浙江大学

金融科技导论大作业报告

Finance Tech Project Report

中文题目: 基于自编码降维的信用评分卡

英文题目: Give me Some Credit: Auto-Encoding

学生姓名: ***

学　　号: ***

指导老师: 郑小林

专业类别: 计算机科学与技术

所在学院: 计算机科学与技术学院

2020 年 7 月 29 日

Declare 申明

特在此声明，“基于自编码降维的信用评分卡”项目所提出的自编码降维在风控领域的应用方法、可视化及实验结果对比与分析，均为个人独立工作成果，并没有对任何现有成果进行借鉴。本文旨在提出一种新的特征处理方法，并不强求实验结果能在 *kaggle* 上名列前茅。

模块概览

基础特征工程模块： 数据初步清洗、WOE 编码

自编码降维模块： 利用 Auto-encoder 神经网络进行特征转换与提取

概率分类器模块： 随机森林、LightGBM 等树模型、LR、DNN 等

目录

1　研究背景与意义

2　国内外研究现状与存在问题

3　研究目标与研究内容

4 研究方法与模型思路

4.1 章节概述

违约概率预测模型的好坏很大程度上取决于特征工程。传统特征工程是通过不断手动尝试分箱，WOE 编码，IV 值筛选，特征交叉组合等方法来提取有效特征。本文提出了一种**基于自编码的特征提取方法**，它在传统方法提取到的特征基础上，利用神经网络对特征进行低维空间的非线性映射和进一步压缩提取，最后将得到的全新特征作为分类模型的输入。

经过调查和后续的实验结果分析，该方法确实能够使得树模型的分类准确率有一定程度的提升，并且在当前风控领域并没有使用先例，属于**全新**的模型思路。

在特征工程方面，本章节将介绍 WOE 编码和 IV 值的基本内容，并从逻辑回归 → 深度学习 → 自编码器的推导关系出发，层层递进，介绍创新点的主要灵感来源。在分类模型方面，将简要介绍逻辑回归、随机森林、LightGBM 和全连接深度神经网络分类器。

4.2 WOE 编码和 IV 值

4.2.1 WOE 编码

WOE 的全称是 “Weight of Evidence”，即证据权重。WOE 是对原始自变量的一种编码形式。要对一个变量进行 WOE 编码，需要首先把这个变量进行分组处理（也叫离散化、分箱等）。分组后，对于第 i 组，WOE 的计算公式如下：

$$WOE_i = \ln(\frac{Py_i}{Pn_i}) = \ln(\frac{\frac{\sum y_i}{\sum y_T}}{\frac{\sum n_i}{\sum n_T}}) \tag{1}$$

其中，Py_i 是这个组中响应客户（风险模型中，对应的是违约客户，总之，指的是模型中预测变量取值为 1 的个体）占所有样本中所有响应客户的比例，Pn_i 是这个组中未响应客户占样本中所有未响应客户的比例，$\sum y_i$ 是这个组中响应客户的数量，$\sum n_i$ 是这个组中未响应客户的数量，$\sum y_T$ 是样本中所有响应客户的数量，$\sum n_T$ 是样本中所有未响应客户的数量。

从这个公式中可以体会到，WOE 表示的实际上是 “当前分组中响应客户占所有响应客户的比例” 和 “当前分组中没有响应的客户占所有没有响应的客户的比例” 的差异。对这个公式做一个简单变换，可以得到：

$$WOE_i = \ln(\frac{Py_i}{Pn_i}) = \ln(\frac{\frac{\sum y_i}{\sum y_T}}{\frac{\sum n_i}{\sum n_T}}) = \ln(\frac{\frac{\sum y_i}{\sum n_i}}{\frac{\sum y_T}{\sum n_T}}) \tag{2}$$

变换以后 WOE 表示的是当前这个组中响应的客户和未响应客户的比值与所有样本中该比值的差异。这个差异为两个比值的比值取对数表示。WOE 越大，这种差异越大，这个分组里的样本响应 (用户违约) 的可能性就越大，WOE 越小，差异越小，这个分组里的样本响应 (用户违约) 的可能性就越小。这就是 WOE 编码所表示的意义。

