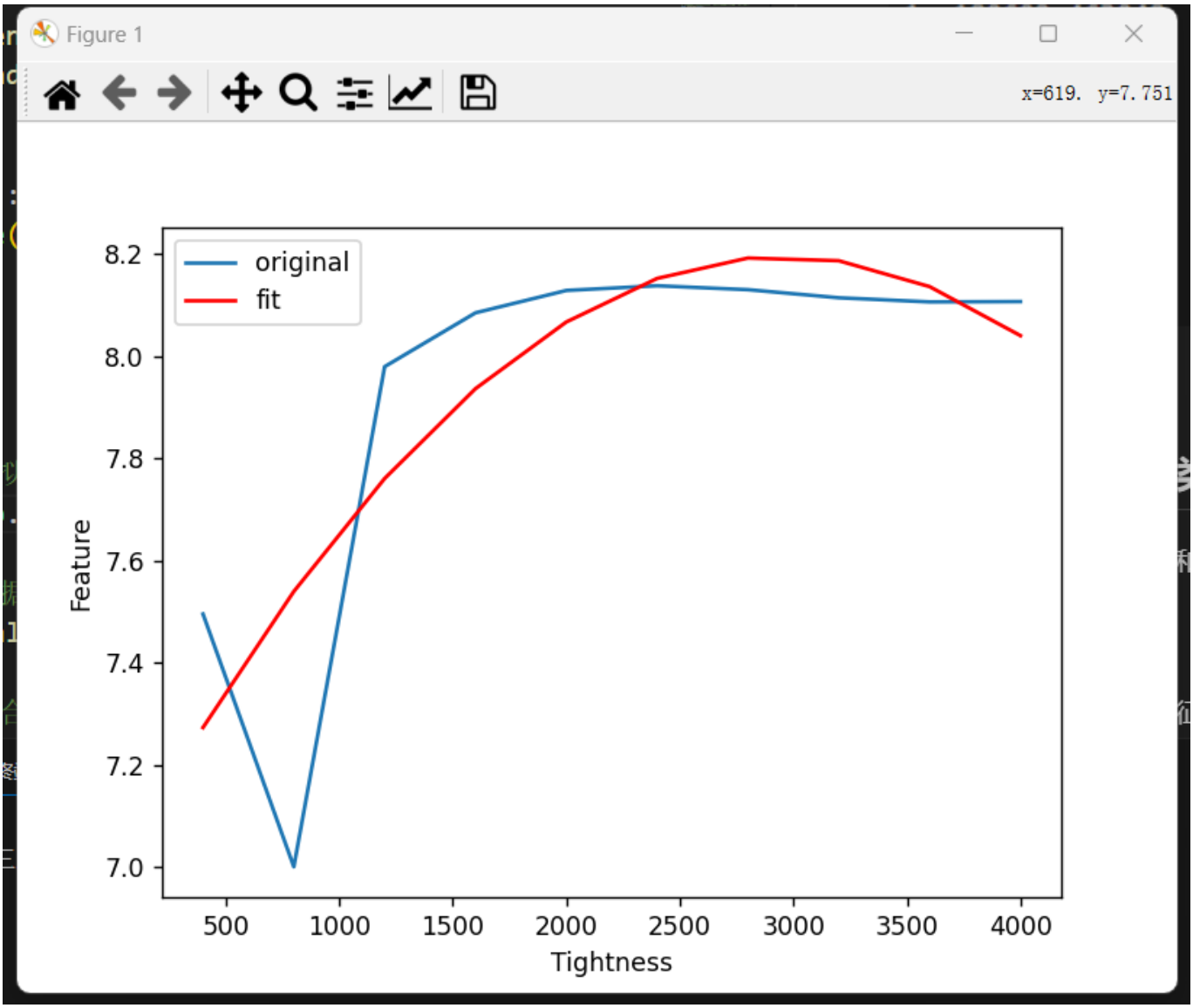


其二阶拟合曲线在可以区域仍和一阶类似，故不展示。

熵

接着是熵的一阶拟合曲线和二阶拟合曲线，分别如下所示：



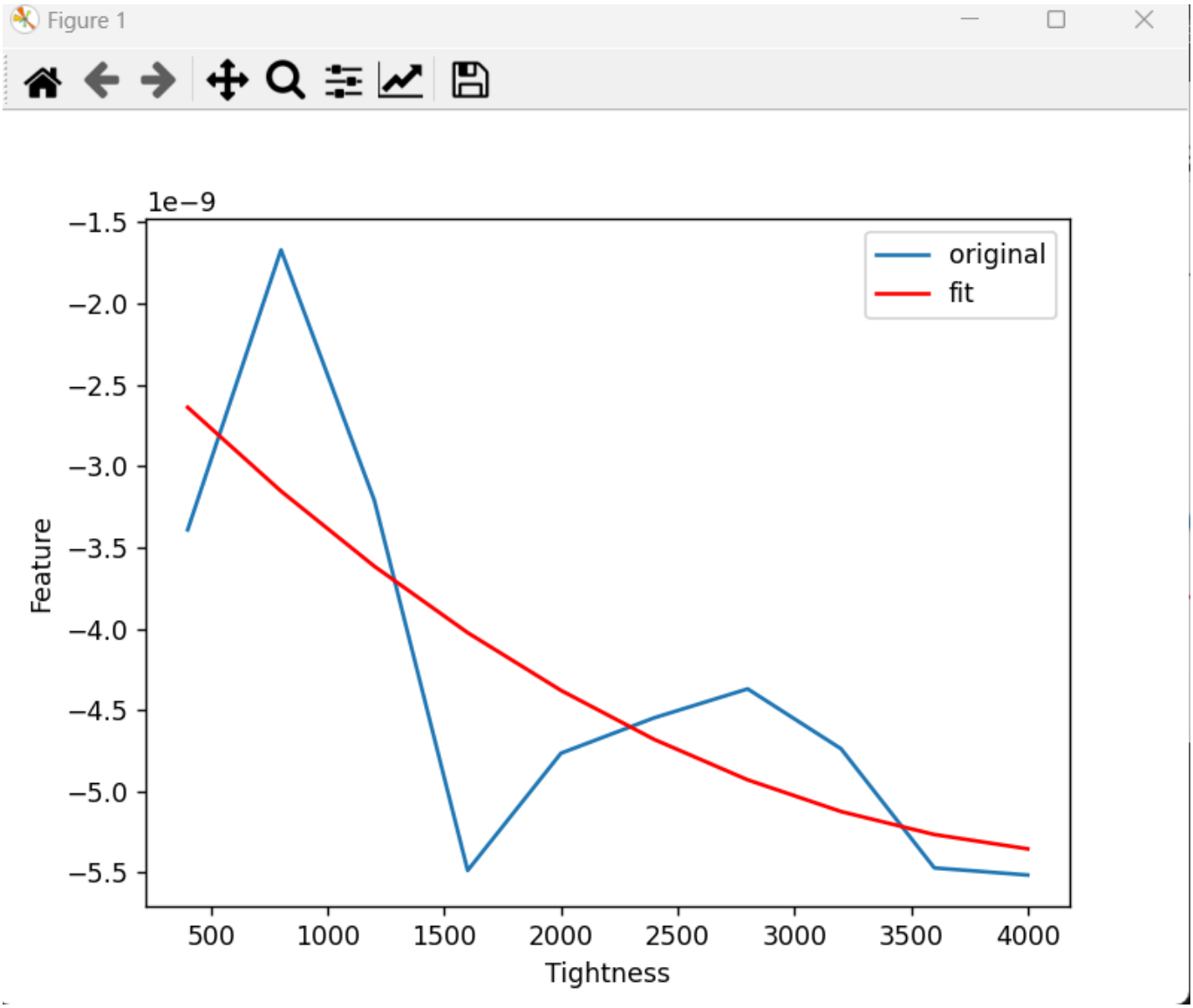


能量的加权集中度

代码如下所示：

```
def center_energy(signal):
    spectrum = np.fft.fft(signal)
    # 有负是正常的，正一半负一半
    freq = np.fft.fftfreq(len(signal))
    # 计算频谱的幅度谱（能量）
    magnitude_spectrum = np.abs(spectrum)
    # 计算能量加权的平均频率
    weighted_average_frequency = np.sum(freq * np.abs(magnitude_spectrum)) /
np.sum(magnitude_spectrum)
    # 计算频谱的总能量
    total_energy = np.sum(magnitude_spectrum)
    # 计算能量集中度
    energy_concentration = weighted_average_frequency / total_energy

return energy_concentration
```

