第一次大作业：语音端点检测

521030910387 薛翔元

1. 基于线性分类器和语音短时能量的简单语音端点检测算法

1.1. 任务描述

基于线性分类器和语音短时信号特征，实现简单的语音端点检测算法，要求如下：

- 提取语音的短时信号特征（如短时能量、过零率、短时频谱、基频等）
- 使用简单的线性分类器进行分类（如阈值分类器等）
- 使用简单的状态机完成预测

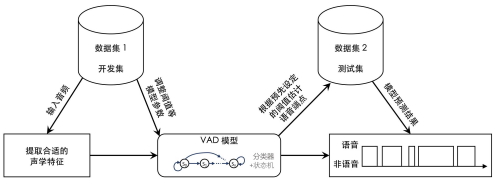

图 1: 任务介绍

1.2. 数据预处理

为将音频.wav 文件转换为可供计算的特征，我们首先将音频读入为 numpy 数组。以 vad/wavs/dev/107-22885-0023.wav 为例，得到如下长度为 243440，采样率为 16kHz 的波形。

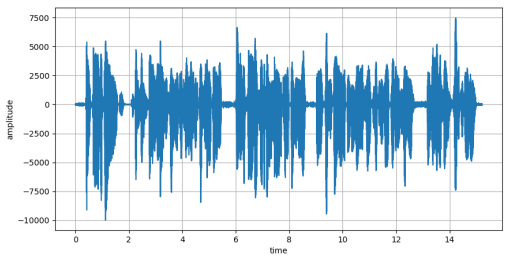

图 2: 音频波形

根据测试标准按照 25ms 帧长，10ms 帧位移对波形进行分帧，对于波形尾部不足 25ms 的片段，通过 pad 函数进行补零，窗函数可以选择矩形窗和 Hamming 窗，分别用于不同特征的提取。

算法 1 波形分帧

输入 波形 $w^{1\times p}$, 帧长 n, 帧位移 m, 窗类型 s
输出 分帧结果 $F^{q\times n}$

依据 s 构造窗函数 $f^{1\times n}$ 置于起始点
while f 与 w 存在重叠 **do**
  if f 末端未超出 w **then**
    直接截取 w 作为一帧
  else
    截取至 w 末端并补零作为一帧
  将该帧加入分帧结果 F
  将 f 向右移动 m 位
return F

根据上述算法所得分帧结果 F 为 $q \times n$ 的矩阵，其中 q 为波形产生的总帧数，n 为帧长，可以通过以下公式计算矩阵的形状：

$$q = \left\lceil \frac{p-n}{m} \right\rceil + 1 \tag{1}$$

算法中，矩形窗定义如下：

$$R(k) = \begin{cases} 1, & 0 \le k \le n-1 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Hamming 窗定义如下：

$$H(k) = \left[0.54 - 0.46\cos\left(\frac{2\pi k}{n-1}\right)\right] R(k) \tag{3}$$

以上述音频为例，得到的分帧结果为 1522×400 的矩阵，其中 400 的长度对应于 16kHz 采样率下 25ms 的帧长。

1.3. 特征提取

1.3.1. 短时能量

短时能量是某一帧内波形数值的平方和，反映了音频的信号强度，在一定程度上可以揭示语

音状态，其计算公式如下：

$$E(f) = \sum_{k=1}^{n} x_f^2(k) \tag{4}$$

其中 f 代表一个帧，$x_f(k)$ 代表该帧中第 k 个采样点的数值，n 为该帧的长度。

短时能量在 `numpy` 中可以直接向量化计算，所得结果应根据最大值进行归一化，以便作为特征输入分类器。

1.3.2. 过零率

过零率是某一帧内波形经过零点的频率，反映了音频的变化速率，在一定程度上可以揭示语音状态，其计算公式如下：

$$Z(f) = \frac{1}{2n} \sum_{k=1}^{n-1} |\text{sign}(x_f(k)) - \text{sign}(x_f(k+1))| \tag{5}$$

其中 f 代表一个帧，$x_f(k)$ 代表该帧中第 k 个采样点的数值，n 为该帧的长度，$\text{sign}(x)$ 为符号函数，其定义如下：

$$\text{sign}(x) = \begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases} \tag{6}$$

过零率在 `numpy` 中可以直接向量化计算，所得结果在 $[0, 1]$ 范围内，不需要进行归一化，直接作为特征输入分类器。

1.3.3. 频谱质心

频谱质心是某一帧内频谱能量的加权平均值，反映了音频的频谱均值，在一定程度上可以揭示语音状态，其计算公式如下：

$$C(f) = \frac{\sum_{k=1}^{n} k X_f(\omega_k)}{\sum_{k=1}^{n} X_f(\omega_k)} \tag{7}$$

其中 f 代表一个帧，ω_k 代表第 k 个频谱点的频率值，$X_f(\omega_k)$ 代表该帧中第 k 个频谱点的幅度，n 为该帧的长度，同时也是离散傅里叶变换后的频谱点数。

短时信号的频谱可以通过离散傅里叶变换得到，`numpy` 中提供了快速傅里叶变换的工具，利用 `fft.rfft` 可以得到变换后的复数结果，通过取模的方式将其转换为实数幅度，而各个频谱点对应的频率值可以通过 `fft.rfftfreq` 得到，其计算公式如下：

$$\omega_k = \frac{k}{2n} \cdot f_s \tag{8}$$

其中 k 为频谱点数，n 为帧长，f_s 为音频的采样率。根据人类的听觉特性，频谱质心的范围一般在 20Hz 到 20kHz，因此不需要进行归一化，直接作为特征输入分类器。

1.3.4. 基频

基频是某一帧内语音的基本频率，反映了声源的发声特性，在一定程度上可以揭示语音状态。基频的计算有多种方法，本文采用的是基于自相关函数的方法，短时信号 $x_f(k)$ 的自相关函数定义为：

$$R_x(k) = \sum_{m=1}^{n-k} x_f(m) x_f(m+k) \tag{9}$$

假如 $x_f(k)$ 为周期信号，其自相关函数 $R_x(k)$ 在某一合理范围内的最大值点即对应于该信号的基频。因此，基频的计算可以通过计算自相关函数的极大值点来实现。

在 `numpy` 中，可以通过 `correlate` 函数计算自相关函数，然后，我们取其中频率在 20Hz 到 2kHz 的部分，通过 `argmax` 函数找到最大值点，该点即对应于基频。基频的范围一般在 20Hz 到 2kHz，因此不需要进行归一化，直接作为特征输入分类器。

