神经网络编码分类变量—categorical_embedder

前言

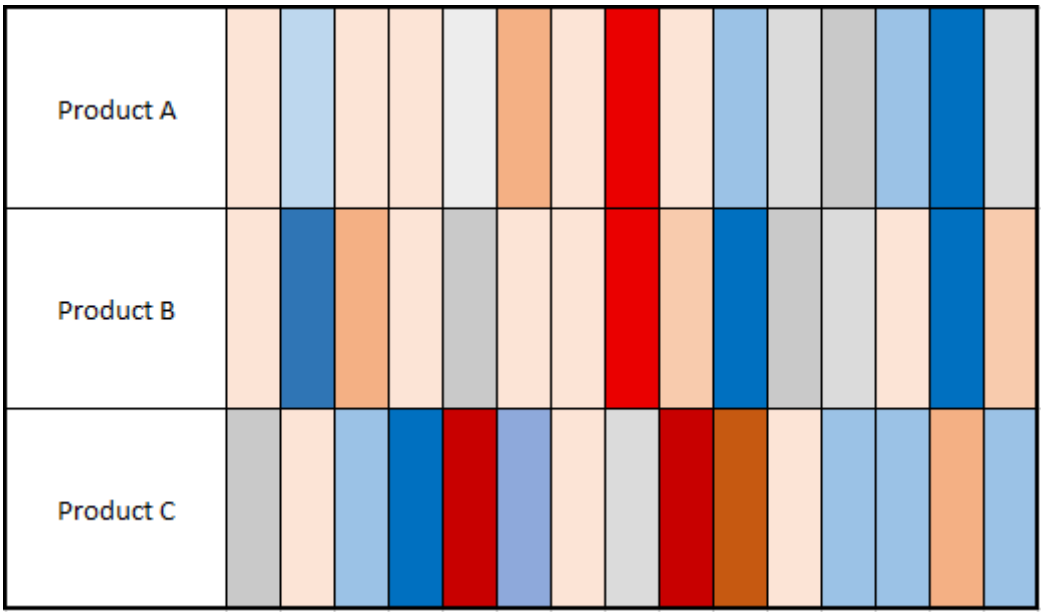
> 上一节我们学习了无监督降维方法**PCA**，在学习**PCA**的输入数据时我们了解一些数据处理的方法，其中提到了非结构化数据如何转化成结构化数据的方法序号编码，独热编码，二进制编码以及**encoding**编码。本节将讲述**Categorical Embedder**方法，如何通过神经网络编码分类变量将非结构化指标映射为浮点数张量，进而满足神经网络输入需求。

1.神经网络的数据预处理

机器学习模型离不开数据的预处理，预处理对于构建网络模型同样非常重要往往能决定训练结果，对于不同的数据集，预处理方法都会有或多或少的局限性和特殊性，特别是神经网络输入数据仅支持浮点数张量。无论处理什么数据（声音，图像还是文本），都需要将其转化为张量，然后再提供给神经网络模型。当我们遇到比如这些数据可以通过序号编码（**Ordinal Encoding**）、独热编码（**One-hot Encoding**）、二进制编码 （**Binary Encoding**）等方法将其转化为数字。接下来介绍一种更为先进的神经网络编码方法。

2.什么是embedding

embedding是将离散变量转化为连续向量表示的一种方式。在**Google**官方教程表述**embedding**使大型输入（例如表示单词的稀疏向量）进行机器学习变得更加容易。它可以帮助我们研究非结构化数据内容时，将这些离散变量转换为数字更有助于模型训练。比如说某一公司有三个产品，平均每个产品又五万条评论，语料库中唯一的词总数为**100**万。我们将得到一个形状为（**150K，1M**）的矩阵。对任何模型来说，这种输入非常大也非常稀疏。我们假设将维度减少到15（每个产品的15位ID），取每个产品嵌入的平均值，然后根据数值对他们进行上色，将得到如下图；

