设计报告

辜俊皓 2017011281，雷城乐阳 2017011155，谢家祺 2017011140

1. 基于给定图像特征的 LR 分类

1.1 预调研（详见 torchLR.py）

首先根据课上内容，我们将特征向量 x 与全一向量进行拼接，从而将 b 与 w 合并，我们有：$y = \frac{1}{1+e^{-w^T x}}$

同时直接采用课件中提到的损失函数：$\text{Loss} = -\log l(w) = -\Sigma_{i=1}^{n}\left(y_i * w^T x_i - \log\left(1 + e^{w^T x_i}\right)\right)$

同时得到：$\frac{\partial \text{Loss}}{\partial w_j} = -\Sigma_{i=1}^{n} y_i * x_{ij} - \frac{x_{ij} e^{w^T x_i}}{1+e^{w^T x_i}}$，$\frac{\partial^2 \text{Loss}}{\partial w_j^2} = \Sigma_{i=1}^{n} \frac{x_{ij}^2 e^{w^T x_i}}{\left(1+e^{w^T x_i}\right)^2}$

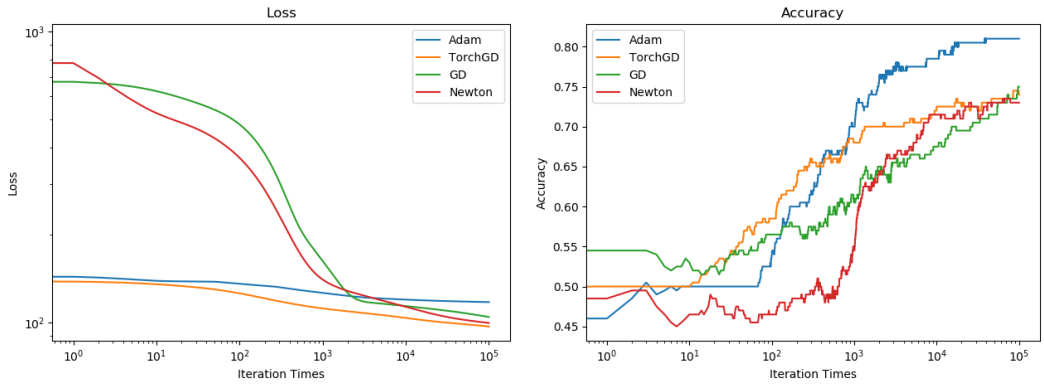
其中$\frac{\partial \text{Loss}}{\partial w_j}$表示对向量 w 的第 j 个分量求偏导，$x_{ij}$表示第 i 个 x 特征向量的第 j 个分量；

首先我们采用了 pytorch 库（版本 1.3.1）对几种方法进行了比较测试，主要包括①pytorch 自带的 Adam 算法（Adam）；②利用 pytorch 中 Variable 自带的数值梯度计算实现的梯度下降(TorchGD)；③上述推导得到的理论梯度下降(GD)；④上述推导得到的理论牛顿迭代法(Newton)；
我们将以上方法均在 torchLR.py 中予以实现，并测试了各自模型的准确率（训练集上）；最终得到的损失函数值以及准确率表格如下：

Methods	Learning Rate	Times	Loss	Accuracy
train_torch (Adam)	1.00E-02	1.00E+05	117.4044	81.0%
train_torchGD	1.00E-03	1.00E+05	96.7714	74.0%
train_GD	1.00E-04	1.00E+05	102.9180	74.5%
train_Newton	1.00E-02	1.00E+05	99.9791	73.0%