1.3.5. 特征向量

现在，我们已经确定了上述四种特征的计算方法，为了综合利用这些特征进行分类，我们将其合成一个特征向量，作为某帧波形的特征信息，

特征向量定义如下：

$$\boldsymbol{v}(f) = (E(f), Z(f), C(f), P(f))^T \tag{10}$$

其中 f 代表一个帧，$E(f)$ 为 f 的短时能量，$Z(f)$ 为 f 的过零率，$C(f)$ 为 f 的频谱质心，$P(f)$ 为 f 的基频。

由于四种特征的数值差异较大，我们给每种特征赋予一个权重，使得特征向量的每个数值具有相同的数量级，权重定义如下：

$$\boldsymbol{w} = (10^{-3}, 1, 10^{-1}, 10^{-2})^T \tag{11}$$

于是归一化特征向量可以表示为 $\tilde{\boldsymbol{v}}(f) = \boldsymbol{w} \cdot \boldsymbol{v}(f)$。

算法 2 归一化特征向量生成

输入 帧向量 $f^{1\times n}$ 及其采样频率
输出 归一化特征向量 $\tilde{\boldsymbol{v}}(f)$
1: 计算 f 的短时能量 $E(f)$
2: 计算 f 的过零率 $Z(f)$
3: 计算 f 的频谱质心 $C(f)$
4: 计算 f 的基频 $P(f)$
5: $\boldsymbol{w} \leftarrow (10^{-3}, 1, 10^{-1}, 10^{-2})^T$
6: $\boldsymbol{v}(f) \leftarrow (E(f), Z(f), C(f), P(f))^T$
7: **return** $\boldsymbol{w} \cdot \boldsymbol{v}(f)$

根据上述算法，输入 $1 \times n$ 的帧向量可以得到 1×4 的归一化特征向量，该向量确定了该帧的音频特性，对每帧的特征向量纵向堆叠，可以得到输入音频在时间序列上的特征矩阵。

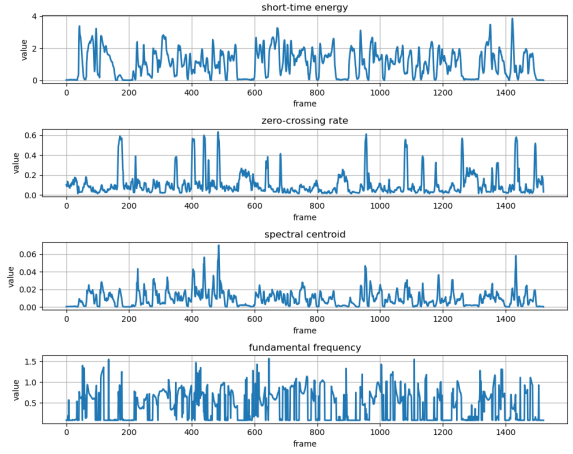

图 3: 音频特征

采用与上一节相同的样例音频，根据上述算法绘制音频特征如上图所示。可以看到，各个特征与原始音频具有较强的相似性，可以较好地反映语音状态，我们将在下一节中利用这些特征进行分类。

1.4. 算法描述

1.4.1. 逻辑回归

逻辑回归是一种广义的线性分类器，其基本思想是将输入的特征向量映射到一个概率空间，然后通过概率值判断输入的类别。逻辑回归的模型定义如下：

$$\hat{y} = \sigma\left(\boldsymbol{w}^T \boldsymbol{x} + b\right) \tag{12}$$

其中 $\boldsymbol{w}$ 为权重向量，$\boldsymbol{x}$ 为特征向量，b 为偏置，$\hat{y}$ 为预测结果，σ 为 sigmoid 激活函数，定义如下：

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{13}$$

逻辑回归的求解方法很多，最常见的是设计损失函数，利用梯度下降算法求解优化问题。逻辑回归的损失函数定义如下：

$$\mathcal{L}(\boldsymbol{w}, b) = \frac{1}{m} \sum_{i=1}^{m} \phi(y_i, \hat{y}_i) \tag{14}$$

其中 m 为样本数，y_i 为第 i 个样本的真实值，$\hat{y}_i$ 为第 i 个样本的预测值，ϕ 为二分类交叉熵损失函数，定义如下：

$$\phi(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \tag{15}$$

于是逻辑回归转化为如下优化问题：

$$\min_{\boldsymbol{w}, b} \mathcal{L}(\boldsymbol{w}, b) \tag{16}$$

根据梯度下降算法，其更新公式为：

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \cdot \frac{\partial \mathcal{L}(\boldsymbol{w}, b)}{\partial \boldsymbol{w}} \tag{17}$$

$$b \leftarrow b - \alpha \cdot \frac{\partial \mathcal{L}(\boldsymbol{w}, b)}{\partial b} \tag{18}$$

其中 α 为梯度下降的步长，又称为学习率。

我们首先读取训练集中的所有音频文件，分帧并提取特征向量。然后，将所有特征向量拼接为特征矩阵，将所有标签拼接为标签矩阵，作为模型输入，调用 `sklearn` 包中 `LogisticRegression` 类训练逻辑回归模型，得到如下训练结果：

```
[progress] training dataset has been created
feature: (5088315, 4), label: (5088315, )
[progress] classifier training has been finished
weight: [2.098, -3.066, 590.357, 1.396]
```

训练完成后，我们可以类似地将模型应用到开发集和测试集中的音频文件，产生每帧的预测结果。最后，我们将每帧的预测结果合并，得到音频文件的预测结果。

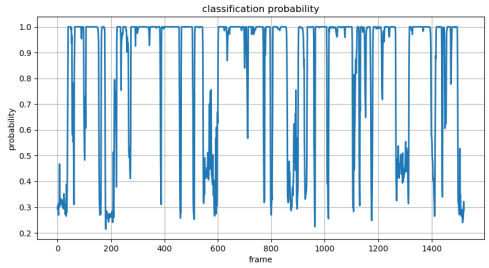

图 4: 原始预测结果

样例结果显示，逻辑回归模型以概率形式产生了预测结果，且与原始音频符合较好，但由于噪声的存在，预测曲线比较粗糙。

1.4.2. 平滑滤波

在上一节中，我们使用逻辑回归模型对每帧语音的特征向量进行分类，从而得到每帧的预测结果。然而，这种预测假定各帧之间是相互独立的，忽略了语音信号在时域上的连续性。为了得到更可靠的预测结果，我们需要每个音频的预测结果进行平滑滤波，可以写为如下形式：

$$\hat{y}'_n = \frac{1}{p} \sum_{k=n-p+1}^{n} \hat{y}_k \tag{19}$$

其中 p 为滤波窗口的大小，$\hat{y}'_n$ 为第 n 帧的平滑预测结果，$\hat{y}_n$ 为第 n 帧的原始预测结果。

在 `numpy` 中，我们可以使用 `convolve` 函数将原始预测结果与归一化矩形窗进行卷积运算，从而实现平滑滤波。该过程中需要注意边缘的处理，保证平滑预测结果与原始预测结果具有相同的长度。