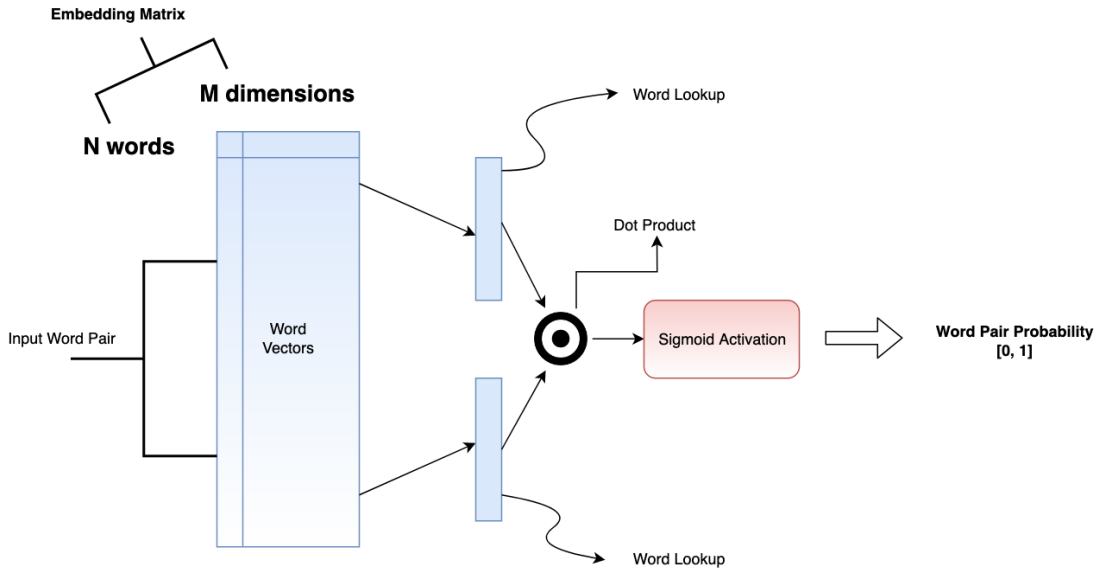


嵌入意味着用一组固定的数字来表示一个类别，我们可以通过（3，15）的矩阵来表示每个产品之间的相似程度。更加可视化也降低复杂度。

每个类别都映射到一个不同的向量，并且在训练神经网络时可以调整或学习向量的属性。向量空间提供类别的投影，从而使得那些接近或相关的类别自然地聚在一起。为了学习嵌入，我们创建了一个使用这些嵌入作为特征并与其他特征交互以学习上述任务。此时需要引用一个概念：词向量。

2.1 词向量

词向量是一种语言的每个词的嵌入向量。词向量的整个想法是，在句子中出现更近的词通常彼此更接近。嵌入是**n** 维向量。每个维度捕获每个单词的某些属性/属性，因此更接近属性，更接近单词。为了学习词向量，我们创建了一组出现在一个小词窗口（比如 5 个词）内的词对作为正例，并创建一组没有出现在该窗口中的词对作为负例。



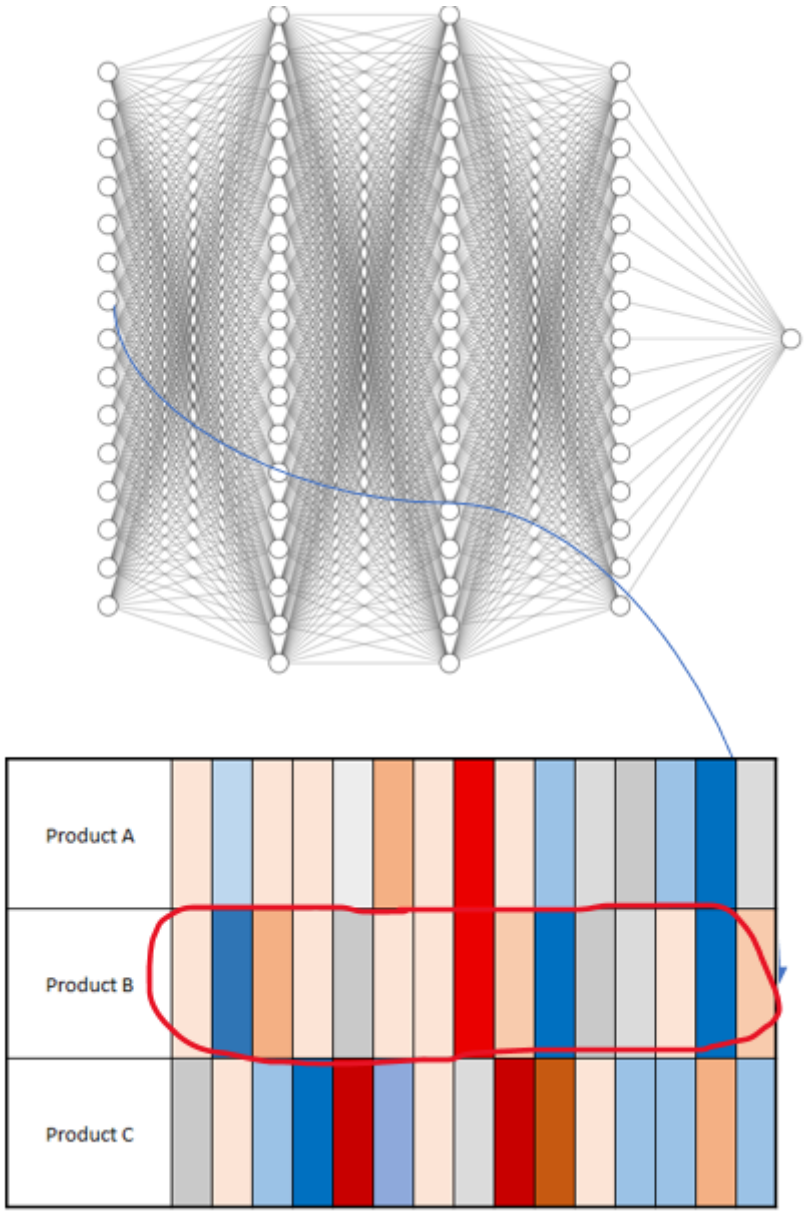
当我们在足够大的数据集上训练上述神经网络时，模型会学习预测两个词是否相关。然而，这个模型的**副产品**是嵌入矩阵，它是词汇表中每个单词的信息丰富的向量表示。