4.2.2 IV 值

IV 的全称是 Information Value，中文意思是信息价值，或者信息量。在用逻辑回归、决策树等模型方法构建分类模型时，经常需要对自变量进行筛选。IV 值依赖 WOE 编码，是一个很好的衡量自变量对目标变量影响程度的指标。

对于每个分组 i，IV_i 的表达式如下：

$$\begin{aligned} IV_i &= (Py_i - Pn_i) \cdot WOE_i \\ &= (Py_i - Pn_i) \cdot \ln(\frac{Py_i}{Pn_i}) \\ &= (\sum \frac{y_i}{y_T} - \sum \frac{n_i}{n_T}) \cdot \ln(\frac{\sum \frac{y_i}{y_T}}{\sum \frac{n_i}{n_T}}) \end{aligned} \tag{3}$$

将该变量各分组的 IV 值相加便可计算得到 $IV = \sum IV_i$

4.3 从逻辑回归到深度学习

4.3.1 简要说明

逻辑回归是一种常见的分类手段，然而它对某些复杂的非线性分类场景却无能为力，深度学习可以看做是多个逻辑回归的组合，通过特征转换和降维达到非线性分类的效果。本小节将从这一观点出发，介绍从逻辑回归到深度学习的推导过程。此外，这一过程也为自编码方法的提出提供了主要灵感。

4.3.2 逻辑回归

假设在二元分类的场景下，C_1 和 C_2 为两个不同的类别，则根据逻辑回归的思想，用参数 (w, b) 组合成 z，并通过 sigmoid 函数缩放到 0-1 范围内，就得到了样本点 x 属于某一类的概率 $P_{w,b}(C_i|x)$

$$\begin{aligned} P_{w,b}(C_1|x) &= \sigma(z) = \frac{1}{1+e^{-z}} \\ z &= w \cdot x + b = \sum_i w_i x_i + b \end{aligned} \tag{4}$$

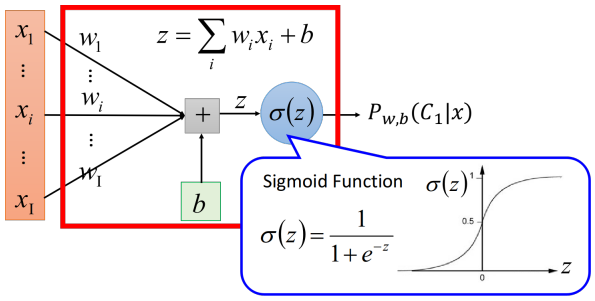

图 1: 逻辑回归示意图

假设 Training data 是从我们定义的 Posterior Probability 中产生的 (后置概率，某种意义上就是概率密度函数)，而 w 和 b 决定了这个 Posterior Probability，此时可以去计算某一组 w 和 b 产生这 N 个样本点的概率，根据极大似然估计的思想，最好的那组参数就是有最大可能性产生当前 N 个样本点分布的 w^* 和 b^*。最终利用极大似然估计法推导出其损失函数，并表示为交叉熵的形式如下：

$$\begin{aligned} w^*, b^* &= \arg\max_{w,b} L(w,b) \\ &= \arg\min_{w,b} (-\ln L(w,b) \\ &= \sum_n -[\hat{y}^n \ln f_{w,b}(x^n) + (1-\hat{y}^n)\ln(1-f_{w,b}(x^n))] \end{aligned} \tag{5}$$

随后计算出参数的偏导，利用梯度下降法更新参数即可：

$$w_i = w_i - \eta \sum_n -(\hat{y}^n - f_{w,b}(x^n))x_i^n \tag{6}$$

相比于 square error，损失函数选择 cross entropy 能够加快参数更新的步伐和速度，而不会导致图 2 中由于损失函数曲面过于平缓而停滞不前的现象