注：lr 的选取均采用测试中结果的近似最优值

随迭代次数的变化曲线如下，其中左图横纵坐标均为对数值，右图仅横坐标为对数值：



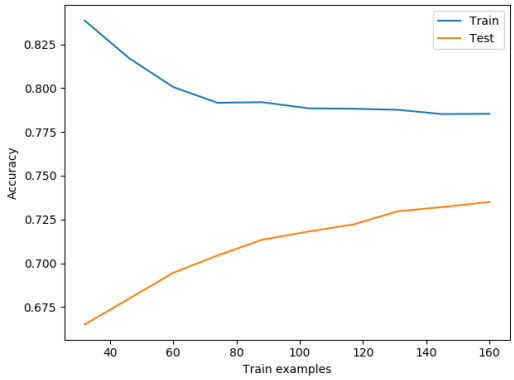
损失函数值　　准确率

从最终结果上来看，可以发现后三种方法得到的最终准确率和损失函数结果均相近，均在$\text{Loss} = 100$，$\text{Accuracy} = 74\%$左右，而 Adam 得到的结果尽管损失函数值较大，但准确率超过了 80%；而从收敛速度上来看，由于后两种方法中我们设定的初始条件为所有参数服从$[-5,5]$上的均匀分布，可以看到最开始后两种方法的损失函数明显较高，但随着迭代次数的增加($> 10^4$)，四种方法的损失函数值均相近，而从准确率上来说，四种方法最初基本都在 50%附近，虽然 GD 和 TorchGD 方法最初增长较快，但在10^3次迭代后，均被 Adam 方法超过，而尽管 Newton 方法最初收敛较慢，但在10^4次迭代过后，后三种方法均收敛到了近似的水平；
有了上述部分研究的结果，为了便于交叉验证以及助教测试，我们将代码移植到了 sklearn 平台上；

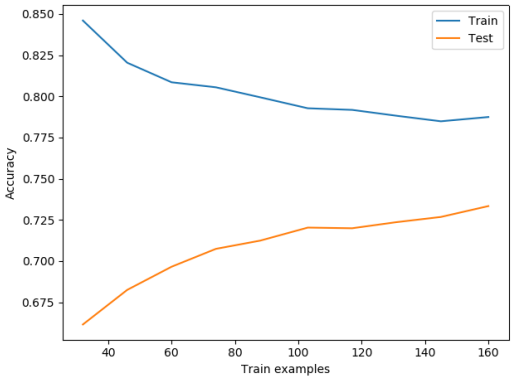
1.2 具体 sklearn 实现（详见 sklearnLR.py）

在 sklearn.linear_model 中直接包含了 LogisticRegression 类，我们可以直接通过设定其参数如学习率，迭代结束误差，优化器选择等，完成简单的 LR 回归
但在之前的预调研中，我们可以看到 LR 即使在训练集上也不能达到较好的结果，这主要是由于 LR 模型自身

描述能力的限制，以下我们分别选择 liblinear 即坐标轴下降（梯度下降）和 newton-cg 拟牛顿法作为优化器，利用 learning_curve 函数测试，在 10-fold 交叉验证下得到的训练集与验证集准确率如下图（进行了 100 次重复测试）：



梯度下降



牛顿迭代

从中可以明显看出线性 LR 模型的描述能力有限，在训练集变大时，准确率会随之降低，因为线性模型已经不再能确保将所有样本成功分离；而反观测试集，可以看到随着训练集的增大，准确率在随之上升，说明尽管 LR 描述有限，但仍然能提取样本整体的共有特征；

总的来说，我们基本可以认为简单 LR 模型在该特征向量下，测试集的准确率应该不会高于 75%，而这在很大程度上时线性模型本身的描述能力所限制的，最终我们小组的测试结果详见"Ans/A.npy"，而代码直接生成的数据文件以长度为 100 的 ndarray 的形式存储在"TestAns/LR.npy"文件中；

2. 基于音频的特征提取和 SVM 分类

2.1 语音信号的特征提取（详见 svm_source.py）

在本次作业中，我们采用语音的梅尔倒谱系数（MFCC）作为音频的特征。该提取过程主要包括几步：首先对连续的语音信号进行预加重，即将语音通过高通滤波器，来补偿语音信号中较弱的高频部分；然后对信号进行分帧，并对每一帧乘以一个汉明窗来增加帧左右端的连续性；第三步对加窗后的信号做快速傅里叶变换，然后将能量谱通过三角形滤波器以消除谐波；再将能量取对数并进行 DCT 变换得到 MFCC 系数，最后为使特征更好的体现时域连续性，对系数进行差分。直接调用 python 中 librosa 库中的 mfcc 函数提取 MFCC 特征的参数，具体调用格式如下：

```
mf = librosa.feature.mfcc(y=signal, sr=sample_rate, n_mfcc=15)
```

