对抗实验:企业财务困境预测报告

专　　业:　信管创新班 20-1 班

姓名: 史康威 徐梦理 高纪铭 贾其豪 金硕

老　　师:　王刚 王钊 任刚

时　　间:　2023 年 4 月

小组对抗实验：企业财务困境预测报告

摘要

此实验的目的为用一些数值型指标与不等数目的中文文本指标预测一个指标数值为 1 的概率。我们首先对中文文本进行向量化处理然后在用向量化后的数据与其提供的数值型指标一起来进行模型的训练预测。在此过程中我们尝试了很多模型结构，并采用了提取一部分数据来进行训练预测等方法来加快模型筛选。在中文文本向量化的过程中，我们尝试了 CountVectorizer 和 Bert 预训练模型进行编码两种方式，并最终采用了 Bert 编码。对于在不等长度的中文文本向量中如何确定最终参与训练的向量的问题上，我们首先采用了只取最新的文本向量参与，再采用每个企业采用前十个文本来进行训练。最终的结果表明采用第一个文本向量作为指标效果比较好。由于 Bert 编码之后的向量为 768 维而其数值指标为 52 维，如果直接拼接则更加偏向于文本信息，使得数值信息相对而言并没有得到很好的体现。对此问题，我们首先将文本信息通过一层 MLP 进行降维处理，将文本信息从 768 维降到 100 维，再进行拼接，输入到两层 MLP 和一层 Sigmoid 函数来进行处理，其输出便为对其数值为 1 的预测。在第三次实验的过程中我们又对深度神经网络进行了其他变式的探索比如增加 LSTM 层来对文本数据进行处理，使其能从文本数据集中获取到更多相关信息，以及试采用 Self-Attention 模型来提取更多有用信息。我们发现 LSTM 在相同情况下比 Attention 的效果要好一些。我们猜测可能是数据量较小 Attention 模型没能发挥出其优越性，而 LSTM 更加适用于小型数据集的缘故。

最终我们小组以 AUC=0.94633 取得了第二名的好成绩。

0. 实验题目简介：

实验背景：

实验以金融风控中的企业财务困境预测为背景，要求同学根据企业年报以及临时公告的数据信息预测其是否会出现困境，是一个典型的分类问题。通过本实验来引导大家了解商务数据分析中的一些业务背景，解决实际问题。

实验数据：

实验以预测新三板挂牌企业是否出现财务困境为任务，数据包含 2017 年挂牌的 8040 家企业样本，其中正常公司 7871 家，非正常公司 169 家。财务数据来自三大财务报表，特征总数量 106，文本数据包含 306121 条企业临时公告。本次对抗实验抽取 90%的样本作为训练集，10%的样本作为测试集，数据集将会对股票代码等信息进行脱敏。

结果提交及评测标准:本次对抗实验需提交完整程序和最终实验结果集 results.csv，实验评价指标为 AUC。

AUC

sklearn.metrics.roc_auc_score¶

sklearn.metrics.**roc_auc_score**(*y_true*, *y_score*, ***, *average='macro'*, *sample_weight=None*, *max_fpr=None*, *multi_class='raise'*, *labels=None*) [source]

1.原理介绍

1.1CountVectorizer

CountVectorizer 是属于常见的特征数值计算类，是一个文本特征提取方法。对于每一个训练文本，它只考虑每种词汇在该训练文本中出现的频率。

CountVectorizer 会将文本中的词语转换为词频矩阵，它通过 fit_transform 函数计算各个词语出现的次数。

CountVectorizer 参数详解:

CountVectorizer 类的参数很多，分为三个处理步骤：preprocessing、tokenizing、n-grams generation.

一般要设置的参数是:ngram_range,max_df，min_df，max_features 等，具体情况具体分析

参数表	作用
input	一般使用默认即可，可以设置为"filename'或'file'
encodeing	使用默认的utf-8即可，分析器将会以utf-8解码raw document
decode_error	默认为strict，遇到不能解码的字符将报UnicodeDecodeError错误，设为ignore将会忽略解码错误，还可以设为replace，作用尚不明确
strip_accents	默认为None，可设为ascii或unicode，将使用ascii或unicode编码在预处理步骤去除raw document中的重音符号
analyzer	一般使用默认，可设置为string类型，如'word', 'char', 'char_wb'，还可设置为callable类型，比如函数是一个callable类型
preprocessor	设为None或callable类型
tokenizer	设为None或callable类型
ngram_range	词组切分的长度范围，待详解
stop_words	设置停用词，设为english将使用内置的英语停用词，设为一个list可自定义停用词，设为None不使用停用词，设为None且max_df∈[0.7, 1.0)将自动根据当前的语料库建立停用词表
lowercase	将所有字符变成小写
token_pattern	过滤规则，表示token的正则表达式，需要设置analyzer == 'word'，默认的正则表达式选择2个及以上的字母或数字作为token，标点符号默认当作token分隔符，而不会被当作token

属性表	作用
vocabulary_	词汇表；字典型
get_feature_names()	所有文本的词汇；列表型
stop_words_	返回停用词表

方法表	作用
fit_transform(X)	拟合模型，并返回文本矩阵
fit(raw_documents[, y])	Learn a vocabulary dictionary of all tokens in the raw documents.
fit_transform(raw_documents[, y])	Learn the vocabulary dictionary and return term-document matrix.