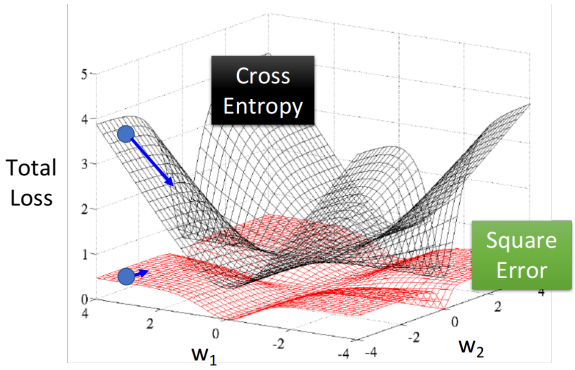

图 2: 交叉熵 vs 方均根

4.3.3 逻辑回归的限制

Logistic Regression 的应用其实有不少的限制，给出图 3 中样本点分布，想要用逻辑回归对它进行分类，其实是做不到的。因为逻辑回归在两个 class 之间的 boundary 就是一条直线，但是在这个平面上无论怎么画直线都不可能把图中的两个 class 分隔开来。

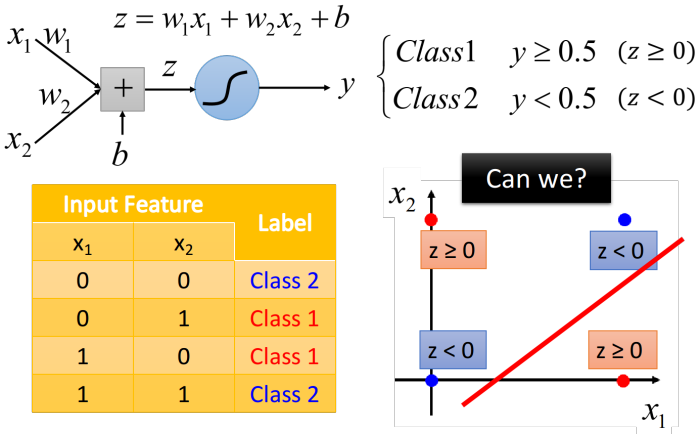

图 3: 逻辑回归的限制

如果坚持要用逻辑回归的话，可以使用 Feature Transformation 把原先的特征分布投影到一个比新的特征空间。

假设定义 x_1' 是原来的点到 $\begin{bmatrix}0\\0\end{bmatrix}$ 之间的距离，x_2' 是原来的点到 $\begin{bmatrix}1\\1\end{bmatrix}$ 之间的距离，重新映射之后如图 4 右侧所示 (红色两个点重合)，此时逻辑回归就可以顺利把它们划分开来。

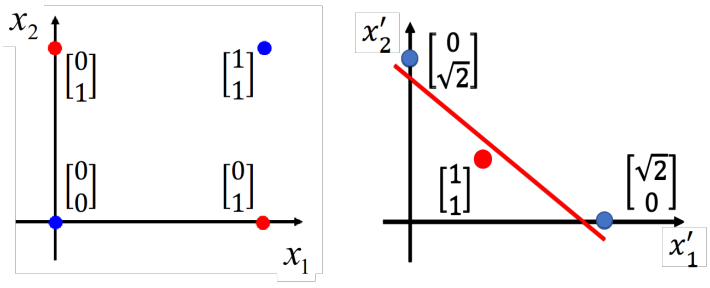

图 4: 特征映射

但实际上，如何做恰当 feature Transformation 本身就是个难题，如果在这上面花费太多时间就得不偿失了。**级联逻辑回归**的提出解决了这个问题，先用 n 个逻辑回归做特征转换，再用 1 个逻辑回归作为新特征的分类器，它们的参数都可以通过梯度下降来得到。

4.3.4 级联逻辑回归到神经网络

在级联逻辑回归中，如果把每一个逻辑回归叫做一个 neuron(神经元)，把这些用于 feature Transformation 的逻辑回归串联起来所形成的网络，叫做 Neural Network(神经网络)，那么级联逻辑回归就是我们所熟知的深度学习神经网络模型，sigmoid 函数则是每个神经元的激活函数。