2.2传统方法

- 序号编码（**Ordinal Encoding**）序号编码通常用于处理类别间具有大小关系的数据。
- 独热编码（**One-hot Encoding**）使用稀疏向量来节省空间，配合特征选择来降低维度
- 二进制编码 （**Binary Encoding**）二进制编码主要分为两步，先用序号编码给每个类别赋予一个类别ID，然后 将类别ID对应的二进制编码作为结果。

2.3 categorical_embedder工作原理

首先，每个分类变量类别都映射一个**n**维向量。这种映射是在标准的监督训练过程中由神经网络学习的。如果我们想使用上述15维ID作为特征，那么我们将以监督方式训练神经网络，获取每个特征的向量并生成如下所示的3*15的矩阵。如下图所示（太多神经节点不好表示，下图仅参考）



接下来，我们将使用数据中对应的向量来替换每个类别。这样做的优点在于我们限制了每个类别所需的列的数量。这在列具有高基数是非常有用（基数是指对集合元素数量的度量）。从神经网络获得的生成嵌入揭示了分类变量的内在属性。这意味着相似的类别将具有相似的嵌入。

2.4 学习嵌入矩阵

嵌入矩阵是浮点数**N*M**矩阵。这里**N**是唯一类别的数量，**M**是嵌入维度。我们决定**M**的值。通常将**M**设置为等于**N**的平方根，然后根据需要增加或减少。实际上，嵌入矩阵是向量的查找表。嵌入矩阵的每一行都是一个唯一类别的向量。

对于公司而言，它有三个产品，每个产品的评价为五万个唯一值，要构建分类嵌入，我们需要解决有意义的任务深度学习模型，在上述任务中使用嵌入矩阵来表示分类变量。我们用15维变量来预测公司的产品关联。可以通过颜色的区分来分析哪个产品具有相关性，当然这个属于推荐系统的一个分析思路。更多的是将样本的属性分门别类构建对应的**embedding**矩阵，通过这些**Label_embedding**矩阵，通过**Faltten**层，输入神经网络中进行训练

3 基于Python的categorical_embedder

3.1 神经网络编码代码复现

```
pip install categorical_embedder
```

注意：这个库要求**tensorflow**的版本在2.1以下，高于此版本会出现未知错误。

在这个**categorical_embedder**包含一些重要的函数定义，我们仔细描述其含义。

- **ce.get_embedding_info(data,categorical_variables=None)**：这个函数的目的是识别数据中所有的分类变量，确定其嵌入大小。分类变量的嵌入大小由至少50个或一半的数量决定。唯一值，即嵌入列大小 = **Min（50**，#该列中的唯一值）。我们可以在**categorical_variables**参数中传递一个明确的分类变量列表。如果没有，这个函数会自动接受所有数据类型为对象的变量。

```
def get_embedding_info(data, categorical_variables=None):
    '''
    this function identifies categorical variables and its embedding size
    :data: input data [dataframe]
    :categorical_variables: list of categorical_variables [default: None]
    if None, it automatically takes the variables with data type 'object'
    embedding size of categorical variables are determined by minimum of 50 or
half of the no. of its unique values.
    i.e. embedding size of a column  = Min(50, # unique values of that column)
    '''
    if categorical_variables is None:
        categorical_variables = data.select_dtypes(include='object').columns

    return {col:(data[col].nunique(),min(50,(data[col].nunique()+ 1) //2)) for
col in categorical_variables}
```