拟合曲线（能量的拟合曲线可以运行代码得到），如下所示：



机器学习

在代码提交中我采用了五种方法，其代码分别为classifier_KNN.py,classifier_LSTM.py,classifier_NN.py,classifier_SVM.py（后面两个的效果比较好）,其命名就代表了我使用的方法，而最后我创新得到了一个方法，代码是classifier_mix.py，其使用了KNN分类器，创新点在于我将同一个信号的fft和mfcc变化之后的信号拼接组成新的信号代表原信号，这样可以使得信号特征更明显，分类效果更好。以下我展示三种方法。在每种方法中，我留取所有信号的四个作为测试集，剩下的全部是训练集。

SVM分类器

首先是SVM分类器，其通过以下代码创建：

```
clf = svm.SVC(kernel='linear')
```

最后运算得到对于声音信号的分类结果如下所示,正确率为0.95：

```
Classification Report:
              precision    recall  f1-score   support

         400       1.00      0.75      0.86         4
         800       1.00      1.00      1.00         4
        1200       1.00      1.00      1.00         4
        1600       1.00      1.00      1.00         4
        2000       0.80      1.00      0.89         4
        2400       1.00      1.00      1.00         4
        2800       1.00      1.00      1.00         4
        3200       1.00      1.00      1.00         4
        3600       1.00      0.75      0.86         4
        4000       0.80      1.00      0.89         4

    accuracy                           0.95        40
   macro avg       0.96      0.95      0.95        40
weighted avg       0.96      0.95      0.95        40
```

而对于fft变化之后的信号分类结果如下所示，正确率为0.88：

```
Classification Report:
              precision    recall  f1-score   support

         400       1.00      1.00      1.00         4
         800       1.00      1.00      1.00         4
        1200       1.00      1.00      1.00         4
        1600       1.00      1.00      1.00         4
        2000       1.00      1.00      1.00         4
        2400       1.00      1.00      1.00         4
        2800       0.80      1.00      0.89         4
        3200       1.00      0.75      0.86         4
        3600       0.50      0.50      0.50         4
        4000       0.50      0.50      0.50         4

    accuracy                           0.88        40
   macro avg       0.88      0.88      0.87        40
weighted avg       0.88      0.88      0.87        40
```

NN分类器

其次是NN方法（这是采用神经网络进行分类的方法），其通过以下代码创建：

```
model = SimpleClassifier(input_size, hidden_size, num_classes)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

在这里计算正确率的方法是用判断正确的数目除以总的测试集合:

```
accuracy = total_correct / total_samples
```

最后训练了两千轮，学习率是0.01，隐藏层大小是输入层大小的三倍时，loss = 0.0012，而正确率 = 92.5%（但是每次的正确率会有所不同，这说明有一些信号比较模糊，难以100%判断其松紧度）

而将训练轮数改为3000轮（大于3000轮的意义不大了）的时候，loss = 0.0008，正确率达到了97.5%.

混合特征

其代码在classifier_mix.py中，主要代码如下所示：

```
while(m <= 3600):
    m += 400
    # 直接把分段之后的声音信号输入其中
    data=np.load("data/mfcc_{}.npy".format(m))
    data1=np.load("data/fft_{}.npy".format(m))
    for i in range(len(data) - 4):
        tmp = []
        for k in range(len(data[0])):
            tmp.append(data[i][k][0])
        for l in range(len(data1[0])):
            tmp.append(data1[i][l][0])
        X_train.append(tmp)
        y_train.append(m)
    for j in range(len(data) - 4,len(data)):
        tmp = []
        for k in range(len(data[0])):
            tmp.append(data[j][k][0])
        for l in range(len(data1[0])):
            tmp.append(data1[j][l][0])
        X_test.append(tmp)
    y_test.append(m)
```

这里的分类器我首先选择了KNN，信号选择了fft和mfcc的结合，结果如下所示,正确率为0.8：

```
Accuracy: 0.80
Classification Report:
              precision    recall  f1-score   support