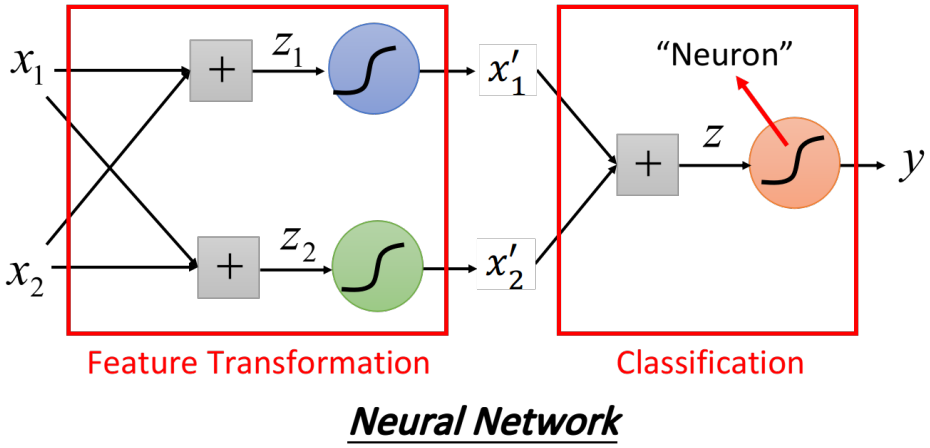

图 5: 级联逻辑回归即神经网络模型

4.4 从神经网络到自编码器

4.4.1 神经网络的本质

从前面的推导可以看出，神经网络的众多隐藏层本质上就是一个个特征提取器，随着网络层数的增加，检测的特征从简单变为复杂，最终通过后续的输出层分类器，得到分类的结果。

如果直接用神经网络作为分类器，也可以得到较好的分类效果。虽然神经网络的特征提取能力是毋庸置疑的，但在整个网络架构中，作为分类器部分的后几层 layer 并不一定能够得到充分的训练，导致分类准确率无法得到进一步的提升。

而本文的创新点在于，用随机森林、LightGBM 等树模型作为神经网络后半部分的分类器。换一种说法就是，并不直接用神经网络作为分类器，而是**把神经网络提取到的特征用于训练其他更专业更高性能的分类模型**。实际操作上就是把神经网络隐藏层的输出接到其他分类模型的输入上，以期得到更好的分类效果。

4.4.2 自编码器提取隐藏层输出

如何获取隐藏层输出呢? 自编码器 (Auto-encoder) 提供了一种很好的思路，它的基本思想是，通过神经网络将原先的高维特征投影到非线性的低维空间上，并通过同样的网络结构重新映射回原先的空间，利用还原前后样本特征之间的差异作为损失函数，直到可以实现稳定特征转换为止。

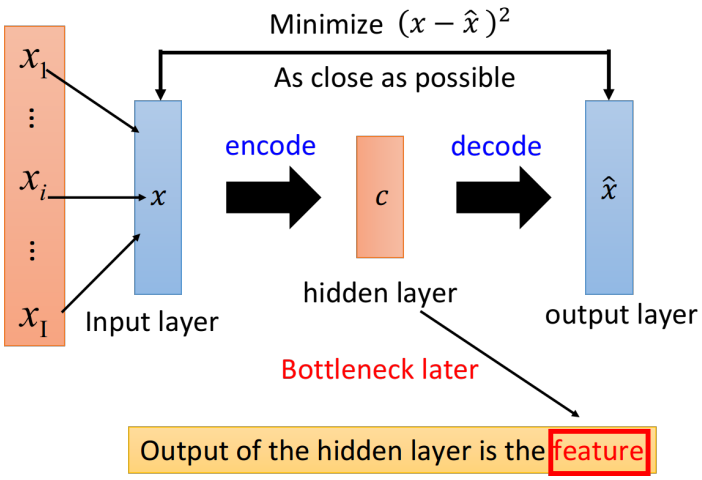

图 6: 自编码器提取特征

从图 6 中可以看出，中间 hidden layer 的输出就是我们希望通过神经网络提取到的特征，接下来用这些特征去训练随机森林、LightGBM 等树模型，或是重新丢给一个神经网络分类器，来获取最终的分类结果。