- **ce.get_label_encoded_data(data, categorical_variables=None)**：此函数标签使用 **sklearn.preprocessing.LabelEncoder** 函数对所有分类变量进行编码（整数编码），并返回标签的数据帧进行训练。**Keras/TensorFlow** 或任何其他深度学习库都希望数据采用这种格式。

```
def get_label_encoded_data(data, categorical_variables=None):
    '''
    this function label encodes all the categorical variables using
sklearn.preprocessing.labelencoder
    and returns a label encoded dataframe for training
    :data: input data [dataframe]
    :categorical_variables: list of categorical_variables [Default: None]
    if None, it automatically takes the variables with data type 'object'
    '''
    encoders = {}

    df = data.copy()

    if categorical_variables is None:
        categorical_variables = [col for col in df.columns if df[col].dtype ==
'object']

    for var in categorical_variables:
        #print(var)
        encoders[var] = __LabelEncoder__()
        df.loc[:, var] = encoders[var].fit_transform(df[var])
```

```
    return df, encoders
```

- **ce.get_embeddings(X_train, y_train, categorical_embedding_info=embedding_info, is_classification=True, epochs=100,batch_size=256)**：这个函数训练一个浅层神经网络并返回分类变量的嵌入。在底层是一个二层神经网络架构，具有1000*500的神经元，带有ReLU激活。它需要四个必须输出**X_train**，**y_train**，**categorical_embedding_info**：**get_embedding_info**函数的输出和**is_classification:True**用于分类任务；False用于回归任务。

  对于分类：**loss = 'binary_crossentropy'; metrics = 'accuracy'**对于回归**loss = 'mean_squared_error'; metrics = 'r2'**

```
def get_embeddings(X_train, y_train, categorical_embedding_info,
is_classification, epochs=100, batch_size=256):
    '''
    this function trains a shallow neural networks and returns embeddings of
categorical variables
    :X_train: training data [dataframe]
    :y_train: target variable
    :categorical_embedding_info: output of get_embedding_info function
[dictionary of categorical variable and it's embedding size]
    :is_classification: True for classification tasks; False for regression
tasks
    :epochs: num of epochs to train [default:100]
    :batch_size: batch size to train [default:256]
    It is a 2 layer neural network architecture with 1000 and 500 neurons with
'ReLU' activation
    for classification: loss = 'binary_crossentropy'; metrics = 'accuracy'
    for regression: loss = 'mean_squared_error'; metrics = 'r2'
    '''

    numerical_variables = [x for x in X_train.columns if x not in
list(categorical_embedding_info.keys())]

    inputs = []
    flatten_layers = []

    for var, sz in categorical_embedding_info.items():
        input_c = Input(shape=(1,), dtype='int32')
        embed_c = Embedding(*sz, input_length=1)(input_c)
        flatten_c = Flatten()(embed_c)
        inputs.append(input_c)
        flatten_layers.append(flatten_c)
        #print(inputs)

    input_num = Input(shape=(len(numerical_variables),), dtype='float32')
    flatten_layers.append(input_num)
    inputs.append(input_num)

    flatten = concatenate(flatten_layers, axis=-1)

    fc1 = Dense(1000, kernel_initializer='normal')(flatten)
    fc1 = Activation('relu')(fc1)



    fc2 = Dense(500, kernel_initializer='normal')(fc1)
```

```
    fc2 = Activation('relu')(fc2)


    if is_classification:
        output = Dense(1, activation='sigmoid')(fc2)

    else:
        output = Dense(1, kernel_initializer='normal')(fc2)


    nnet = Model(inputs=inputs, outputs=output)

    x_inputs = []
    for col in categorical_embedding_info.keys():
        x_inputs.append(X_train[col].values)

    x_inputs.append(X_train[numerical_variables].values)

    if is_classification:
        loss = 'binary_crossentropy'
        metrics='accuracy'
    else:
        loss = 'mean_squared_error'
        metrics=r2



    nnet.compile(loss=loss, optimizer='adam', metrics=[metrics])
    nnet.fit(x_inputs, y_train.values, batch_size=batch_size, epochs=epochs,
validation_split=0.2, callbacks=[TQDMNotebookCallback()], verbose=0)

    embs = list(map(lambda x: x.get_weights()[0], [x for x in nnet.layers if
'Embedding' in str(x)]))
    embeddings = {var: emb for var, emb in
zip(categorical_embedding_info.keys(), embs)}
    return
```