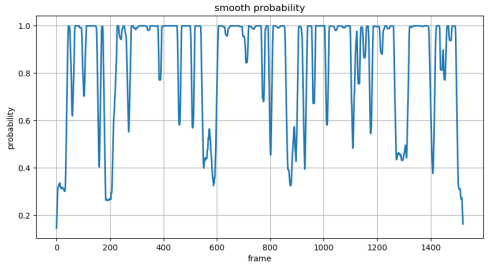

图 5: 平滑预测结果

可以看到，相比原始预测结果，平滑滤波后的预测曲线更为光滑，预测结果更为可靠。

1.4.3. 简单状态机

为最终确定语音端点，我们需要将概率预测转化为二分类标签。我们将阈值设定为 0.5，考虑到语音状态不可能频繁切换，我们使用简单的状态机优化预测结果，其状态转移如下所述：

- 从静音状态转移到语音状态，当且仅当包括当前帧在内的连续 k 帧预测概率 $p \geq 0.5$
- 从语音状态转移到静音状态，当且仅当包括当前帧在内的连续 k 帧预测概率 $p < 0.5$

其中 k 为转移长度，k 越大，预测曲线也就越平滑，但语音端点检测的延迟也会随之提高。

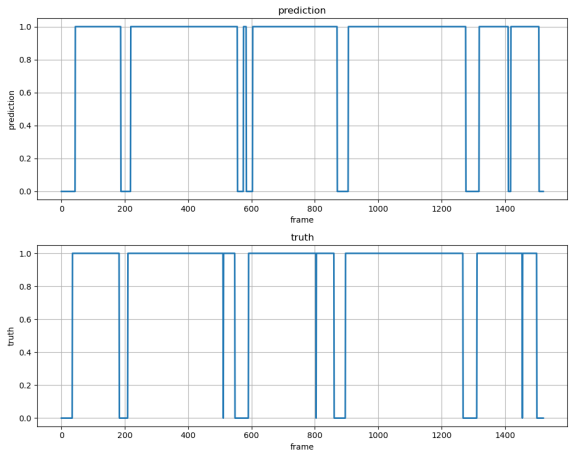

图 6: 最终预测结果

通过简单状态机，我们生成了最终的检测结果。可以看到，检测结果与真实结果基本一致，且端点检测的延迟较小，表明该方法产生了较好的检测效果。我们将算法流程总结如下：

算法 3 基于线性分类器的语音端点检测算法

输入 训练音频文件 $\boldsymbol{X}$ 及标签 $\boldsymbol{Y}$，音频文件 Q
输出 预测结果 P

1: 提取 $\boldsymbol{X}$ 的特征矩阵 $\boldsymbol{F}$
2: 建立逻辑回归模型 M，对 $\boldsymbol{F}$ 和 $\boldsymbol{Y}$ 进行拟合
3: 利用 M 对 Q 进行预测，得到原始预测结果 $\hat{P}$
4: 对 $\hat{P}$ 进行平滑滤波，得到平滑预测结果 $\tilde{P}$
5: 利用简单状态机将 $\tilde{P}$ 转换为最终预测结果 P
6: **return** P

1.5. 实验结果

为验证基于线性分类器的语音端点检测算法的有效性，我们在开发集上进行了实验。首先从 `vad/data/dev_label.txt` 中读取音频文件名和标签，对每个音频文件进行语音端点检测，将帧级预测结果和帧级标签分别拼接为长向量，利用 `utils.py` 中的工具函数计算预测的准确率、真阳率、假阳率，绘制出二分类接收特性（ROC）曲线，从而计算曲线覆盖面积（AUC）和等误差率（ERR），作为评估算法的重要指标。

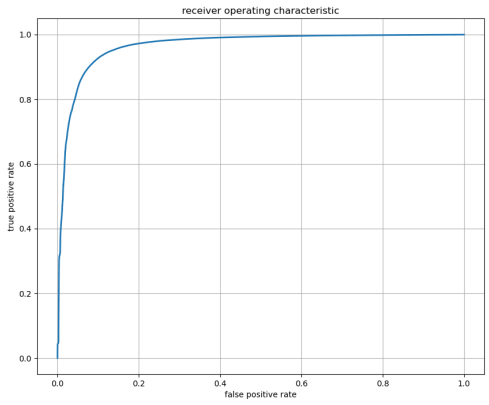

图 7: 接收特性曲线

可以看到，接收特性曲线有显著向左上凸出的特征，当假阳率为 0.05 时，真阳率就可以达到 0.8，因此我们可以认为该二分类任务具有良好的表现。

表 1: 开发集性能表现

ACC	AUC	ERR
0.9171	0.9658	0.0872

从上表可以看到，该算法在开发集上检测的帧级准确率为 91.71%，接收特性曲线覆盖面积为 96.58%，等误差率为 8.72%，在基于语音短时特征和线性分类器的传统方法中，已经具有较好的性能表现。

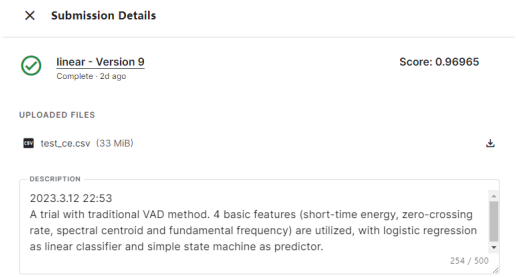

图 8: 提交结果

将该算法应用于测试集上，产生帧级预测结果，再与样例输出文件进行对照，保留有效部分。将预测结果提交至 Kaggle，如上图所示评分结果为 0.96965，与开发集上的结果相符。

2. 基于统计模型分类器和语音频域特征的语音端点检测算法

2.1. 任务描述

基于统计模型分类器和语音频域特征，实现性能较好的语音端点检测算法，要求如下：

- 提取语音的频域特征（如 MFCC、PLP、FBank 等）
- 使用统计模型分类器进行分类（如 GMM、DNN 等）
- 使用简单的状态机完成预测

在本文中，我们采用 FBank 特征和 DNN 模型设计语音端点检测算法。

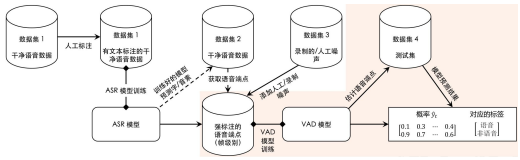

图 9: 任务介绍

2.2. 数据预处理及特征提取

2.2.1. FilterBank 特征提取

FilterBank 特征是语音信号的频域特征，其基本思想是将语音信号的频谱分成若干个频带，将每个频带的能量作为该频带的特征。由于人耳对声音频谱的响应是非线性的，FilterBank 以类似于人耳的方式对音频进行处理，提取音频的频域特征，从而有效提高语音识别的性能。