用数据输入形式为列表，列表元素为代表文章的字符串，一个字符串代表一篇文章，字符串是已经分割好的。CountVectorizer 同样适用于中文;

CountVectorizer 是通过 fit_transform 函数将文本中的词语转换为词频矩阵，矩阵元素 a[i][j] 表示 j 词在第 i 个文本下的词频。即各个词语出现的次数，通过 get_feature_names()可看到所有文本的关键字，通过 toarray()可看到词频矩阵的结果。

1.2Bert 预训练模型

BERT（Transformers 的双向编码器表示)于 2018 年底发布。BERT 是一种预训练语言表示的方法，是 NLP 实践者可以免费下载和使用的模型。可以使用这些模型从文本数据中提取高质量的语言功能，也可以在特定任务（分类、实体识别、问题解答等)上使用自己的数据对这些模型进行微调，以生成最新的预测。

使用 bert 从文本数据中提取特征，即单词和句子 embedding vectors(嵌入向量)。这些嵌入对于关键字/搜索扩展、语义搜索和信息检索很有用。例如，如果希望匹配客户问题或针对已回答问题或有良好文档记录的搜索，这些表示将帮助您准确地检索与客户意图和上下文含义匹配的结果，即使没有关键字或短语重叠。

其次，也许更重要的是，这些向量被用作下游模型的高质量特征输入。像 lstms 或 cnns 这样的 nlp 模型需要以数字向量的形式输入，这通常意味着将词汇和部分演讲等特征转换为数字表示。在过去，单词要么被表示为唯一的索引值（独热编码)，要么更有用地表示为神经单词嵌入，其中词汇单词与固定长度的特征嵌入相匹配，这是由 Word2Vec 或 FastText 等模型产生的。BertNord2vec 等模型更具优势，因为每个单词在 Word2vec 下都有一个固定的表示，而不管单词出现在什么上下文中，Bert 都会生成由它们周围的单词动态 inform 的单词表示。例如，给出两个句子:

```
1 “The man was accused of robbing a bank.”
2 “The man went fishing by the bank of the river.”
```

word2vec 将在两个句子中为单词"bank"生成相同的词嵌入，而在 BERT 下，“bank'"的单词嵌入对于每个句子都是不同的。除了捕获明显的差异(如多义)，上下文通知的单词嵌入还捕获其他形式的信息，这些信息会导致更精确的特征表示，进而导致更好的模型性能。

由于 bert 是一个预训练模型，它期望输入数据采用特定格式，因此我们需要:

- special tokens to mark the beginning ([CLS]) and separation/end of sentences ([SEP])
- tokens that conforms with the fixed vocabulary used in BERT
- token IDs from BERT's tokenizer
- mask IDs to indicate which elements in the sequence are tokens and which are padding elements
- segment IDs used to distinguish different sentences
- positional embeddings used to show token position within the sequence

从隐藏状态创建单词和句子向量

现在，我们如何处理这些隐藏的状态?我们希望为每个 token 获取单独的向量，或者可能是整个句子的单个向量表示，但是对于我们输入的每个 token，我们有 12 个单独的向量，每个向量的长度为 768

为了获得单个向量，我们需要组合一些层向量……但哪个层或层组合提供了最佳表示?BERT 作者通过将不同的向量组合作为输入特征输送到用于命名实体识别任务的 BiLSTM 并观察得到的 F1 分数来测试这一点。

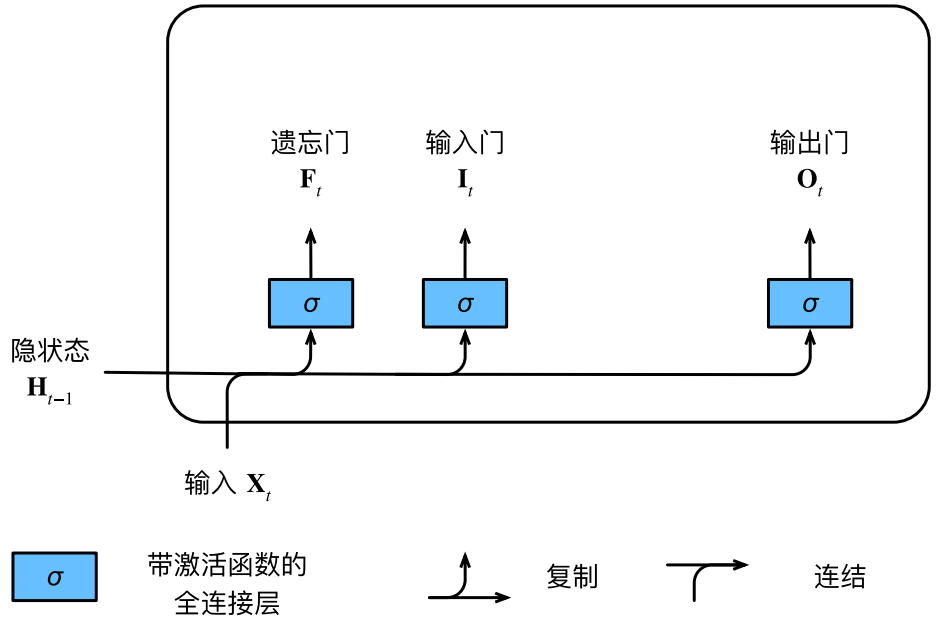
虽然最后四层的连接在这个特定任务上产生了最好的结果，但许多其他方法紧随其后，一般来说，建议为您的特定应用测试不同的版本:结果可能会有所不同。通过注意 BERT 的不同层编码非常不同类型的信息来部分地证明这一点，因此适当的池化策略将根据应用而改变，因为不同的层编码不同类型的信息。