因为提取的 mfcc 参数是分帧后获得的，有一千多帧，总共就是上万维特征，考虑到这个数据量比较大，我们决定选择一些典型的特征向量，这是通过 k-means 聚类算法来实现的。k-mean 聚类简单来说就是先随机选取特征点，按选取的特征点对样本分类（样本离哪个点近就归于哪类），再在每类中选取质心重新作为该类的特征向量，之后对新的特征向量再分类，上述过程不断迭代，最终获得比较理想的分类结果。由 k-means 算法提取出来的 k 组 mfcc 系数作为该样本的音频特征。

2.2 SVM 的处理数据（详见 SVM.py）

提取完音频特征之后，就可以用支持向量机（SVM）对提取的特征进行处理了，SVM 的基本思路就和课程中提到的那样，用一个超平面分割空间，使得两类点离超平面的距离的最小值最大，这是一个凸优化问题，而且还可以通过对 SVM 对偶问题的求解简化思路，并且只需要计算两两向量的内积再优化就行了。实际处理时，对直接的 SVM 进行了一些改进，首先是为了解决线性不可分问题，把向量映射到高维空间，此时向量的内积可以用核函数来替代，常见的核函数有线性核、多项式核、sigmoid 核、高斯核（rbf）等；另一个改进是引入松弛变量，即允许样本点超过超平面，但这时会加上一个惩罚因子 c，加入松弛因子可以减小那些离群样本对 SVM 的影响，增加模型的泛化能力。

由于之前对音频提取特征的时候每一帧都提取了多个特征向量，此时处理有两种方式，一种是把提取出来的所有的特征向量直接拼成一行向量送进 SVM 处理，另一种方式是将每一个特征向量根据其所属音频分别打上标记，把打上标记的向量分别送进 SVM 进行处理，由于担心前一种方式中向量的拼接顺序可能会影响判决结果，

因此我选择了第二种方法，判决时则是同样将测试语音提取多个特征向量，每个向量分别分类，哪一类的特征向量多就判断该音频属于哪类。

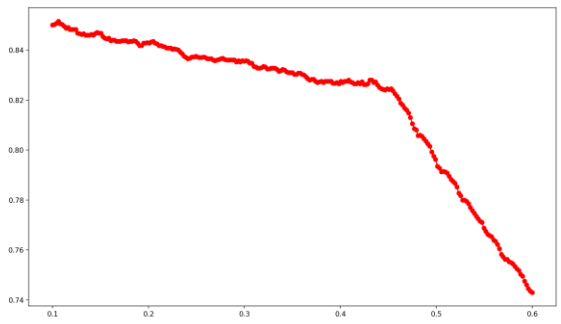
代码实现：直接使用了 sklearn 中的 svm 工具进行处理，具体实现代码如下：

```
rbf_kernel_svm_clf = Pipeline((
    ("scaler", StandardScaler()),
    ("svm_clf", SVC(kernel="rbf", gamma=0.241, C=1))
))
```

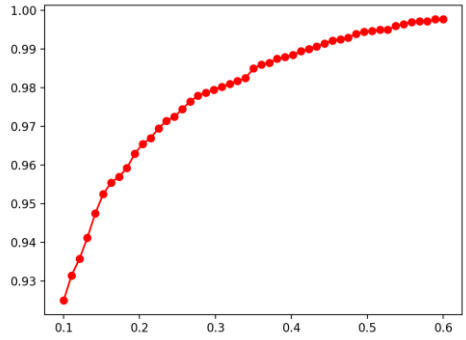
其中 StandardScaler 函数是对输入数据进行标准化，之后再使用 svc 进行学习，kernel 参数是核函数，gamma 是核函数的参数，和过拟合有关，c 是惩罚因子。