声源向外辐射声波的时候，空气作为语音信号的载体会对能量产生损耗。在声源尺寸一定的情况下，频率越高，介质对能量的损耗越严重。因此，在读入语音信号后，我们首先进行预加重，以加强信号的高频部分。预加重的公式为：

$$y(n) = x(n) - \alpha \cdot x(n-1) \tag{20}$$

其中 $x(n)$ 为原始语音信号，$y(n)$ 为预加重后的语音信号，α 为预加重系数，一般取 $\alpha = 0.97$。

预加重后，我们将语音信号分帧，其中帧长为 25ms，帧位移为 10ms。对每一帧语音信号，我们需要进行加窗处理，这里我们同样使用 Hamming 窗，定义如下：

$$w(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right) \tag{21}$$

其中 N 为窗的长度，这里我们取 $N = 400$，对应于 25ms 的帧长。

为将时域信号转换为频域信号，我们对每帧信号进行离散傅里叶变换，公式如下：

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-jk\frac{2\pi n}{N}} \tag{22}$$

其中 $x(n)$ 为时域信号，$X(k)$ 为频域信号，k 为离散频谱上的频率，N 为帧长。然后我们将频谱转化为能量谱，公式如下：

$$E(k) = \frac{1}{N} |X(k)|^2 \tag{23}$$

其中 $E(k)$ 为能量谱，$X(k)$ 为频谱，$\frac{1}{N}$ 为归一化系数。傅里叶变换可以通过 `numpy` 中的 `fft.rfft` 函数快速计算。

得到频域能量谱后，我们需要进行梅尔滤波，梅尔频率与真实频率近似具有如下关系：

$$M(f) = 2595 \cdot \lg\left(1 + \frac{f}{700}\right) \tag{24}$$

其中 f 为真实频率，$M(f)$ 为梅尔频率，梅尔频率与人耳的听觉特性是一致的。我们设定频率下限和频率上限，在梅尔频率轴上构造一组等间距的三角滤波器，每个滤波器的中心频率为：

$$M_k = \frac{k}{K+1} (M_{\min} + M_{\max}) \tag{25}$$

其中 $M_{\min}$ 为梅尔频率下限，$M_{\max}$ 为梅尔频率上限，K 为滤波器的个数，这里我们取 $K = 40$。对能量谱进行梅尔滤波后，我们可以得到 K 个输出，取对数后得到的特征向量即为语音信号的 K 维 FilterBank 特征。

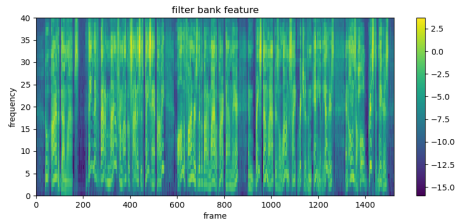

图 10: 频域特征

在 `torchaudio` 包中，我们可以直接调用 `kaldi.fbank` 函数快速计算 FilterBank 特征。从样例音频中提取的 FilterBank 特征如图所示，可以看到，特征图谱与音频波形具有高度相关性，很好地反映了语音信号的状态。

2.2.2. 数据集构造

由于读入音频文件和提取特征的过程会产生较大的时间开销，为了方便模型训练，我们将训练

集和开发集的所有音频文件提取特征并拼接为完整的tensor，保存在.pt 文件中。在训练时，我们将.pt 文件读入内存，并将其转换为 Dataset 类，然后使用 DataLoader 加载数据。

```
[progress] training dataset has been created
size: 5088281
[progress] verifying dataset has been created
size: 704098
```

如日志所示，生成训练集的大小为 5088281，生成开发集的大小为 704098。

2.3. 算法描述

2.3.1. 深度神经网络

为确定语音状态与 FBank 特征的关系，我们使用深度神经网络（DNN）进行学习。根据前文所述，输入为 40 维的 FBank 特征，为实现二分类任务，将输出设定为 1 维，范围在 $[0,1]$ 内，代表预测该帧为语音的概率。

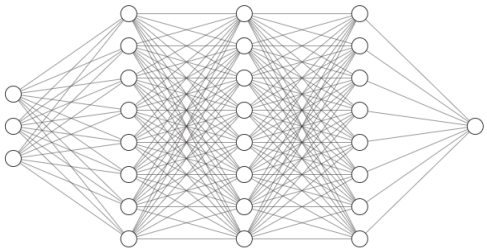

图 11: 网络结构

如图所示，我们使用多层感知机（MLP）的范式，将二分类网络设计为包含三个隐藏层的全连接网络，输入层的维度为 40，代表一帧的 FBank 特征向量，每个隐藏层的维度为 256，使用 BatchNorm 进行归一化，使用 ReLU 作为激活函数，输出层的维度为 1，使用 Sigmoid 激活函数将输出限制在 $[0,1]$ 范围内。整个神经网络大约包含 200k 参数量，具体结构如下所示：

```
(network): Sequential(
  (0): Linear(in_features=40, out_features=256,
    bias=True)
  (1): BatchNorm1d(256, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): Linear(in_features=256, out_features=256,
    bias=True)
  (4): BatchNorm1d(256, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
  (5): ReLU(inplace=True)
  (6): Linear(in_features=256, out_features=256,
    bias=True)
  (7): BatchNorm1d(256, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
  (8): ReLU(inplace=True)
  (9): Linear(in_features=256, out_features=1,
    bias=True)
  (10): Sigmoid()
)
```

训练时，使用 Dataloader 将 1024 个数据作为一个 batch，仍然使用二分类交叉熵（BCE）作为损失函数，其公式为：

$$\phi(y,\hat{y}) = -y\log\hat{y} - (1-y)\log(1-\hat{y}) \qquad (26)$$

其中，y 为真实标签，$\hat{y}$ 为预测标签，实际损失为一个 batch 内的累计损失。使用 Adam 优化器，将学习率设定为 10^{-3}，训练 100 轮。每轮训练完成后，在开发集上进行验证，记录损失和性能并保存模型，选择效果最佳的作为最终模型。