如何获得句向量呢？为了获得整个句子的单个向量，我们有多个依赖于应用的策略，但是一个简单的方法是平均每个 token 的倒数第二层，产生一个 768 长度向量。

1.3LSTM

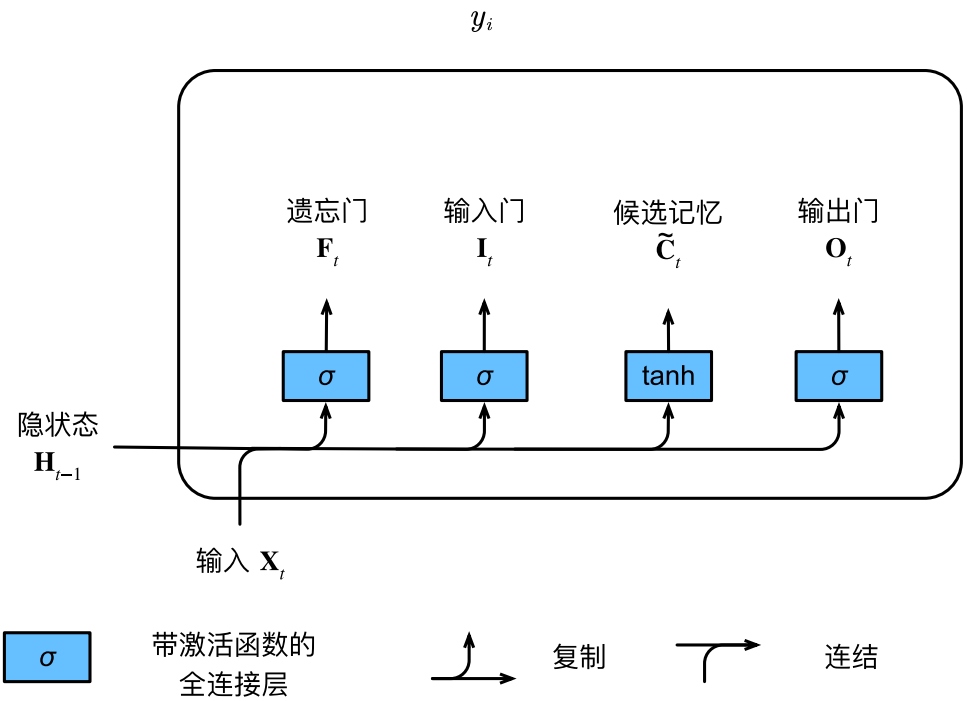
长期以来，隐变量模型存在着长期信息保存和短期输入缺失的问题。解决这一问题的最早方法之一是长短期存储器（(long short-term memory， LSTM)。它有许多与门控循环单元一样的属性。有趣的是，长短期记忆网络的设计比门控循环单元稍微复杂一些，却比门控循环单元早诞生了近 20 年。

门控记忆单元：就如在门控循环单元中一样，当前时间步的输入和前一个时间步的隐状态作为数据送入长短期记忆网络的门中。它们由三个具有 sigmoid 激活函数的全连接层处理，以计算输入门、遗忘门和输出门的值。因此，这三个门的值都在(0,1)的范围内。



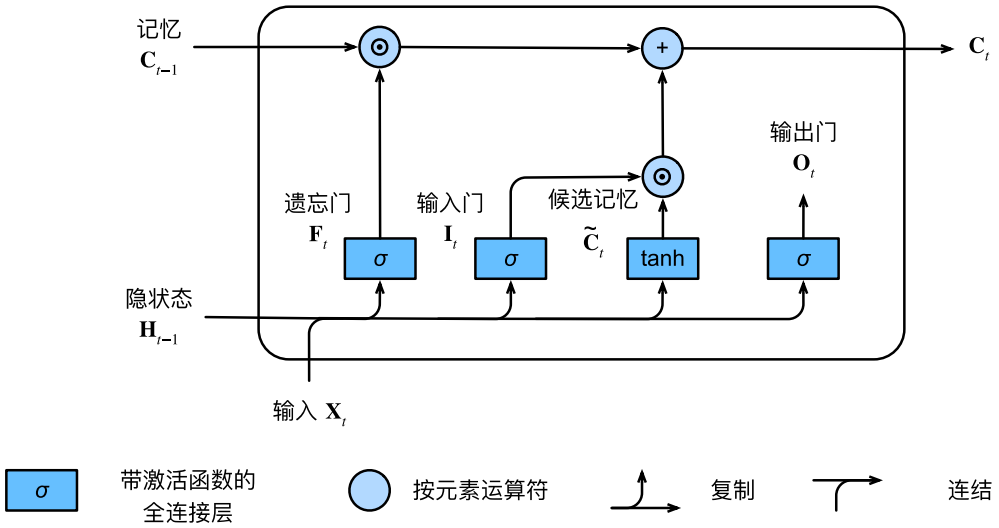
$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i),$$
$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f),$$
$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o),$$