         400       1.00      1.00      1.00         4
         800       1.00      1.00      1.00         4
        1200       1.00      1.00      1.00         4
        1600       1.00      1.00      1.00         4
        2000       0.75      0.75      0.75         4
        2400       0.50      0.75      0.60         4
        2800       0.20      0.25      0.22         4
        3200       1.00      0.50      0.67         4
        3600       1.00      1.00      1.00         4
        4000       1.00      0.75      0.86         4

    accuracy                           0.80        40
   macro avg       0.84      0.80      0.81        40
weighted avg       0.84      0.80      0.81        40
```

可得和原先的KNN相比，其相同的松紧度的判断正确率不一样。而如果我换用SVM分类器，特征选择sound和mfcc，其正确率达到了97%！有了较好的进步，几乎接近神经网络的方法，结果如下所示：

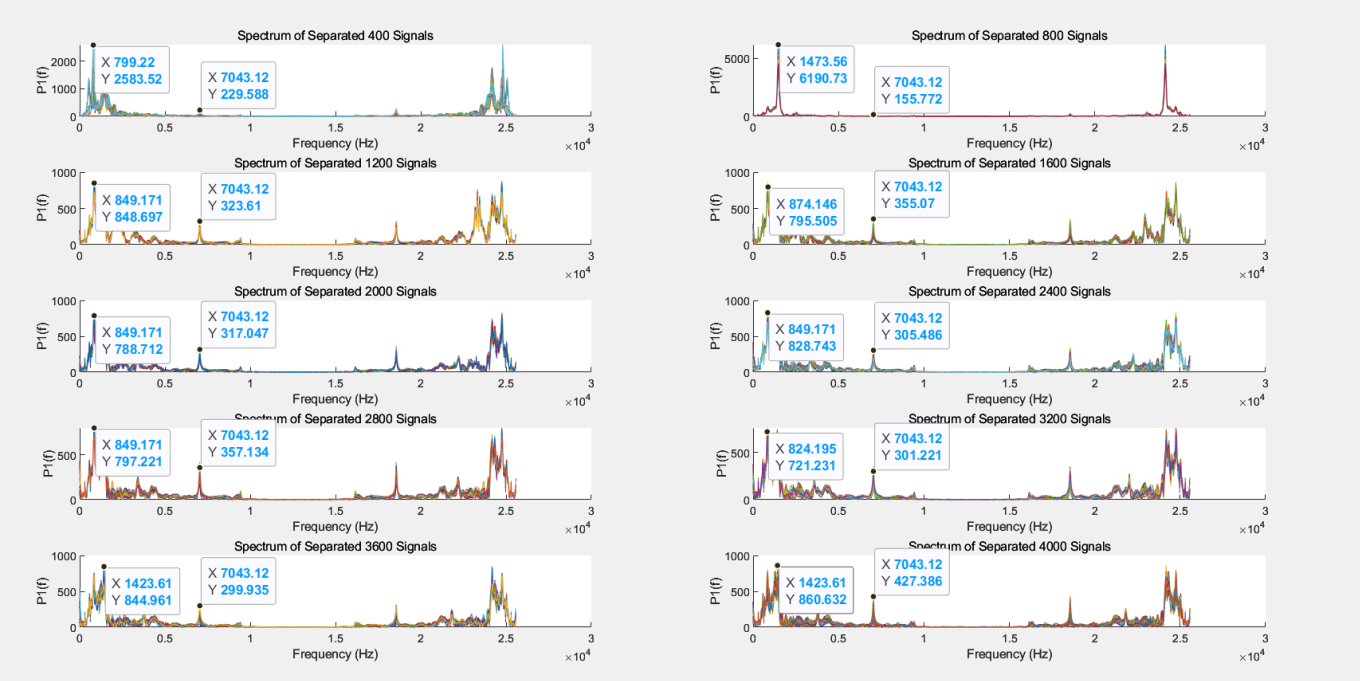
```
Classification Report:
              precision    recall  f1-score   support