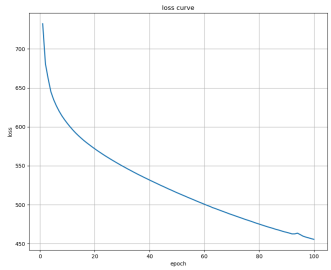

图 12: 损失曲线

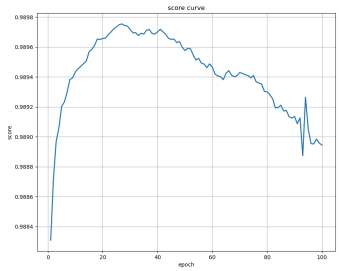

图 13: 性能曲线

在 100 轮训练中，损失曲线不断下降，但性能曲线先上升后下降，在第 27 轮达到峰值，此后

模型开始过拟合。因此，我们选择在第 30 轮左右提前结束训练，以保证模型具有良好的性能。

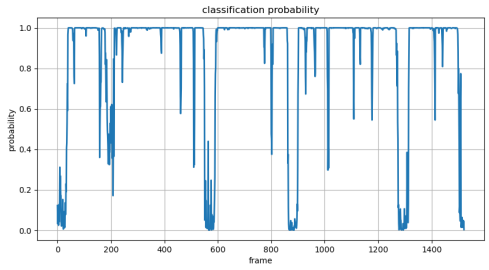

图 14: 原始预测结果

深度神经网络对样例音频产生的概率预测结果如上图所示，可以看出，该模型相比逻辑回归模型具有更好的预测效果。

2.3.2. 平滑滤波与简单状态机

类似地，我们利用语音信号在时间上的连续性，对深度神经网络的原始预测结果进行平滑滤波，以增强预测的可靠性。这里，我们使用宽度为 12 的归一化矩形窗，其公式为：

$$w(n) = \begin{cases} \dfrac{1}{12}, & 0 \le n < 12 \\ 0, & \text{otherwise} \end{cases} \tag{27}$$

滤波操作可以通过 `numpy` 中的 `convolve` 函数实现，平滑滤波后的预测结果参见下图。

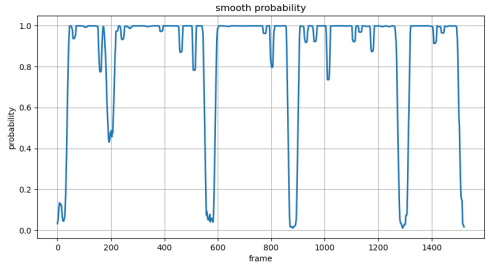

图 15: 平滑预测结果

上图中的平滑预测结果已经和真实标签十分接近，为将概率预测转化为二分类标签，我们同样使用简单状态机进行处理，状态机的定义与前文类似，但由于深度神经网络模型的预测更为准确，我们适当减小转移长度 k，使语音端点检测的延迟尽可能低。

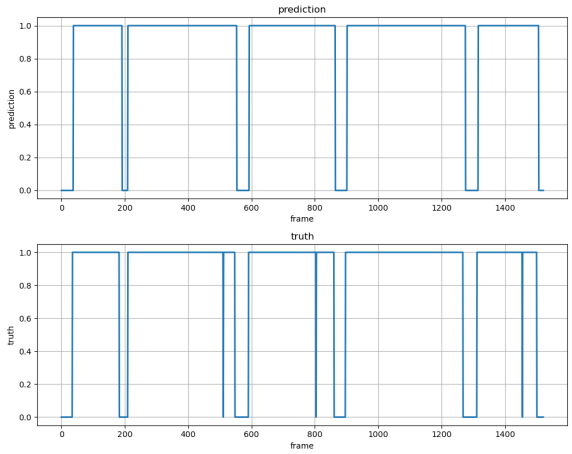

图 16: 最终预测结果

相比逻辑回归模型，深度神经网络模型具有更加优越的检测性能。从上图中可以看出，利用该模型检测语音端点的结果与真实情况几乎完全一致，且检测延迟非常小。我们将算法流程总结如下：

算法 4 基于统计模型分类器的语音端点检测算法

输入 训练音频文件 $\boldsymbol{X}$ 及标签 $\boldsymbol{Y}$，音频文件 Q
输出 预测结果 P

1: 提取 $\boldsymbol{X}$ 的 FBank 特征矩阵 $\boldsymbol{F}$
2: 将 $\boldsymbol{F}$ 和 $\boldsymbol{Y}$ 转换为训练数据集 $\mathcal{D}$
3: 构建深度神经网络 M，在 $\mathcal{D}$ 上训练 M
4: 利用 M 对 Q 进行预测，得到原始预测结果 $\hat{P}$
5: 对 $\hat{P}$ 进行平滑滤波，得到平滑预测结果 $\tilde{P}$
6: 利用简单状态机将 $\tilde{P}$ 转换为最终预测结果 P
7: **return** P

2.4. 实验结果

为验证基于统计模型分类器和语音频域特征的语音端点检测算法的有效性，我们在开发集上进行了实验。与线性分类器的情形类似，首先从 `vad/data/dev_label.txt` 中读取音频文件名和标签，利用该模型对每个音频文件进行语音端点检测，将帧级预测结果和帧级标签分别拼接为长向量，利用 `utils.py` 中的工具函数计算预测的准确率、真阳率、假阳率，绘制出二分类接收特性（ROC）曲线，从而计算曲线覆盖面积（AUC）和等误差率（ERR），作为评估算法的重要指标。

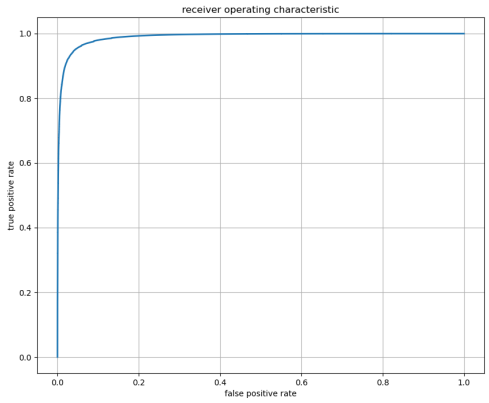

图 17: 接收特性曲线

相比基于线性分类器的算法，基于统计模型分类器的语音端点检测算法的二分类接收特性曲线更向左上角凸出，具有更大的曲线覆盖面积和更小的等误差率，表明后者性能更为优越。

表 2: 开发集性能表现

ACC	AUC	ERR
0.9520	0.9901	0.0466

从上表可以看到，该算法在开发集上检测的帧级准确率为 95.20%，接收特性曲线覆盖面积为 99.01%，等误差率为 4.66%，在基于统计模型分类器的语音端点检测算法中性能优越。

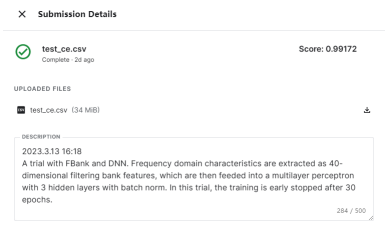

图 18: 提交结果

将该算法应用于测试集上，产生帧级预测结果，再与样例输出文件进行对照，保留有效部分。将预测结果提交至 Kaggle，如上图所示评分结果为 0.99172，与开发集上的结果相符。

