



湖南大學

HUNAN UNIVERSITY

人工智能实验七报告
神经网络

目录

一、 实验目的	3
二、 实验描述	3
三、 实验及结果分析	4
(1) 开发语言及运行环境;	4
(2) 实验的具体步骤;	4
① 交通运量预测	4
② 语音性别识别	7
(3) 根据实验数据集,按实验要求给出相应的结果(截图)并对 实验结果进行简要分析。	10
① 交通运量预测	10
② 语音性别识别	11
四、 心得	13
五、 程序文件名清单	13
六、 附录	13

一、实验目的

建议使用python编程实现，并在Mindspore框架下实现。

二、实验描述

神经网络是一种模拟人脑神经元网络结构和功能的计算模型，用于解决分类、回归和特征学习等多种机器学习问题。

基本思想：

1. 核心思想：神经网络通过调整网络中的权重和偏置来最小化预测输出和真实标签之间的差异。它由多层节点组成，包括输入层、一个或多个隐藏层以及输出层。每个节点（神经元）通过激活函数处理输入信号，并产生输出信号。

2. 网络结构：神经网络的结构由层数和每层的节点数决定。输入层节点数与特征数量相对应，输出层节点数与类别数量相对应（对于分类问题）。隐藏层的数量和节点数可以根据问题复杂度进行调整。

3. 前向传播：在前向传播过程中，输入数据通过网络的每一层，每层的输出成为下一层的输入，直到最后一层（输出层）产生预测结果。

4. 损失函数：损失函数（如交叉熵损失或均方误差）用于衡量模型预测与真实标签之间的差异。在分类问题中，常用交叉熵损失。

5. 反向传播：反向传播算法用于计算损失函数关于网络权重的梯度。这些梯度指示了如何调整权重以减少损失。

6. 权重更新：使用梯度下降或其变体（如随机梯度下降、Adam等）来更新网络权重。这个过程涉及到学习率，它控制了权重更新的步长。

7. 训练过程：神经网络的训练是一个迭代过程，包括前向传播、计算损失、反向传播和权重更新。这个过程重复进行，直到满足停止条件，如达到最大迭代次数或损失函数收敛。

8. 特点：神经网络能够处理非线性问题，可以自动学习特征，并且能够处理高维数据。然而，它们也容易过拟合，需要正则化技术（如dropout、L2正则化等）来提高泛化能力。

数据集处理：

trafficPredictor中使用pd.read_excel读取数据，并对数据进行处理并进行标准化，然后再训练模型。（对训练数据和预测数据都进行标准化）

和第一个问题类似的，VoiceGenderClassifier类用于预处理，使用pd.read_csv处理数据、进行标准化，并转成tensor向量。

三、实验及结果分析

（1）开发语言及运行环境；

与实验一相同，不再赘述。

（2）实验的具体步骤；

① 交通运量预测

题目：

公路运量主要包括公路客运量和公路货运量两个方面。据研究，某地区的公路运量主要与该地区的人口数量、机动车数量和公路面积有关。list1中给出了某地区20年的公路运量相关数据。

根据相关部门数据，该地区2010年和2011年的人口数量分别为73.39和75.55万人，机动车数量分别为3.9635和4.0975万辆，公路面积将分别为0.9880和1.0268万平方千米。

list2中是2010年和2011的人口数量、机动车数量、公路面积。

（我给数据集改了名字）

数据集：

表1中给出了某地区20年的公路运量相关数据。

年份	人口数量	机动车数量	公路面积	公路客运量	公路货运量
1991	22.44	0.75	0.11	6217	1379
1992	25.37	0.85	0.11	7730	1385
1993	27.13	0.9	0.14	9145	1399

步骤与要求：

（1）请利用BP神经网络预测该地区2010年和2011年的公路客运量和公路货运量。

BPNeuralNetwork类实现了一个具有三个全连接层的神经网络，使用了ReLU激活函数，添加了BatchNorm层作批归一化，改善性能。

```

15 class BPNeuralNetwork(nn.Cell):
16     """BP神经网络模型"""
17
18     def __init__(self):
19         super(BPNeuralNetwork, self).__init__()
20         # 使用三层神经网络结构
21         self.layer1 = nn.Dense(3, 16)
22         self.layer2 = nn.Dense(16, 8)
23         self.layer3 = nn.Dense(8, 1)
24         # 使用ReLU激活函数
25         self.relu = nn.ReLU()
26         # 添加BatchNorm层
27         self.bn1 = nn.BatchNorm1d(16)
28         self.bn2 = nn.BatchNorm1d(8)
29
30     def construct(self, x):
31         x = self.bn1(self.relu(self.layer1(x)))
32         x = self.bn2(self.relu(self.layer2(x)))
33         x = self.layer3(x)
34         return x