         400       1.00      0.75      0.86         4
         800       1.00      1.00      1.00         4
        1200       1.00      1.00      1.00         4
        1600       1.00      1.00      1.00         4
        2000       0.80      1.00      0.89         4
        2400       1.00      1.00      1.00         4
        2800       1.00      1.00      1.00         4
        3200       1.00      1.00      1.00         4
        3600       1.00      1.00      1.00         4
        4000       1.00      1.00      1.00         4

    accuracy                           0.97        40
   macro avg       0.98      0.97      0.97        40
weighted avg       0.98      0.97      0.97        40
```

因此我们可以知道这个创新的方法有其优势所在。

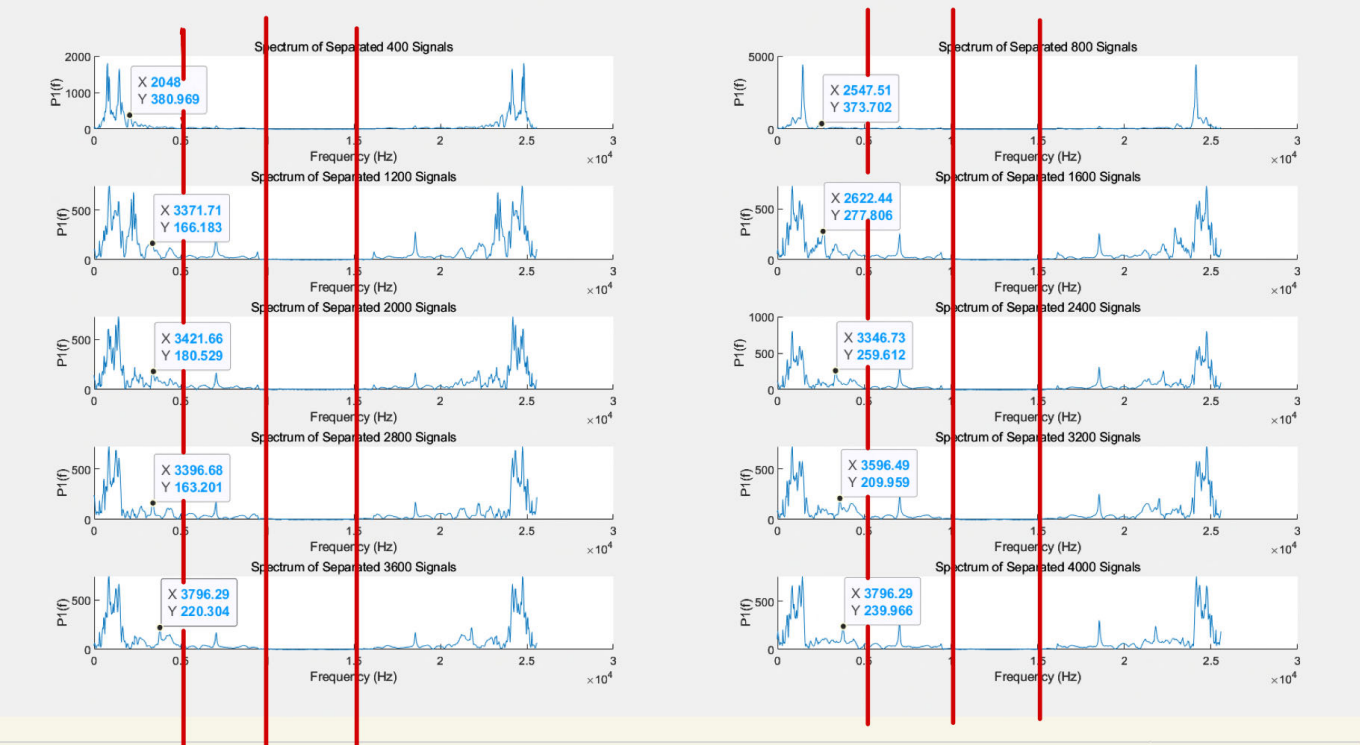
六. 获得松紧度（压力值）和频率的变化关系

由于机器学习或者神经网络的方法并没有一个物理意义上的好的特征来区分不同松紧度，因此我结合传统方法，来得到松紧度和频率的变化关系。其拟合曲线的代码在tightness_feature.py中，并且这个也可以作为**五.提取特征并分类**的补充。代码是frequency.m（这个代码显示了频谱图）拟合直线的代码在8.py中。