注意这些代码为该库源代码，只需了解即可。

3.2 案例分析—二手车价格预测

基于给定的二手车交易样本数据作为训练，验证，测试样本。构建二手车零售交易价格预测模型。首先要查看整个数据的结构，指标的意义。通过观察数据我们可以发现有一些指标是日期，未知特征等非结构化数据。如果我们想构建机器学习模型(神经网络模型)。

我们需要先使用**categorical_embedding**方法将非结构数据进行转化，再进行处理。首先导入相关库；

```
import pandas as pd
import numpy as np
import categorical_embedder as ce
from sklearn.model_selection import train_test_split
from keras import models
from keras import layers
import matplotlib.pyplot as plt
import csv
```

观察整个数据集的形状并输出前五行观察数据集的内容，部分展示如下；

```
train_data = pd.read_csv('train_estimate.csv')
train_data.shape
train_data.head()
```

(30000, 36)

	carid	tradeTime	brand	serial	model	mileage	color	cityId	carCode	transferCount	...	anonymousFeature7	anonymousFeature8	anonymousFeature9	a
0	1	2021/6/28	1	1	1	4.01	1	1	1	0	...	0	1	5	
1	2	2021/6/25	2	2	2	8.60	1	2	1	0	...	0	2	4	
2	5	2021/6/19	5	5	5	15.56	1	2	3	0	...	0	0	0	
3	6	2021/6/29	6	6	6	6.04	1	3	1	3	...	2018/8/18	2	5	
4	7	2021/6/30	7	7	7	5.70	4	1	2	2	...	2020/9/20	1	5	

5 rows × 36 columns

我们可以看到'trade Time'，'anonymousFeature12'等指标是非结构化的，我们先将排列顺序的第一列和价格最后一列删掉，为构建输入数据准备。确定分类变量，将非结构化数据展示，使用ce.get_embedding_info函数获取非结构化数据的变量。

```
X = train_data.drop(['carid', 'price'], axis=1)
y = train_data['carid']
embedding_info = ce.get_embedding_info(X)
embedding_info
```

结果如下所示；

```
{'tradeTime': (553, 50),
 'registerDate': (200, 50),
 'licenseDate': (3690, 50),
 'anonymousFeature7': (1955, 50),
 'anonymousFeature11': (7, 4),
 'anonymousFeature12': (2175, 50),
 'anonymousFeature15': (1981, 50)}
```

我们可以看到需要编码训练的数据部分，接下来需要使用**ce.get_label_encoded_data**：此函数标签使用**sklearn.preprocessing.LabelEncoder**对所以分类变量进行编码（整数编码），并返回标签编码的数据帧进行训练。

```
# 分割数据
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y)

# ce.get_embeddings trains NN, extracts embeddings and return a dictionary
containing the embeddings
embeddings = ce.get_embeddings(X_train, y_train,
categorical_embedding_info=embedding_info, is_classification=True,epochs=100,
batch_size=256)
embeddings
```

Training: 100% 100/100 [00:48<00:00, 2.12it/s]

```
{'tradeTime': array([[-2.8130016, -2.8222551,  2.8077073, ..., -2.8426743,  2.8399181,
        -2.8477864],
       [-4.703432 , -4.681968 ,  4.6859136, ..., -4.7571015,  4.750078 ,
        -4.7366977],
       [-3.734909 , -3.7263925,  3.6819875, ..., -3.7349012,  3.725073 ,
        -3.711513 ],
       ...,
       [-6.374183 , -6.417619 ,  6.3677726, ..., -6.378419 ,  6.403563 ,
        -6.395538 ],
       [-6.5962777, -6.596868 ,  6.5809946, ..., -6.6029735,  6.573271 ,
        -6.642335 ],
       [-6.3416476, -6.27181  ,  6.3436418, ..., -6.3527145,  6.3568826,
        -6.3386793]], dtype=float32),
 'registerDate': array([[ 1.2767382 ,  1.3336782 , -1.3366479 , ...,  1.3363225 ,
         1.3373188 , -1.3079093 ],
       [ 1.3295187 ,  1.298884  , -1.3462937 , ...,  1.3329318 ,
         1.2711751 , -1.2770027 ],
       [-0.01741026,  0.03139189,  0.02903868, ..., -0.02187575,
         0.02338437,  0.04476347],
       ...,
       [ 2.3339007 ,  2.3194146 , -2.3259344 , ...,  2.3301706 ,
         2.3796325 , -2.3016748 ],
       [ 1.2930498 ,  1.3295791 , -1.268576  , ...,  1.2504487 ,
         1.3119986 , -1.2906067 ],
       [ 1.2668233 ,  1.2139847 , -1.2492843 , ...,  1.2969378 ,
         1.2811793 , -1.2406701 ]], dtype=float32),
 'licenseDate': array([[ 1.3220539e+00, -1.3019379e+00, -1.3167660e+00, ...,
        -1.3120569e+00, -1.3030366e+00, -1.2817383e+00],
       [ 2.0266209e-02, -2.4142696e-02,  9.5603615e-04, ...,
         5.8592446e-03,  1.0971725e-02, -1.6427111e-02],
       [ 1.2809091e+00, -1.3274233e+00, -1.3312523e+00, ...,
```