4.5 分类模型简介

获取到新的特征之后，最终还是需要用专业的分类模型来划分样本点，本文主要使用随机森林、LightGBM、逻辑回归、DNN 分类器来作为进一步的分类模型。这里将简要介绍随机森林和 LightGBM 这两个树模型。

4.5.1 随机森林

随机森林是一种集成算法（Ensemble Learning），它属于 Bagging 类型，通过组合多个弱分类器，最终结果通过投票或取均值，使得整体模型的结果具有较高的精确度和泛化性能。

Bagging 也叫自举汇聚法（bootstrap aggregating），是一种在原始数据集上通过有放回抽样重新选出 k 个新数据集来训练分类器的集成技术。它使用训练出来的分类器的集合来对新样

本进行分类，然后用多数投票或者对输出求均值的方法统计所有分类器的分类结果，结果最高的类别即为最终标签。此类算法可以有效降低 bias，并能够降低 variance。

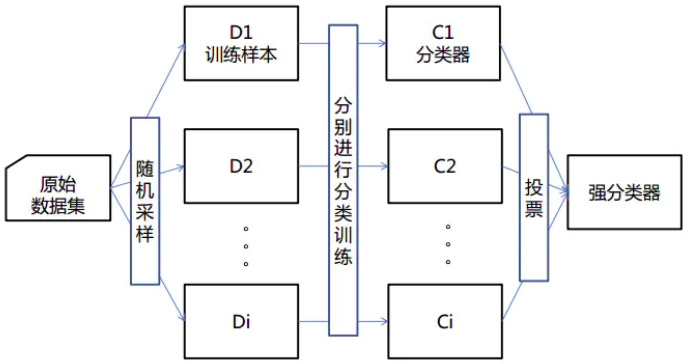

图 7: 随机森林示意图

随机森林的基本特性如下：

- RF 使用了 CART 决策树作为弱学习器
- 在生成每棵树的时候，每个树选取的特征都仅仅是随机选出的少数特征，一般默认取特征总数 m 的开方
- 相对于一般的 Bagging 算法，RF 会选择采集和训练集样本数 N 一样个数的样本
- 由于随机性，对于降低模型的方差很有作用，故随机森林一般不需要额外做剪枝，即可以取得较好的泛化能力和抗过拟合能力

4.5.2 LightGBM

GBDT (Gradient Boosting Decision Tree) 是最常见的分类模型之一，其主要思想是利用弱分类器（决策树）迭代训练以得到最优模型，具有训练效果好、不易过拟合等优点。LightGBM (Light Gradient Boosting Machine) 则是一个实现 GBDT 算法的框架，支持高效率的并行训练，并且具有以下优点：

- 更快的训练速度
- 更低的内存消耗
- 更好的准确率
- 分布式支持，可以快速处理海量数据

GBDT 在每一次迭代的时候，都需要遍历整个训练数据多次，导致训练效率低下，而 LightGBM 则针对这一情况做了优化，变得更轻量。它使用了基于 Histogram 的决策树算法，根据直方图的离散值，遍历寻找最优的分割点。此外，它抛弃了大多数 GBDT 工具使用的按层生长 (level-wise) 的决策树生长策略，而使用了带有深度限制的按叶子生长 (leaf-wise) 算法。

Leaf-wise 是一种更为高效的策略，每次从当前所有叶子中，找到分裂增益最大的一个叶子，然后分裂，如此循环。同 GDBT 使用的 Level-wise 策略相比，在分裂次数相同的情况下，Leaf-wise 可以降低更多的误差，得到更好的精度。Leaf-wise 的缺点是可能会长出比较深的决策树，产生过拟合。因此 LightGBM 在 Leaf-wise 之上增加了一个最大深度的限制，在保证高效率的同时防止过拟合。