候选记忆单元：由于还没有指定各种门的操作，所以先介绍候选记忆元(candidate memory cell) $\tilde{C}_t \in \mathbb{R}^{n*h}$。它的计算与上面描述的三个门的计算类似，但是使用 tanh 函数作为激活函数，函数的值范围为(—1,1)。下面导出在时间步 t 处的方程:

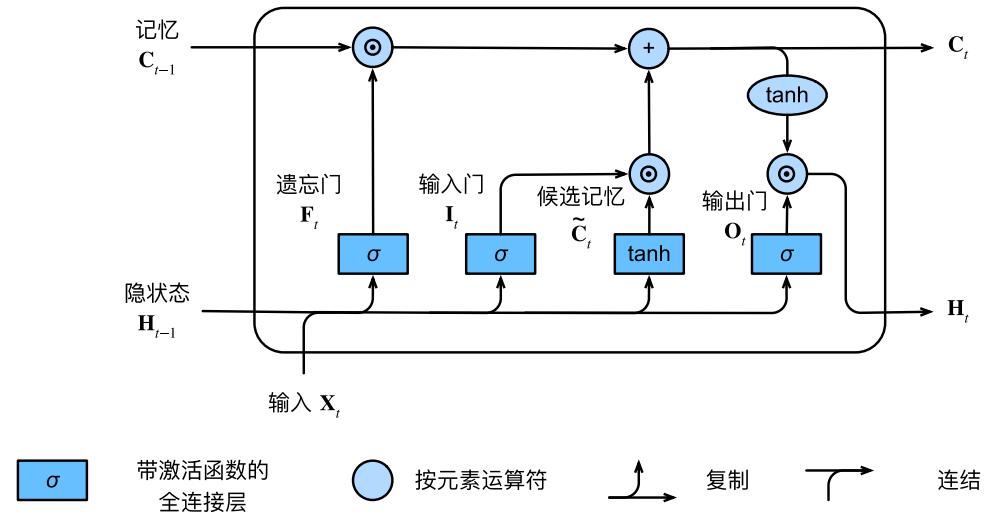


记忆元：在门控循环单元中，有一种机制来控制输入和遗忘(或跳过)。类似地，在长短期记忆网络中，也有两个门用于这样的目的:输入门I_t,控制采用多少来自$\tilde{\mathbf{C}}_t$的新数据，而遗忘门F_t控制保留多少过去的记忆元$C_{t-1} \in \mathbb{R}^{n*h}$的内容。使用按元素乘法，得出:

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t.$$



隐状态：最后，我们需要定义如何计算隐状态$H_t \in \mathbb{R}^{n*h}$，这就是输出门发挥作用的地方。在长短期记忆网络中，它仅仅是记忆元的 tanh 的门控版本。这就确保了H_t的值始终在区间(-1,1)内:



1.4Self-Attention

在深度学习中，经常使用卷积神经网络〈CNN)或循环神经网络(RNN)对序列进行编码。想象一下，有了注意力机制之后，我们将词元序列输入注意力池化中，以便同一组词元同时充当查询、键和值。具体来说，每个查询都会关注所有的键一值对并生成一个注意力输出。由于查询、键和值来自同一组输入，因此被称为自注意力(self-attention)，也被称为内部注意力(intra-attention)。

自注意力：给定一个由词元组成的输入序列 x];...,Xn，其中任意$x_i \in \mathbb{R}^d$ (1<i<n)。该序列的自注意力输出为一个长度相同的序列 y1,.. . , yn，其中:

$$\mathbf{y}_i = f(\mathbf{x}_i, (\mathbf{x}_1, \mathbf{x}_1), ..., (\mathbf{x}_n, \mathbf{x}_n)) \in \mathbb{R}^d$$

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} \alpha(x, x_i) y_i \\ &= \sum_{i=1}^{n} \frac{\exp\left(-\frac{1}{2}(x - x_i)^2\right)}{\sum_{j=1}^{n} \exp\left(-\frac{1}{2}(x - x_j)^2\right)} y_i \\ &= \sum_{i=1}^{n} \mathrm{softmax}\left(-\frac{1}{2}(x - x_i)^2\right) y_i. \end{aligned}$$

如果一个键x_i越是接近给定的查询x，那么分配给这个键对应值y_i的注意力权重就会越大，也就"获得了更多的注意力"。

1.5Sigmoid

sigmoid 函数也叫 Logistic 函数，用于隐层神经元输出，取值范围为(0,1)，它可以将一个实数映射到(0,1)的区间，可以用来做二分类。在特征相差比较复杂或是相差不是特别大时效果比较好。Sigmoid 作为激活函数有以下优缺点:

优点：平滑、易于求导。

缺点：激活函数计算量大，反向传播求误差梯度时，求导涉及除法；反向传播时，很容易就会出现梯度消失的情况，从而无法完成深层网络的训练。

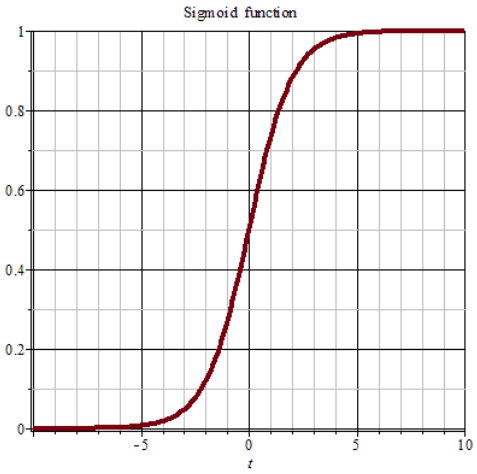
Sigmoid 函数由下列公式定义

$$S(x) = \frac{1}{1 + e^{-x}}$$

其对 x 的导数可以用自身表示:

$$S'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = S(x)(1 - S(x))$$

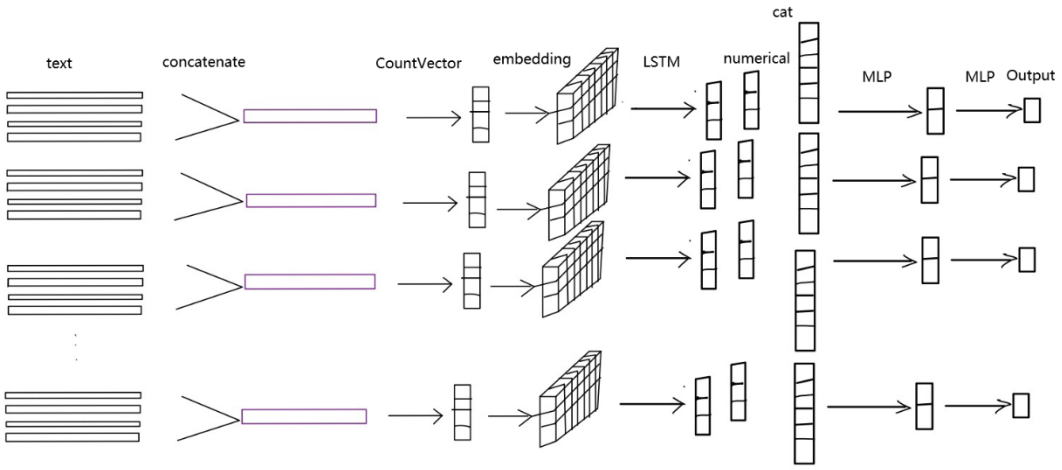
Sigmoid 函数的图形如 S 曲线：



2.架构设计

2.1 第一次提交

模型结构如下图所示：

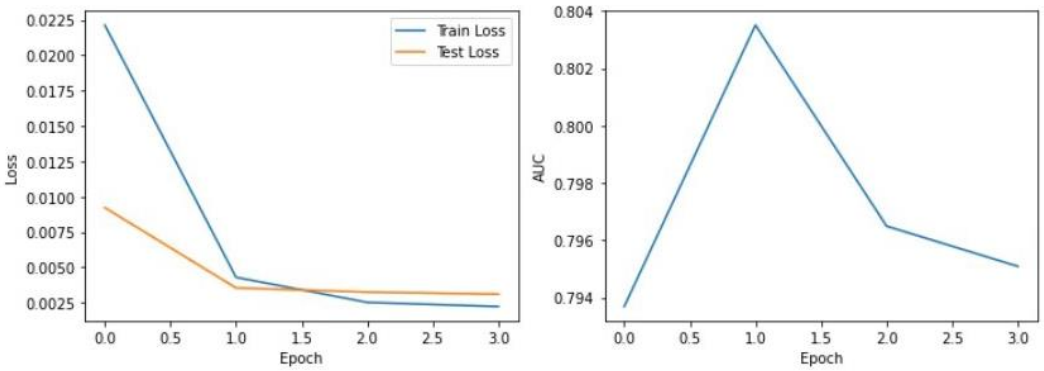


首先将每个公司的所有评论按照时间顺序连接，使其成为一个超长的文本。再对每一条文本进行 embedding 操作，这里用的方法是 CountVectorizer 方法对其进行的处理。由于只是一个简单的计数统计，其编码的向量并不具有语义，并且编码后的向量非常的长，为 42668 维，在此情况下再将得到的向量进行 embedding 操作，这一步骤其实完全可以省略，这里采取此种操作完全是因为什么都不懂。

之后将 embedding 过后的向量进行 LSTM 操作，并去每一个向量的最后一个值，并将其与 numerical 的数值指标进行连接操作。

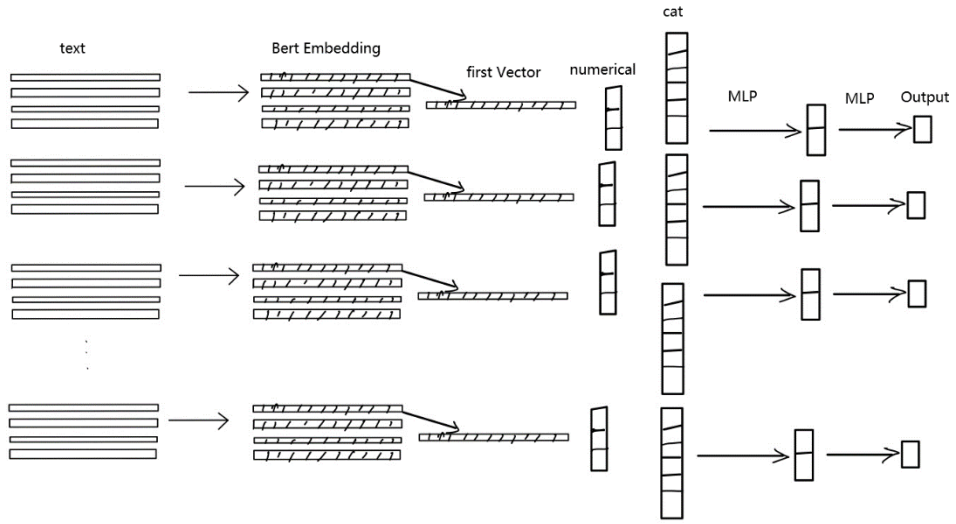
最后我们将连接到的最终的特征向量传入神经网络，进行目标值的预测。最后再根据梯度下降算法进行误差的反向传播。