经过数据验证发现高斯核（rbf）的效果最好，其他核函数的测试结果就不在报告中赘述了。

由于 rbf 函数相当于把数据延申到无穷维，因此通过参数选取可以把测试集拟合到 100%准确率，但这时就很容易发生“过拟合”的情况，为了避免这种情况，采用交叉验证的方法，即把数据划分出验证集和训练集，观察模型在验证集上的正确率。采用 10 折交叉验证，下面是正确率随参数 gamma 的变化曲线图：



验证集准确率

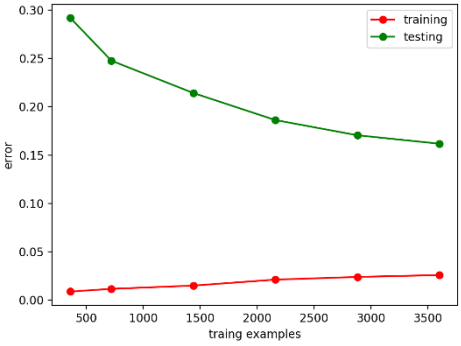


训练集准确率

可以看到验证集上准确率随 gamma 升高而减小，测试机上准确率随 gamma 增大而增大，综合考虑两者，我们最终考虑选用的 gamma=0.25

还可以画出学习曲线如右图：

从图像中我们可以看出，随着训练集样本的增大，验证集上的错误率越来越低，训练集上的错误率略有增加，由此验证增大样本数量可以有效改善模型的性能，因此可以通过增大样本数解决过拟合问题



2.3 不足与改进的想法：

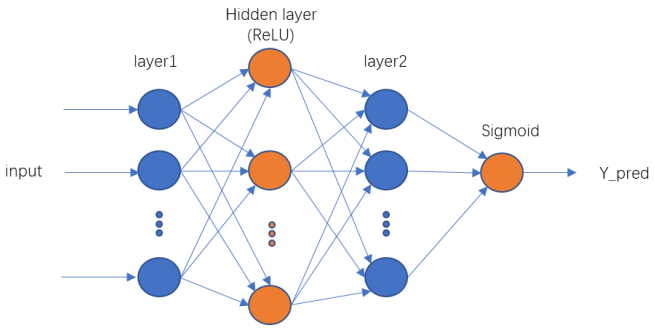
从之前的图像中可以看出在训练集上的准确率还不够高，不过由于是通过对一个语音样本的多个特征向量进行归类，根据不同类向量的多少进行判别，因此最终的结果会更好一些。不过时间所迫没有在这上面画交叉验证的图像了。

对于音频特征的提取，我还了解到一些方法，比如提取 MFCC 系数后直接用 PCA 进行降维，或者将 k-means 聚类改成 GMM 聚类，也没来得及对比效果。

3. 结合图像和音频的自选方法分类

3.1 多层神经网络的识别方法（详见 neuralNetwork.py）

第三问中我们尝试使用多层神经网络来进行识别，我们以第二问提取出的 MFCC 作为特征，以 pytorch 架构进行了尝试。我们尝试构建了两个线性层，中间的隐藏层采用 ReLU，为了避免参数过多，我们没有采用全连接网络，对于一个 MFCC 特征，输入层节点数为 108，输出层节点数为 1，我们依照 $h = \sqrt{m+n} + a$的方式设定隐藏层节点数（其中 h 为隐藏层节点数，m 为输入层节点数，n 为输出层节点数，a 为 0-10 中的一个随机数）。



我们尝试了不同的损失函数和优化器的选取，其中效果最好的为 BCELoss 和 Adam 优化器的组合，在训练集上能够较快的达到 0.98 的正确率，但在交叉验证中，这种方法的表现与第一问差不多，推测可能是由于训练样本或过少而导致了过拟合；具体采用的网络模型如下图，中间层采用的非线性层为 ReLU，最后一层采用 Sigmoid 输出，采用 0.5 的 Dropout：

3.2 LR 回归改进尝试（详见 GBDT_LR.py）

但实际上，我们可以对原有的特征向量引入一些非线性变换，使 LR 模型能更好地拟合这一数据，因此我们尝试了利用 PolynomialFeatures 引入二级交叉项，提高特征向量维度，最终结果尽管在训练集上可以达到更高的准确率，但对验证集上效果仍然不是很好；