LightGBM 还具有支持高效并行的优点。LightGBM 原生支持并行学习，目前支持特征并行和数据并行的两种，并且针对这两种并行方法都做了优化。

5　数据集分析

6 实验与分析

6.1 实验设计说明

为了验证本文提出的基于自编码降维的特征工程对最终分类结果的实际有效性，整个实验设计将分别使用两份数据集，并在相同参数的模型上分别进行验证和对比：

- 使用常见的 WOE 编码、IV 值筛选、特征交叉组合的特征工程处理之后的数据集
- 经过简单处理和 WOE 编码后，再利用神经网络自编码器 (Auto-Encoder) 进行非线性降维和特征提取后的数据集

6.2 实验模型介绍

如下图所示，本文采用的模型基本思路为：先通过初步特征工程和 WOE 编码，得到 13 个基本特征 x；将这 13 个基本特征输入 Auto-encoder 神经网络模型，分别进行 13 维 $\rightarrow$3 维的特征压缩和 3 维 $\rightarrow$13 维的特征还原，还原后的结果记为 $\hat{x}$

将还原前后的 x 和 $\hat{x}$ 之间的差距作为神经网络的损失函数，即 $Loss = \sum_i (x_i - \hat{x}_i)^2$，对损失函数进行梯度下降，训练出一个能够稳定将 13 维特征压缩到 3 维特征的神经网络自编码器。最后，提取出隐藏层的 3 维特征，作为特征工程的最终结果。