3. 附录

3.1. 代码实现

3.1.1. 波形分帧

```python
def divide_wave_to_frame(wave, frame_size=400, frame_shift=160, gate_function='rectangle'):
    """Divide audio wave into frames.

    Args:
        wave: audio wave to be divided
        frame_size: size of each frame
        frame_shift: shift of each frame
        gate_function: rectangle / hamming
    Returns:
        frames: divided frames from audio wave
        timer: corresponding time of each frame

    """
    if gate_function == 'hamming':
        gate = 0.54 - 0.46 * np.cos(2 * np.pi * np.arange(frame_size) / (frame_size - 1))
    else:
        gate = np.ones(frame_size)
    begin, length = 0, len(wave)
    frames = []
    while begin < length:
        end = begin + frame_size
        if end <= length:
            frame = wave[begin:end]
        else:
            frame = np.pad(wave[begin:], (0, end - length), 'constant', constant_values=(0, 0))
        frames.append(gate * frame)
        begin += frame_shift
    return np.array(frames, dtype=float)
```

3.1.2. 短时能量

```python
def short_time_energy(frames, frame_size=400):
    """Evaluate short-time energy of each frame.

    Args:
        frames: two-dimensional array of frames generated by rectangle window
        frame_size: size of each frame
    Returns:
        energy: short-time energy of each frame

    """
    return np.sum(np.square(frames), axis=1) / frame_size
```

3.1.3. 过零率

```
def zero_crossing_rate(frames, frame_size=400):
    """Evaluate zero-crossing rate of each frame.

    Args:
        frames: two-dimensional array of frames generated by rectangle window
        frame_size: size of each frame
    Returns:
        rate: zero-crossing rate of each frame

    """
    return np.sum(0.5 * np.abs(np.sign(frames[:, 1:]) - np.sign(frames[:, :-1])), axis=1) / frame_size
```

3.1.4. 频谱质心

```
def spectral_centroid(frames, sample_rate, frame_size=400):
    """Evaluate spectral centroid of each frame.

    Args:
        frames: two-dimensional array of frames generated by hamming window
        sample_rate: sample rate of audio wave
        frame_size: size of each frame
    Returns:
        centroid: spectral centroid of each frame

    """
    magnitude = np.abs(np.fft.rfft(frames, axis=1)) / frame_size
    frequency = np.fft.rfftfreq(frame_size, 1 / sample_rate)
    return np.sum(magnitude * frequency, axis=1) / np.sum(magnitude)
```

3.1.5. 基频

```
def fundamental_frequency(frames, sample_rate, frame_size=400):
    """Evaluate fundamental frequency of each frame.

    Args:
        frames: two-dimensional array of frames generated by hamming window
        sample_rate: sample rate of audio wave
        frame_size: size of each frame
    Returns:
        frequency: fundamental frequency of each frame

    """
    length, low, high = len(frames), sample_rate // 2000, sample_rate // 20
    extreme = np.zeros(length)
    for i in range(length):
        relate = np.correlate(frames[i], frames[i], mode='full')[frame_size:]
        extreme[i] = low + np.argmax(relate[low:high])
    return extreme
```

3.1.6. 特征向量

```
def wave_feature(wave, sample_rate, frame_size=400, frame_shift=160):
    """Evaluate feature of given audio wave.

    Args:
        wave: audio wave to be evaluated
        sample_rate: sample rate of audio wave
        frame_size: size of each frame
        frame_shift: shift of each frame
    Returns:
        feature: two-dimensional array of wave feature, each line includes four values, which respectively
        stand for short-time energy, zero-crossing rate, spectral centroid and fundamental frequency.

    """
    frames_rectangle = divide_wave_to_frame(wave, frame_size, frame_shift, 'rectangle')
    frames_hamming = divide_wave_to_frame(wave, frame_size, frame_shift, 'hamming')
    ste = short_time_energy(frames_rectangle)
    zcr = 1e0 * zero_crossing_rate(frames_rectangle)
    spc = 1e-1 * spectral_centroid(frames_hamming, sample_rate)
    ffq = fundamental_frequency(frames_hamming, sample_rate)
    return np.stack([1e-3 * np.sqrt(ste), zcr, 1e-1 * spc, 1e-2 * ffq], axis=0).T
```

3.1.7. 平滑滤波

```
def mean_filtering(curve, width=10):
    """Smooth curve by mean filtering.

    Args:
        curve: audio wave to be evaluated
        width: length of filtering window
```

```
    Returns:
        curve: smooth curve generated by mean filtering

    """
    window = np.ones(width) / width
    return np.convolve(curve, window, mode='same')
```

3.1.8. 简单状态机

```
def generate_prediction(probability, holder=8):
    """Convert probability to prediction by simple state machine.

    Args:
        probability: probability array of voice in each frame
        holder: minimum number of frames for state transition
    Returns:
        prediction: prediction generated by state machine

    """
    prediction = []
    active = False
    for i in range(len(probability)):
        state = probability[max(0, i-holder+1): i+1]
        if active and np.max(state) < 0.5:
            active = False
        elif not active and np.min(state) > 0.5:
            active = True
        prediction.append(int(active))
    return np.array(prediction)
```

3.1.9. 接收特性曲线

```
def get_metrics(prediction, label):
    """Calculate metrics for a binary classification task.

    Args:
        prediction: sequence of probabilities
        label: sequence of class labels
    Returns:
        auc: area under curve
        eer: equal error rate
        fpr: false positive rate
        tpr: true positive rate

    """
    assert len(prediction) == len(label), (len(prediction), len(label))
    fpr, tpr, _ = metrics.roc_curve(label, prediction, pos_label=1)
    auc = metrics.auc(fpr, tpr)
    eer, _ = compute_eer(
        [pred for i, pred in enumerate(prediction) if label[i] == 1],
        [pred for i, pred in enumerate(prediction) if label[i] == 0],
    )
    return auc, eer, fpr, tpr
```

3.1.10. 逻辑回归训练

```
data = utils.read_label_from_file('./data/train_label.txt')
feature_list, label_list = [], []
for name, value in tqdm(data.items()):
    path = os.path.join('./wavs/train', name + '.wav')
    sample_rate, wave = wavfile.read(path)
    feature = utils.wave_feature(wave, sample_rate)
    label = np.pad(np.array(value), (0, feature.shape[0] - len(value)), 'constant', constant_values=(0, 0))
    feature_list.append(feature)
    label_list.append(label)
feature_list = np.concatenate(feature_list, axis=0)
label_list = np.concatenate(label_list, axis=0)
print('[progress] training dataset has been created')
print('feature: {}, label: {}'.format(feature_list.shape, label_list.shape))

classifier = LogisticRegression(penalty='l1', solver='liblinear')
classifier.fit(feature_list, label_list)
print('[progress] classifier training has been finished')
print('weight: {}'.format(classifier.coef_[0]))
return classifier
```