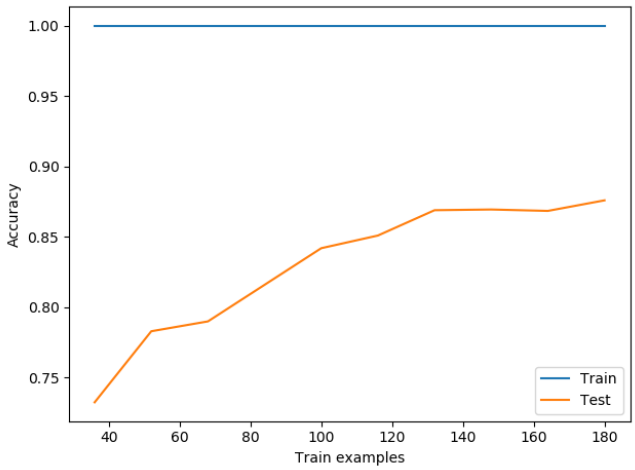
除此之外，还尝试了一种 GBDT+LR 的实现方式，先采用 sklearn 自带的 GradientBoostingClassifier 对数据建立一棵最优分类树，随后利用训练得到的分类依据对原有的 200 个训练集进行多层分类，最终将分类得到的叶子节点对应的数字进行 one-hot 编码得到一个 01 稀疏矩阵作为新的特征向量 x'，然后再利用 LR 模型进行二分类；最终在测试集上已经可以稳定在 95%以上的准确率，说明模型的描述能力是足够的，可在验证集上的结果仍然位于 75%左右，具体结果同样以长度为 100 的 ndarray 的形式存储在"TestAns/GBDT_LR.npy"文件中，同时 GBDT 自身分类结果也以长度为 100 的 ndarray 的形式存储在"TestAns/GBDT.npy"文件中，仅作为参考结果；最终推测分类结果不理想的原因可能是特征向量即"feat.npy"本身缺乏代表性，故后续的改进中不再使用"feat.npy"文件；

3.3 基于 MFCC 特征的 GBDT 分类（详见 GBDT_mfcc.py，C 部分最终方案）

基本操作与上述 3.2 部分相同，主要差别在于考虑到第 2 部分中 MFCC 较好的效果，我们将特征向量由图片信息换为了前述聚类 MFCC 特征；但相比于第 2 部分的做法，这一部分采用的 MFCC 特征维数更低，所以相对来说运行效率更高，以下进一步分析其运算复杂度；

首先，每次维护一棵决策树需要计算各维特征的划分点对应的基尼系数，故如果最终建成一棵完全二叉树，即深度d ≈ log(n)，每棵树所需的时间复杂度为O(dim × nlogn)，其中 dim 为每个输入特征向量的维数，n 为样本个数；而整个算法中每次梯度提升（Gradient Boosting, GB）都是利用建树过程的凸性进行优化残差，反复迭代总共生成 numTrees 棵决策树，故最终的时间复杂度为O(numTrees × dim × nlogn)；其中如果考虑在建树时限制最大树高 max_depth，子采样 subsample 都会对最终的复杂度产生影响，但最终的复杂度数量级基本不变；从中我们可以看出减小输入的聚类 MFCC 特征的维度能够有效地降低运算复杂度，除此之外，通过提高学习率来降低迭代次数也能在牺牲一定性能的条件下提高效率；

而关于最后的运算结果，在测试集上进行 10-fold 交叉验证，最终在训练集上稳定为 100%，在验证集的准确率约为 90%，略低于第 2 部分结果，但实际运行复杂度相比之下更低；测试结果详见"Ans/C.npy"，而代码直接生成的数据文件以长度为 100 的 ndarray 的形式存储在"TestAns/GBDT_MFCC.npy"文件中，除此之外同样也有与 LR 方法进一步结合的结果，但最终效果不如直接采用 GBDT 分类；



附录：

①小组 github 仓库：https://github.com/STD-Group/Debate-Estimate

②本次实验的内容相关性比较强，难以拆分，基本是由小组成员共同完成的，而且主要时间花在降低误差中，这一部分也是所有组员经过讨论分别实现各种优化方案的；