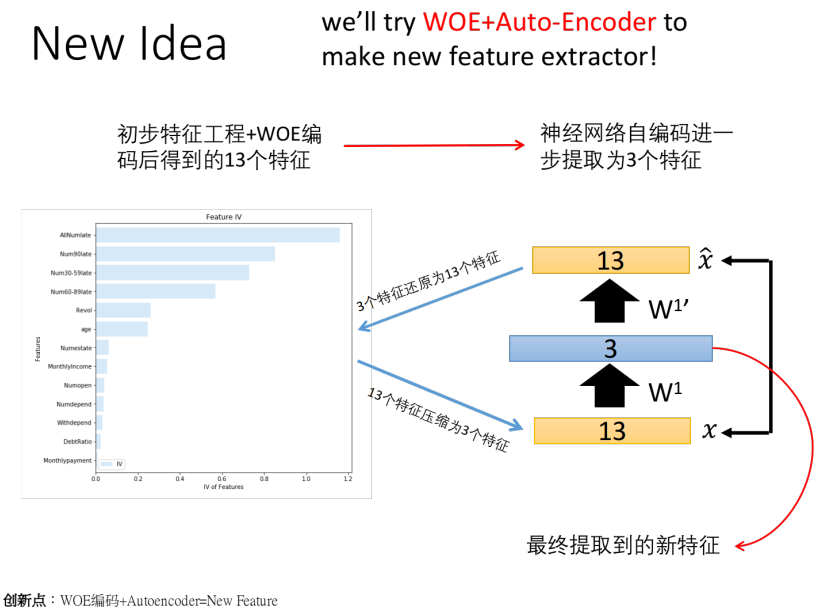

图 8: 自编码模型简介

6.3 模型算法实现

特征处理模型算法实现主要分为两部分，WOE 编码和 Auto-encoder 特征提取。
实现数据分箱、WOE 编码的关键算法如下：

```
def bin_woe(tar, var, n=None, cat=None):
"""
连续自变量分箱,woe,iv变换
tar:target目标变量
var:进行woe,iv转换的自变量
n:分组数量
"""
total_bad = tar.sum()
total_good =tar.count()-total_bad
totalRate = total_good/total_bad

if cat == 's':
    msheet = pd.DataFrame({tar.name:tar,var.name:var,'var_bins':pd.qcut(var, n, duplicates
        ='drop')})
    grouped = msheet.groupby(['var_bins'])
elif (cat == 'd') and (n is None):
    msheet = pd.DataFrame({tar.name:tar,var.name:var})
    grouped = msheet.groupby([var.name])

groupBad = grouped.sum()[tar.name]
groupTotal = grouped.count()[tar.name]
groupGood = groupTotal - groupBad
groupRate = groupGood/groupBad
groupBadRate = groupBad/groupTotal
groupGoodRate = groupGood/groupTotal

woe = np.log(groupRate/totalRate)
iv = np.sum((groupGood/total_good-groupBad/total_bad)*woe)

if cat == 's':
    new_var, cut = pd.qcut(var, n, duplicates='drop',retbins=True, labels=woe.tolist())
elif cat == 'd':
    dictmap = {}
    for x in woe.index:
        dictmap[x] = woe[x]
    new_var, cut = var.map(dictmap), woe.index

return woe.tolist(), iv, cut, new_var
```

PyTorch 实现 Auto-encoder 自编码器的模型如下：
模型训练使用参数: EPOCH=100，lr=0.001，Loss=MSE

```
class AutoEncoder(nn.Module):
def __init__(self):
    super(AutoEncoder, self).__init__()

    self.encoder = nn.Sequential(
        nn.Linear(13, 64),
        nn.Tanh(),
        nn.Linear(64, 32),
        nn.Tanh(),
        nn.Linear(32, 3)
    )

    self.decoder = nn.Sequential(
        nn.Linear(3, 32),
        nn.Tanh(),
        nn.Linear(32, 64),
        nn.Tanh(),
        nn.Linear(64, 13)
    )

def forward(self, x):
    encode = self.encoder(x)
    decode = self.decoder(encode)
    return encode, decode
```

6.4 自编码结果可视化分析

为了更直观地感受基于自编码降维的特征工程的效果，这里将降维后的三个维度可视化在三维空间上。下图是 5 万个样本点根据降维后的三个坐标绘制出来的图像，展示了两个方向上样本点的分布。

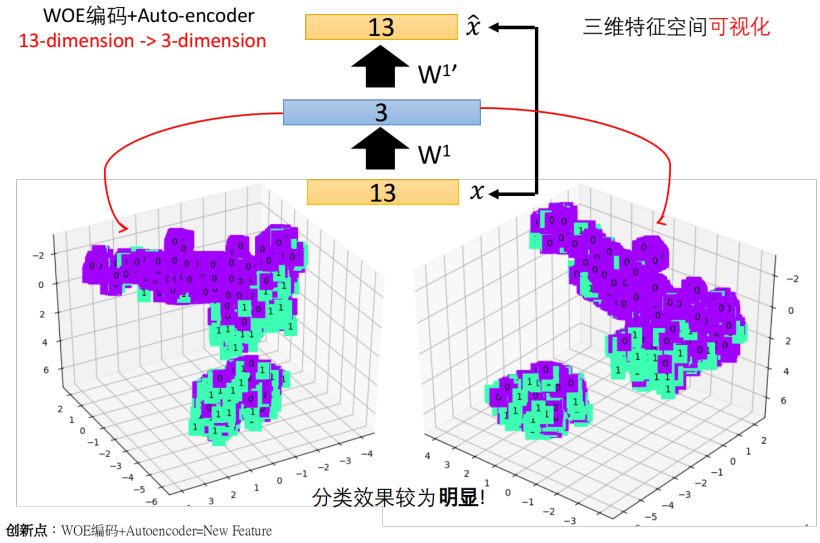

图 9: 自编码特征结果可视化

可以清楚地看到，通过该模型获取到的 3-dimension feature 具备较好的分类能力，能够使两个不同类别的样本点较为明显地分布在空间的两块不同区域。该实验结果表明，本文提出的方法确实具备一定的可行性，能够在一定程度上提高分类效果。

除了定性分析之外，接下来将定量对比分析“是否使用自编码提取特征”在相同分类模型上所表现出来的 AUC 值情况，来进一步证明该方法的正确性。

6.5 不同分类器上的对比

7 结论与展望