3.1.11. 频域特征数据集

```
class VAD_Dataset(Dataset):
    def __init__(self, mode='test'):
        save_path = os.path.join('./data', mode + '_data.pt')
        if os.path.exists(save_path):
            self.feature, self.label = torch.load(save_path)
        else:
```

```
            data_path = os.path.join('./data', mode + '_label.txt')
            wave_path = os.path.join('./wavs', mode)
            data = utils.read_label_from_file(data_path)
            feature_list, label_list = [], []
            for name, value in tqdm(data.items()):
                path = os.path.join(wave_path, name + '.wav')
                wave, rate = torchaudio.load(path)
                feature = kaldi.fbank(wave, sample_frequency=rate, num_mel_bins=40, snip_edges=False)
                label = F.pad(torch.Tensor(value), (0, feature.shape[0] - len(value)), 'constant', 0)
                feature_list.append(feature)
                label_list.append(label)
            self.feature = torch.cat(feature_list, dim=0)
            self.label = torch.cat(label_list, dim=0)
            torch.save((self.feature, self.label), save_path)

    def __getitem__(self, index):
        return self.feature[index], self.label[index]

    def __len__(self):
        return self.feature.shape[0]
```

3.1.12. 深度神经网络模型

```
class VAD_Model(nn.Module):
    def __init__(self, input_dim=40, hidden_dim=256, output_dim=1):
        super(VAD_Model, self).__init__()
        linear_layer = [
            nn.Linear(input_dim, hidden_dim),
            nn.Linear(hidden_dim, hidden_dim),
            nn.Linear(hidden_dim, hidden_dim),
            nn.Linear(hidden_dim, output_dim)
        ]
        norm_layer = [
            nn.BatchNorm1d(hidden_dim),
            nn.BatchNorm1d(hidden_dim),
            nn.BatchNorm1d(hidden_dim),
        ]
        for layer in linear_layer:
            nn.init.xavier_uniform_(layer.weight)
            nn.init.zeros_(layer.bias)
        self.network = nn.Sequential(
            linear_layer[0],
            norm_layer[0],
            nn.ReLU(True),
            linear_layer[1],
            norm_layer[1],
            nn.ReLU(True),
            linear_layer[2],
            norm_layer[2],
            nn.ReLU(True),
            linear_layer[3],
            nn.Sigmoid()
        )

    def forward(self, inputs):
        outputs = self.network(inputs)
        return outputs
```

3.1.13. 深度神经网络训练

```
dataset = VAD_Dataset('train')
loader = DataLoader(dataset, batch_size=1024, drop_last=False)
print('[progress] training dataset has been created')
print('size: {}'.format(len(dataset)))

model = VAD_Model(input_dim=40, hidden_dim=256, output_dim=1).to(device)
criterion = nn.BCELoss().to(device)
optimizer = optim.Adam(model.parameters(), 1e-3)
for epoch in range(total_epoch):
    print('[progress] epoch {} has been started'.format(epoch + 1))
    epoch_loss = 0
    with tqdm(loader) as pbar:
        for feature, label in pbar:
            feature, label = feature.to(device), label.to(device)
            result = model(feature).squeeze(-1)
            loss = criterion(result, label)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()
            pbar.set_postfix({'loss': loss.item()})
    print('epoch: {}, loss: {}'.format(epoch + 1, epoch_loss))
print('[progress] model training has been finished')
return model
```

3.2. 输出日志

3.2.1. 基于线性分类器的语音端点检测

```
100%|          | 3600/3600 [08:27<00:00,  7.09it/s]
[progress] training dataset has been created
feature: (5088315, 4), label: (5088315,)
[progress] classifier training has been finished
weight: [  2.0984647   -3.06644603 590.35748186   1.39601653]
100%|          | 500/500 [01:11<00:00,  6.99it/s]
[progress] prediction has been generated
probability: (704100,), prediction: (704100,), reality: (704100,)
[progress] estimation has been finished
acc: 0.9171239880698764, auc: 0.9658173588265918, err: 0.0872433007490604
100%|          | 1000/1000 [02:26<00:00,  6.84it/s]
[progress] prediction has been generated
included: 1415031, required: 1388273
[progress] result has been standardized
included: 1388273, required: 1388273
```