如上图所示我们通过一个2层神经网络架构1000*500个神经元，并带有"ReLU激活。我们可以查看一下编码后数据的变化情况，如下图所示；

```
train_data_encoded['tradeTime']
```

	tradeTime_embedding_0	tradeTime_embedding_1	tradeTime_embedding_2	tradeTime_embedding_3	tradeTime_embedding_4	tradeTime_embedding_5
2020/1/1	-2.813002	-2.822255	2.807707	-2.801093	-2.846982	-2.786235
2020/1/10	-4.703432	-4.681968	4.685914	-4.701083	-4.778468	-4.742292
2020/1/11	-3.734909	-3.726393	3.681988	-3.705204	-3.758372	-3.695650
2020/1/12	-3.641496	-3.659539	3.642892	-3.577733	-3.617266	-3.629308
2020/1/13	-2.947183	-2.928936	2.972247	-3.001599	-2.943012	-2.966279
...	...	...	...	...	...	...
2021/7/5	-5.366847	-5.344046	5.335837	-5.407475	-5.331368	-5.415002
2021/7/6	-7.036557	-6.945299	6.941107	-7.005300	-7.027971	-6.957109
2021/7/7	-6.374183	-6.417619	6.367773	-6.363153	-6.385043	-6.427567
2021/7/8	-6.596278	-6.596868	6.580995	-6.598681	-6.572830	-6.634850
2021/7/9	-6.341648	-6.271810	6.343642	-6.327192	-6.283332	-6.279900

553 rows × 50 columns

接下来需要将训练好的数据编码从字典转换到数据集中；

```
data = ce.fit_transform(X, embeddings=embeddings, encoders=encoders,
drop_categorical_vars=True)
data.head()
```

100% 7/7 [00:00<00:00, 54.42it/s]

	brand	serial	model	mileage	color	cityId	carCode	transferCount	seatings	country	...	anonymousFeature15_embedding_40	anonymousFeature15_embe
0	1	1	1	4.01	1	1	1	0	5	779413	...	-11.914192	-
1	2	2	2	8.60	1	2	1	0	5	779415	...	-11.914192	-
2	5	5	5	15.56	1	2	3	0	5	0	...	-11.914192	-
3	6	6	6	6.04	1	3	1	3	5	779413	...	-11.914192	-
4	7	7	7	5.70	4	1	2	2	5	779415	...	-11.914192	-

5 rows × 331 columns

此时使用神经网络编码的步骤已经全部结束，从数据头部可以很清晰的看到数据的结构已经全部转换为浮点数张量，已经满足了机器学习模型的输入需求，为了进一步提高模型的效果，还需要进行标准化等方法进行处理。

4 总结

机器学习模型对数字变量比较敏感，对于非结构化数据十分迟钝。传统的分类变量方法一定程度上限制了算法的能力。在适当的条件下，我们可以学习全新的嵌入来提高模型性能。分类嵌入通常表现很好，有助于模型更好的泛化。

5.参考文献

1. Categorical Embedder: Encoding Categorical Variables via Neural Networks
2. A Deep-Learned Embedding Technique for Categorical Features Encoding
3. Deep embedding's for categorical variables (Cat2Vec)

推荐阅读

- 微分算子法
- 使用PyTorch构建神经网络模型进行手写识别
- 使用PyTorch构建神经网络模型以及反向传播计算
- 如何优化模型参数，集成模型
- TORCHVISION目标检测微调教程
- 神经网络开发食谱
- 主成分分析（PCA）方法步骤以及代码详解