最终结果为（由于进行了很多无效操作，下面结果为电脑跑了一个晚上才跑出来）：



2.2 第二次提交

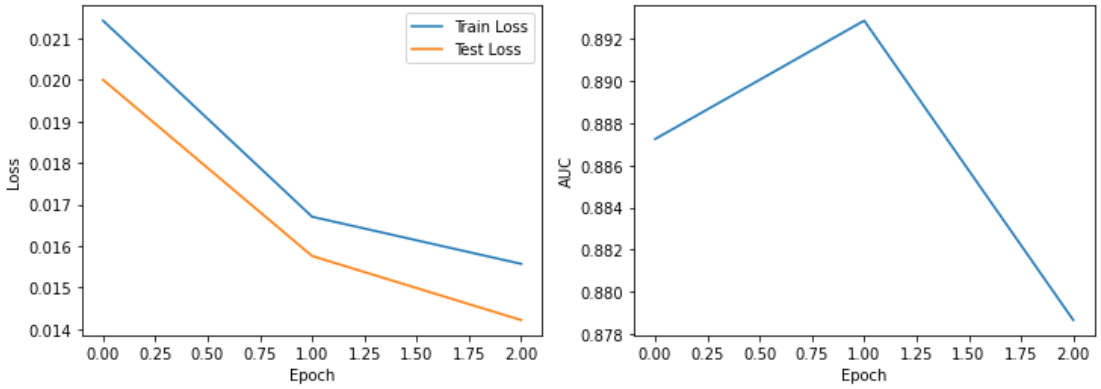
模型结构如下图所示：



第二次提交相对于第一次提交有很多改进之处。在对文本的编码层面，我们选用了能够保留语义信息的 bert 编码来对中文文本进行向量化处理。但是由于时间原因，我们仅采用了每一个公司的最近一条信息来进行预测。因为我们认为针对企业的财务困境信息，最近的一条公司公告所含的信息量会最大。

将每一家公司对应的最新的一条公告向量化之后，我们并没有再进行多于的 embedding 操作（第一次里面的对向量化后的文本值再进行 embedding 操作完全是没有用的。）。再尝试了多次将向量化后的 768 维的数据降维后，我们发现最终实验结果远没有不降维直接连接的效果好。因此，我们将向量化后的数据直接与数值型数据进行拼接，得到代表全体指标的向量。

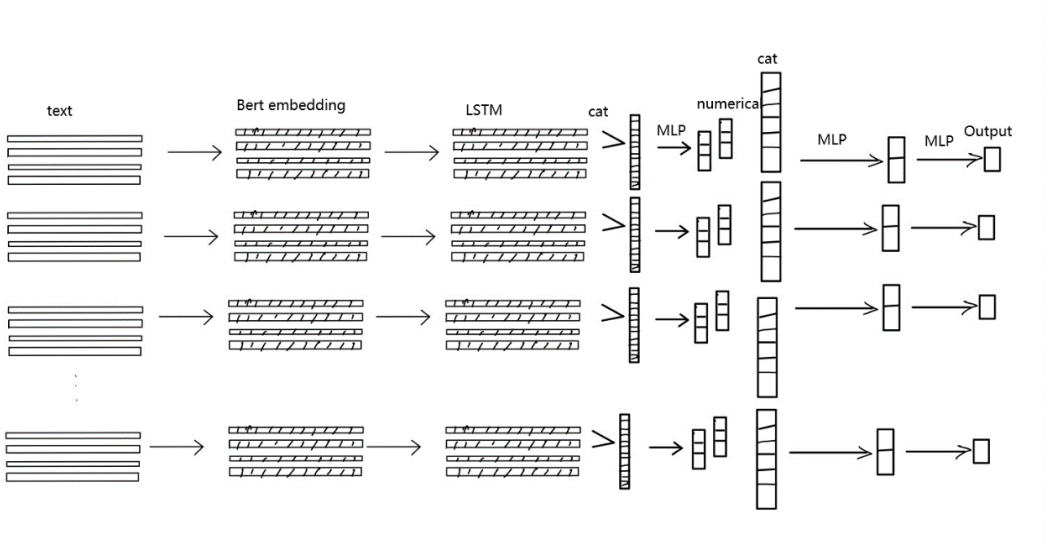
最终我们将得到的连接后的向量送到两层 MLP 中进行训练。最终得到的结果如下：



由上图可知，经过合适的处理之后，在训练的过程中 AUC 的值得到了大幅提升，最高能够超过 89%。在我们提交的时候，AUC 为 91%左右。但是由于神经网络训练过程的不确定性。这里仅展示最新运行的结果。

2.3 第三次提交

模型结构如下图所示：



在提交了两次之后，我们便思考对第二次提交的结果尽心改进。

我们想到第二次提交的不足之处为没有将文本数据所蕴含的信息全部挖掘出来。由于仅仅将每一个公司的最新的一条公告代替全部文本所含的信息量。在这个过程中肯定有往期信息的损失。我们认为可以利用 LSTM 或者 Attention 模型，将往期的信息一同提取出来。