在这里我将采样点数N设置为1024。首先我尝试了对同一松紧度的所有频谱图显示在一张图上，来得到其特殊的频率，结果如下：



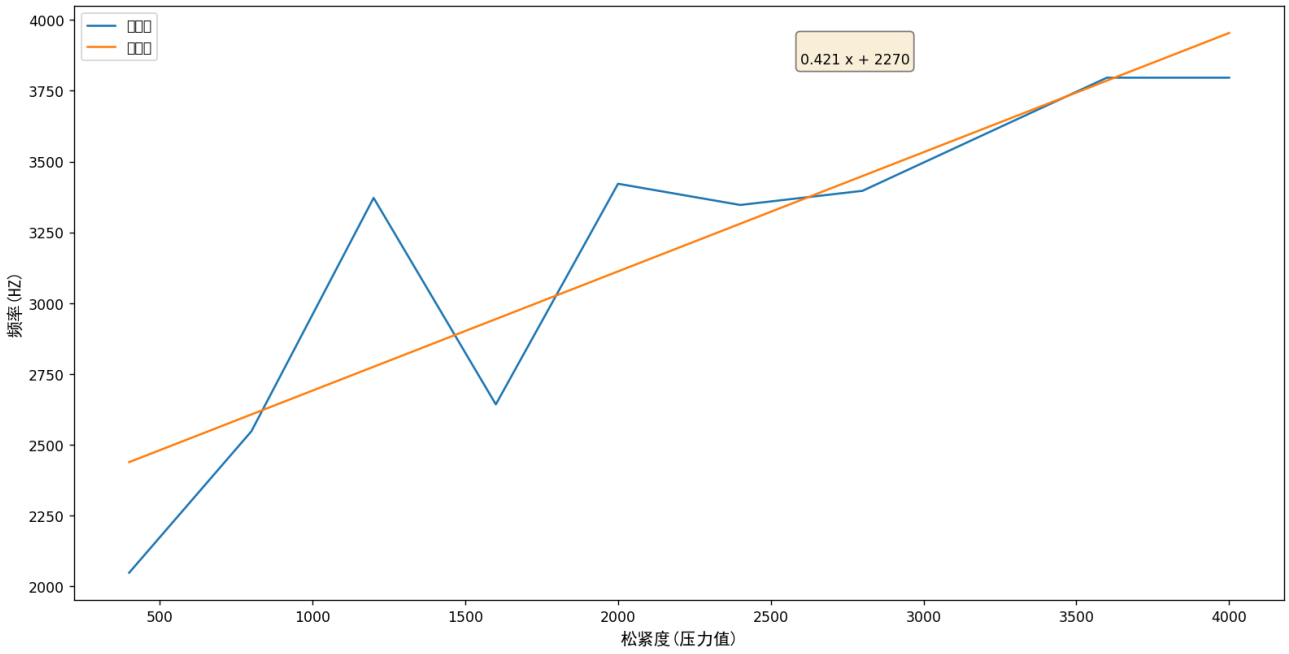
发现其效果并不好，这可能是由于所有相同松紧度的信号的频谱图，总会有一些不具备代表性，因此我选择每个松紧度的一个具有代表性的信号，其显示结果如下：



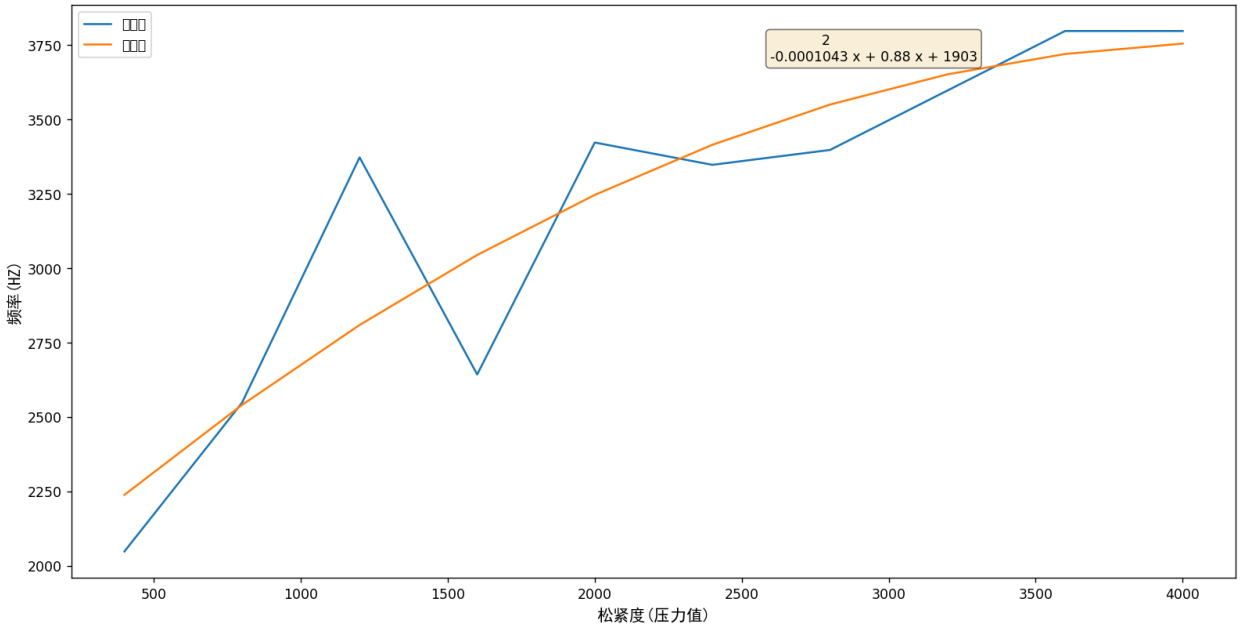
我们根据红线将其分为三部分，分别是f范围为0-500，500-1000，1000-1500，在这里我们仅研究第一个部分。其频率得到的依据是极值但是和最大极值差距大于某个阈值的频率位置。我们将这个频率视为和松紧度有关的频率，提取成表格如下所示：

松紧度（压力值）	频率（HZ）	松紧度（压力值）	频率（HZ）
400	2048	2400	3346.73
800	2547.51	2800	3396.68
1200	3371.71	3200	3596.49