3.2.2. 基于统计模型分类器的语音端点检测

```
[progress] dataset has been created
train: 5088281, estimate: 704098
[progress] epoch 1 has been started
loss: 732.3116237185895, score: 0.9883090137580047
[progress] epoch 2 has been started
loss: 680.9195043649524, score: 0.9887166582437883
[progress] epoch 3 has been started
loss: 661.7241319939494, score: 0.988969009241415
[progress] epoch 4 has been started
loss: 644.9289485402405, score: 0.98905917501288
[progress] epoch 5 has been started
loss: 634.8673458173871, score: 0.9892023710877255
[progress] epoch 6 has been started
loss: 626.8827902097255, score: 0.9892325949575993
[progress] epoch 7 has been started
loss: 620.0613387618214, score: 0.9892969681933047
[progress] epoch 8 has been started
loss: 614.1238299626857, score: 0.9893818456664741
[progress] epoch 9 has been started
loss: 609.038866456598, score: 0.9893937927506413
[progress] epoch 10 has been started
loss: 604.3688704613596, score: 0.9894331978992698
[progress] epoch 11 has been started
loss: 600.0258091837168, score: 0.9894549587333387
[progress] epoch 12 has been started
loss: 595.9600198976696, score: 0.9894700439433909
[progress] epoch 13 has been started
loss: 592.1911593489349, score: 0.9894891473525957
[progress] epoch 14 has been started
loss: 588.7875602114946, score: 0.989505564323825
[progress] epoch 15 has been started
loss: 585.6014908663929, score: 0.9895670759688224
[progress] epoch 16 has been started
loss: 582.5040867626667, score: 0.9895836700207653
[progress] epoch 17 has been started
loss: 579.6864415258169, score: 0.9896093667845716
[progress] epoch 18 has been started
loss: 577.0092806182802, score: 0.9896524291462732
[progress] epoch 19 has been started
loss: 574.3574260547757, score: 0.9896503898027466
[progress] epoch 20 has been started
loss: 571.8087717387825, score: 0.989656046452142
[progress] epoch 21 has been started
loss: 569.325361000374, score: 0.9896586548369097
[progress] epoch 22 has been started
loss: 566.9705189038068, score: 0.9896818785148463
[progress] epoch 23 has been started
loss: 564.6217442080379, score: 0.9896999937837825
[progress] epoch 24 has been started
loss: 562.3587363027036, score: 0.9897210941542889
[progress] epoch 25 has been started
loss: 560.1922410484403, score: 0.9897334828595035
[progress] epoch 26 has been started
loss: 558.0198783129454, score: 0.9897487431622473
[progress] epoch 27 has been started
loss: 555.9319119080901, score: 0.9897540221075002
[progress] epoch 28 has been started
loss: 553.868179006502, score: 0.9897428871702958
[progress] epoch 29 has been started
loss: 551.859291575849, score: 0.9897395744453892
[progress] epoch 30 has been started
loss: 549.8471222929657, score: 0.9897173071994392
[progress] epoch 31 has been started
loss: 547.9439091552049, score: 0.9896943947136381
[progress] epoch 32 has been started
loss: 545.9724533017725, score: 0.9896953402782167
[progress] epoch 33 has been started
loss: 544.1506149284542, score: 0.9896764912357437
[progress] epoch 34 has been started
loss: 542.233340350911, score: 0.9896913006511238
[progress] epoch 35 has been started
loss: 540.4133467953652, score: 0.9896860091906727
[progress] epoch 36 has been started
loss: 538.566174628213, score: 0.9897120655462746
[progress] epoch 37 has been started
loss: 536.866145838052, score: 0.9897167648230419
[progress] epoch 38 has been started
loss: 535.0760048720986, score: 0.9896906684201404
[progress] epoch 39 has been started
loss: 533.4012938691303, score: 0.9896853895015723
[progress] epoch 40 has been started
loss: 531.6550378482789, score: 0.9897006884439896
[progress] epoch 41 has been started
loss: 529.995263967663, score: 0.9897181625491863
[progress] epoch 42 has been started
```

```
loss: 528.3385586533695, score: 0.9897009374870924
[progress] epoch 43 has been started
loss: 526.6860448848456, score: 0.9896846848879133
[progress] epoch 44 has been started
loss: 524.9739827597514, score: 0.9896571579765201
[progress] epoch 45 has been started
loss: 523.4194902274758, score: 0.9896505829477443
[progress] epoch 46 has been started
loss: 521.7891249787062, score: 0.9896523273035149
[progress] epoch 47 has been started
loss: 520.13334343303, score: 0.989629276450057
[progress] epoch 48 has been started
loss: 518.6374864606187, score: 0.9896362036522378
[progress] epoch 49 has been started
loss: 517.0000530574471, score: 0.9895987436562607
[progress] epoch 50 has been started
loss: 515.4418589286506, score: 0.9895757871004176
[progress] epoch 51 has been started
loss: 513.9283475056291, score: 0.9895903148971552
[progress] epoch 52 has been started
loss: 512.4190248083323, score: 0.9895885757182469
[progress] epoch 53 has been started
loss: 510.9520773459226, score: 0.989546852422702
[progress] epoch 54 has been started
loss: 509.38938683923334, score:
    0.9895121359836373
[progress] epoch 55 has been started
loss: 507.94059282541275, score:
    0.9895240935616247
[progress] epoch 56 has been started
loss: 506.48876818083227, score:
    0.9894922066914393
[progress] epoch 57 has been started
loss: 504.95483117364347, score:
    0.9894852412898851
[progress] epoch 58 has been started
loss: 503.5004099421203, score: 0.9894615754305193
[progress] epoch 59 has been started
loss: 502.07601037062705, score:
    0.9894868932960356
[progress] epoch 60 has been started
loss: 500.6455768439919, score: 0.9894639334712974
[progress] epoch 61 has been started
loss: 499.19594351481646, score:
    0.9894179825138121
[progress] epoch 62 has been started
loss: 497.85895635932684, score:
    0.9894064644954665
[progress] epoch 63 has been started
loss: 496.28229978773743, score:
    0.9894015782846022
[progress] epoch 64 has been started
loss: 495.0841580396518, score: 0.9893814466277882
[progress] epoch 65 has been started
loss: 493.7492369124666, score: 0.9894238674521624
[progress] epoch 66 has been started
loss: 492.44828665815294, score:
    0.9894420652907183
[progress] epoch 67 has been started
loss: 491.12517111282796, score:
    0.9894117854626986
[progress] epoch 68 has been started
loss: 489.84034428559244, score:
    0.9894001103372555
[progress] epoch 69 has been started
loss: 488.4618778983131, score: 0.9894081758087124
[progress] epoch 70 has been started
loss: 487.24740630807355, score:
    0.9894298921391107
[progress] epoch 71 has been started
loss: 486.0384118380025, score: 0.9894228259488501
[progress] epoch 72 has been started
loss: 484.7567378524691, score: 0.9894139147341945
[progress] epoch 73 has been started
loss: 483.5676411315799, score: 0.9894096510210117
[progress] epoch 74 has been started
loss: 482.3075234517455, score: 0.9893933779478757
[progress] epoch 75 has been started
loss: 481.04697895096615, score:
    0.9894091012795937
[progress] epoch 76 has been started
loss: 479.9353599003516, score: 0.9893671745320249
[progress] epoch 77 has been started
loss: 478.76319141034037, score: 0.989358668880503
[progress] epoch 78 has been started
loss: 477.5243477327749, score: 0.9893518131136854
[progress] epoch 79 has been started
loss: 476.2642886652611, score: 0.9893022895341247
[progress] epoch 80 has been started
loss: 475.17832685355097, score:
    0.9893006548064194
[progress] epoch 81 has been started
loss: 473.99231961462647, score:
    0.9892799359493188
[progress] epoch 82 has been started
loss: 472.85560407303274, score:
    0.9892541714660158
[progress] epoch 83 has been started
loss: 471.705199487973, score: 0.989193270650746
[progress] epoch 84 has been started
loss: 470.70153172826394, score:
    0.9891943985062568
[progress] epoch 85 has been started
loss: 469.66087824106216, score:
    0.9892114184018249
[progress] epoch 86 has been started
loss: 468.50024257740006, score: 0.989172076796513
[progress] epoch 87 has been started
loss: 467.58024852350354, score:
    0.9891762290937451
[progress] epoch 88 has been started
loss: 466.5157900447957, score: 0.9891341206089552
[progress] epoch 89 has been started
```

```
loss: 465.3969310624525, score: 0.9891242812549328
[progress] epoch 90 has been started
loss: 464.5759692918509, score: 0.9891360668890417
[progress] epoch 91 has been started
loss: 463.60280774813145, score: 0.98908665647329
[progress] epoch 92 has been started
loss: 462.5365690337494, score: 0.9891262351068475
[progress] epoch 93 has been started
loss: 462.5669424254447, score: 0.9888730620555042
[progress] epoch 94 has been started
loss: 463.46049676742405, score:
    0.9892639703657945
[progress] epoch 95 has been started
loss: 461.63016812969, score: 0.9890521527392234
[progress] epoch 96 has been started
loss: 459.66610797541216, score:
    0.9889545858691956
[progress] epoch 97 has been started
loss: 458.4536317002494, score: 0.9889518127387897
[progress] epoch 98 has been started
loss: 457.51193633070216, score:
    0.9889846612845367
[progress] epoch 99 has been started
loss: 456.4456641601864, score: 0.9889584516244291
[progress] epoch 100 has been started
loss: 455.54815990501083, score:
    0.9889445522360689
[progress] model training has been finished
[progress] prediction has been generated
included: 1415022, required: 1388273
[progress] result has been standardized
included: 1388273, required: 1388273
```