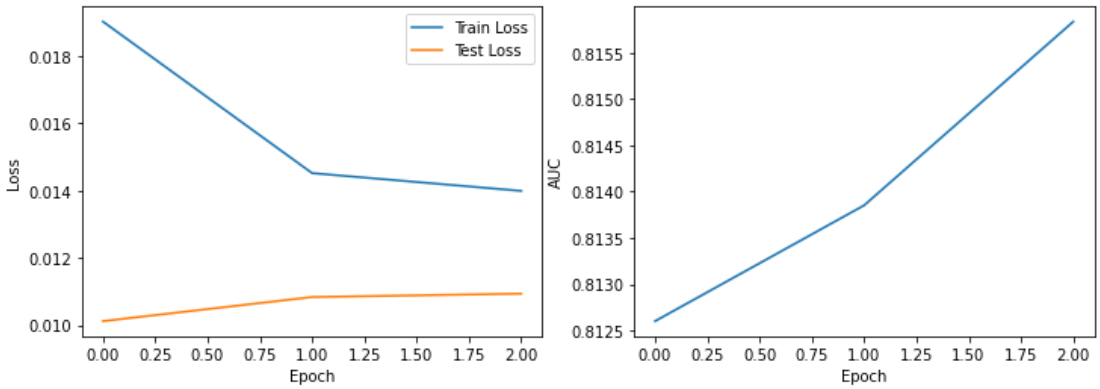
第一步：将全部文本信息进行 Bert 编码，这并是一个简单的过程，却异常耗时间。由于训练文本有 270331 条。这个过程我们花了 12 个小时的时间。从晚上 12 点运行电脑，直至第二天中午才完成编码。同时我们将编码所得的结果存在了 Trainall.pkl 之中，用来方便以后对编码的应用。

对于每个公司公告数目不相等的情况。我们首先采取仅去每个公司前 n 个评论。如果一家公司的公告数目多于 n 个则进行截断，少于 n 个则用进行特殊处理。特殊处理为：一：进行 0 向量扩充，二：进行最后一个向量的直接复制。最后我们发现第二种方法比第一种方法训练出来的模型更优。我们便采用了第二种方法。对于 n 的数目我们也进行了一定程度的探

索。我们利用网格式搜索，发现当 n 为 10 时效果最好。

LSTM 还是 Attention？我们首先用 Attention 模型进行特征的提取，发现效果并不如人意。在同等条件下，LSTM 模型的预测结果更加好。我们猜测是我们采用的单层 Attention 模型不够深或者是评论数目太少，不能体现 Attention 模型对于全局记忆的优越性的。而 LSTM 则更能处理相对而言较少的数据集。最终我们采用的是 LSTM 模型。

最终得到的结果如下：



由图可知，加上 LSTM 后结果反而没有不加的好。在此我们推测是由于其每个信息之间相关连的信息比较少的缘故，在此情况下，利用 LSTM 对其进行处理反而使其原有的语义更加混乱，丧失了 bert 编码所带来的本意。但相对于第一次提交（乱搞）来说仍有进步。

对于在训练集上的结果为 81%。是由于每次训练的结果有很大不确定性所导致的。在最后一次提交时，其 AUC 为 85%，相对于老师测试的结果 0.92660 有较大幅度提升，原因可能是在我们训练的过程中，测试集的数目相对较少，并不能完全反应模型的训练情况。如果测试的数量较大时，训练所得的模型会更好的展现出来。

3.总结

通过此次紧张又刺激的竞赛，我们收获颇丰，从一开始的无从下手，病急乱投医，到最终理解每一步的操作与意图，真正感受到了大数据分析这门课程的真正含义。

在此次实验中，我们首先对问题进行分析，简化问题，抽取其真正意图。我们首先把中文文本进行向量化处理然后在用向量化，再将其与提供的数值型指标一起来进行模型的训练预测。

在此过程中我们尝试了很多模型结构，并采用了提取一部分数据来进行训练预测等方法来加快模型筛选。在中文文本向量化的过程中，我们尝试了 CountVectorizer 和 Bert 预训练模型进行编码两种方式，并最终采用了 Bert 编码。

对于在不等长度的中文文本向量中如何确定最终参与训练的向量的问题上，我们首先采用了只取最新的文本向量参与，再采用每个企业采用前十个文本来进行训练。最终的结果表明采用第一个文本向量作为指标效果比较好。

在第三次实验的过程中我们又对深度神经网络进行了其他变式的探索比如增加 LSTM 层来对文本数据进行处理，使其能从文本数据集中获取到更多相关信息，以及试采用 Self-Attention 模型来提取更多有用信息。我们发现 LSTM 在相同情况下比 Attention 的效果要好一些。我们猜测可能是数据量较小 Attention 模型没能发挥出其优越性，而 LSTM 更加适用于小型数据集的缘故。

最后，通过此次竞赛我们不仅加深了对深度学习模型的了解，也使我们组员之间的友谊大大的加深。感谢队友的通力协作与老师的辛勤付出！

4.参考

1.ChatGPT

2. https://blog.csdn.net/weixin_38278334/article/details/82320307/

3. https://blog.csdn.net/ningyanggege/article/details/104550613/

4. https://zh.d2l.ai/chapter_recurrent-modern/lstm.html

5.https://zh.d2l.ai/chapter_attention-mechanisms/nadaraya-waston.html#equation-eq-attn-pooling