1600	2622.44	3600	3796.29
2000	3421.66	4000	3796.29

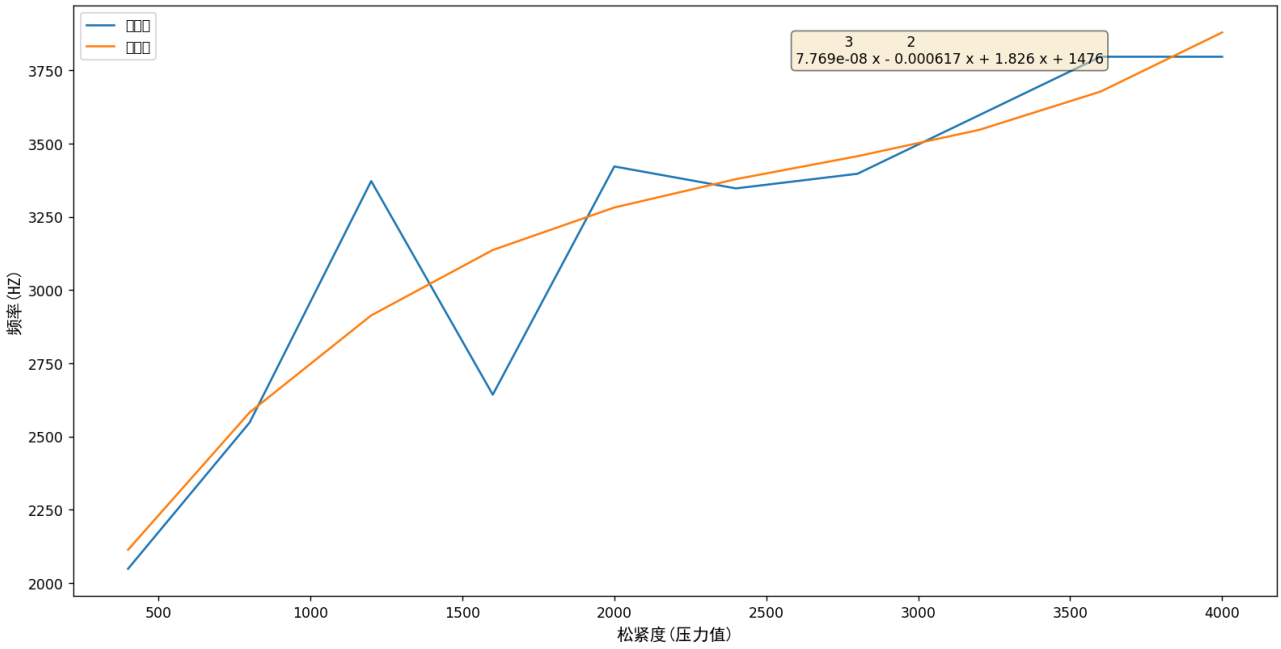
而拟合直线的代码在8.py中，最后得到一次拟合结果如下所示：



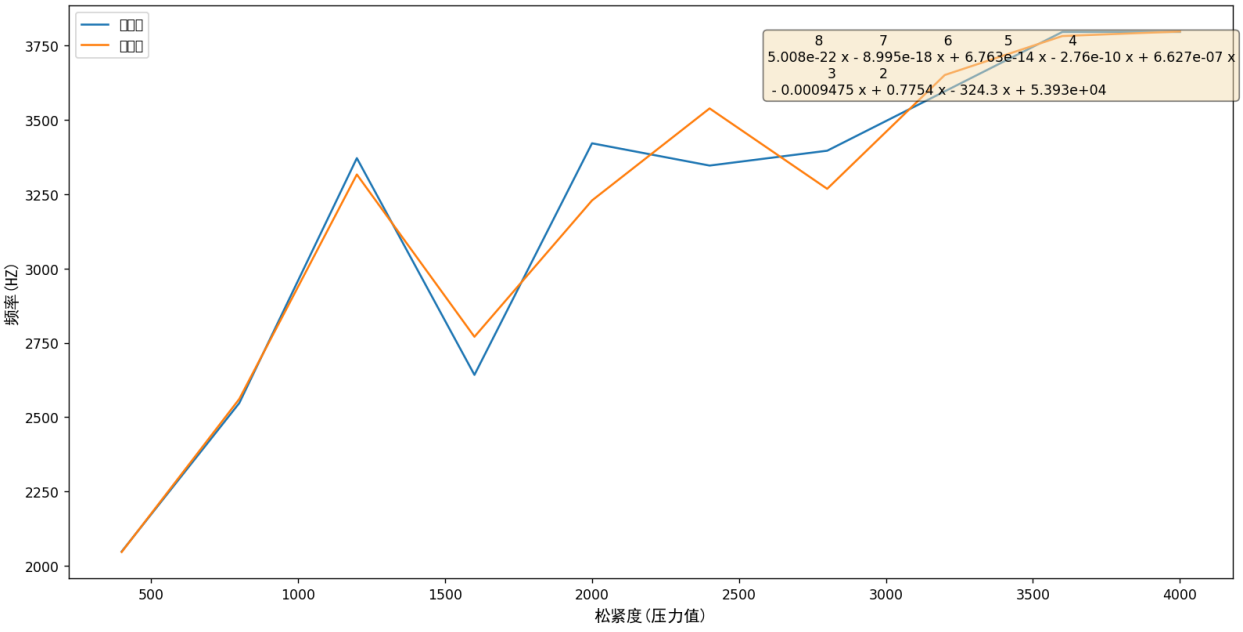
二次拟合结果如下所示：



三次拟合结果如下所示：



高次拟合结果如下所示：



其中黄色框显示的是拟合曲线的方程，最上面一排表面这是x的次数。从图中我们可以得到一次和三次的拟合结果较好，其更能表现总体趋势和变化，而高次会发生过拟合情况，反而曲线情况不好。

七. 总结

通过本次实验我了解了如何区分不同松紧度的敲击信号，并且对机器学习的分类器有了更深刻的认识。一般具有较好物理意义的信号特征，比如熵、总能量、幅度极大值处的频率等等，这些在敲击信号的分类中效果都不是很好，虽然机器学习进行分类可能不能得到一个较好的特征，但是分类效果很不错。

除此之外我还了解了怎么处理一些给定的信号，真正体会到了数字信号处理的过程和意义！