6.https://www.baidu.com/link?url=2D8R1BTZcyHS242RVyHishfhQPn1jIyakyrpJBalPDXTDBVugYyaoAzVpUzlk_w16QrgxjJzXRvXza8n21wIIydQuT_B1cDXbr6OI7VmfdUHJXdoJxGlF2Vu0tlPThVL&wd=&eqid=e3d3e3e90000e57e00000004646f556e

5.附录：核心代码

名称：文本向量化.ipynb
介绍：Bert 编码代码

```
import pandas as pd
import numpy as np
import torch
from transformers import BertTokenizer, BertModel

# 加载预训练的 BERT 模型
tokenizer = BertTokenizer.from_pretrained('bert-base-chinese')
model = BertModel.from_pretrained('bert-base-chinese')
# 对文本数据进行向量化处理
# jishu=1
df = pd.read_csv(r"D:\桌面\训练集竞赛 2\text_train.csv")#270331
df = df[["Company Number","Report Texts"]]
encoded_texts = []
for text in df['Report Texts']:
    # jishu+=1
    # if jishu==10000 or jishu==20000 or jishu==30000 or jishu==40000:
    #     print("又 10000 啦")
    # 对每个文本进行分词，并加上特殊标记
    marked_text = "[CLS] " + text + " [SEP]"
    # 将分词后的文本转换成对应的 ID
    tokenized_text = tokenizer.tokenize(marked_text)
    # 将文本 ID 转换成模型可接受的输入格式
    indexed_tokens = tokenizer.convert_tokens_to_ids(tokenized_text)
    # 将文本 ID 转换成 PyTorch 张量格式
    tokens_tensor = torch.tensor([indexed_tokens])
    # 将文本输入到 BERT 模型中，获取文本向量
    with torch.no_grad():
        outputs = model(tokens_tensor)
        encoded_text = outputs[0][0][0].numpy()
        encoded_texts.append(encoded_text)

# 将向量化的文本数据添加到数据集中
df['Encoded Texts'] = encoded_texts
import pickle
# 将 DataFrame 保存到文件中
with open('Trainall.pkl', 'wb') as f:
    pickle.dump(df, f)
```

名称：第一次提交.ipynb
介绍：网络结构

```
# 定义模型
class MyModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.text_embedding = nn.Embedding(num_embeddings=116,
embedding_dim=100)
        self.text_rnn = nn.LSTM(input_size=100, hidden_size=64, batch_first=True)
        self.linear1 = nn.Linear(in_features=116, out_features=32)
        self.linear2 = nn.Linear(in_features=32, out_features=1)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, text, numerical):
        text = self.text_embedding(text)
        text_output, _ = self.text_rnn(text)
        text_output = text_output[:, -1, :]
        x = torch.cat([text_output, numerical], dim=1)
        x = self.relu(self.linear1(x))
        x = self.sigmoid(self.linear2(x))
        return x
```

名称：第二次提交.ipynb
介绍：网络结构

```
# 定义模型
class MyModel(nn.Module):
    def __init__(self):
        super().__init__()
        # self.linear0 = nn.Linear(in_features=768, out_features=500)
        self.linear1 = nn.Linear(in_features=820, out_features=500)
        self.linear2 = nn.Linear(in_features=500, out_features=100)
        self.linear3 = nn.Linear(in_features=100, out_features=1)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, text, numerical):
        # text=   text.to(torch.float)
        # text =   self.relu(self.linear0(text))
        x = torch.cat([text, numerical], dim=1)
        x=   x.to(torch.float)
        x = self.relu(self.linear1(x))
        x = self.relu(self.linear2(x))
        x = self.sigmoid(self.linear3(x))
        return x
# 将模型移动到 GPU 上
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
model = MyModel().to(device)
# 定义损失函数和优化器
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters())
```

名称：第三次提交.ipynb
介绍：网络结构

```
# 定义模型
class MyModel(nn.Module):
    def __init__(self):
        super().__init__()
        # self.embedding_matrix = nn.Parameter(torch.randn(64, 1))
        #self.text_embedding = nn.Embedding(num_embeddings=116,
embedding_dim=100)
        self.text_rnn = nn.LSTM(input_size=768, hidden_size=52, batch_first=True)
        self.linear0 = nn.Linear(in_features=768*2, out_features=52)
        self.linear1 = nn.Linear(in_features=52+52, out_features=500)
        self.linear2 = nn.Linear(in_features=500, out_features=100)
        self.linear3 = nn.Linear(in_features=100, out_features=1)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, text, numerical):
        # text=text.to(torch.float)
        # text=self.relu(self.linear0(text))
        # numerical = numerical.to(torch.float)
        # text=text.to(torch.float)
        text,_=self.text_rnn(text)
        text = text[:,-1,:]
        # text=torch.cat([text[:,-1,:],text[:,-2,:]],dim=1)
        # text=self.relu(self.linear0(text))
        x = torch.cat([text, numerical], dim=1)
        x=x.to(torch.float)
        x = self.relu(self.linear1(x))
        x = self.relu(self.linear2(x))
        x = self.sigmoid(self.linear3(x))
        return x
```