```

NetWithLoss类将神经网络模型和损失函数结合起来，计算损失，在训练过程中输出。

train_models函数对两个模型分别进行训练。

```

112 def train_models(self):
113     """训练BP神经网络和线性回归模型"""
114     # 训练BP神经网络模型
115     print("\n训练BP神经网络模型...")
116
117     # 客运量预测模型
118     self.bp_model_passengers = BPNeuralNetwork()
119     print("训练客运量预测模型...")
120     self.bp_model_passengers = self.train_single_model(
121         self.bp_model_passengers,
122         self.X_scaled_ms,
123         self.y1_scaled_ms
124     )
125
126     # 货运量预测模型
127     self.bp_model_freight = BPNeuralNetwork()
128     print("\n训练货运量预测模型...")
129     self.bp_model_freight = self.train_single_model(
130         self.bp_model_freight,
131         self.X_scaled_ms,
132         self.y2_scaled_ms
133     )
134
135     # BP预测结果
136     self.bp_pred_passengers = self.scaler_y1.inverse_transform(
137         self.bp_model_passengers(self.X_pred_scaled_ms).asnumpy().reshape(-1, 1))
138     self.bp_pred_freight = self.scaler_y2.inverse_transform(
139         self.bp_model_freight(self.X_pred_scaled_ms).asnumpy().reshape(-1, 1))
140
141     # 训练线性回归模型
142     print("\n训练线性回归模型...")
143     self.lr_passengers = LinearRegression()
144     self.lr_freight = LinearRegression()
145
146     self.lr_passengers.fit(self.X_scaled, self.y1_scaled)
147     self.lr_freight.fit(self.X_scaled, self.y2_scaled)
148
149     # 线性回归预测结果
150     self.lr_pred_passengers = self.scaler_y1.inverse_transform(
151         self.lr_passengers.predict(self.X_pred_scaled).reshape(-1, 1))
152     self.lr_pred_freight = self.scaler_y2.inverse_transform(
153         self.lr_freight.predict(self.X_pred_scaled).reshape(-1, 1))

```

plot_results绘出结果，print_predictions打印预测结果。

(2) 请利用其他方法预测该地区2010年和2011年的公路客运量和公路货运量，并比较这两种方法的优缺点。

使用的是线性回归的模型。

在train_models函数中。

```

143     self.lr_passengers = LinearRegression()
144     self.lr_freight = LinearRegression()
145
146     self.lr_passengers.fit(self.X_scaled, self.y1_scaled)
147     self.lr_freight.fit(self.X_scaled, self.y2_scaled)
148
149     # 线性回归预测结果
150     self.lr_pred_passengers = self.scaler_y1.inverse_transform(
151         self.lr_passengers.predict(self.X_pred_scaled).reshape(-1, 1))
152     self.lr_pred_freight = self.scaler_y2.inverse_transform(
153         self.lr_freight.predict(self.X_pred_scaled).reshape(-1, 1))

```

优缺点:

1. 模型复杂度和灵活性:

线性回归模型相对简单，易于理解和实现。它假设自变量与因变量之间存在线性关系，这使得模型具有很好的解释性。然而，这种简单性也限制了它处理复杂非线性关系的能力。

BP神经网络，作为一种非线性模型，能够学习和存储大量的输入输出映射关系，对于任何连续函数都可以进行任意精度的逼近。这使得**BP神经网络**具有很强的容许性和灵活性，可以适应各种复杂的输入输出关系。

2. 训练和计算效率:

线性回归模型的训练计算效率高，尤其是在大规模数据集上，因为它通常涉及到简单的数学运算，如矩阵乘法和求逆。

BP神经网络的训练过程可能需要更多的计算资源和时间，特别是当网络结构复杂或者数据集较大时。这是因为它涉及到权重的多次迭代更新，以及反向传播算法的计算。

3. 对异常值和共线性的敏感性:

线性回归对异常值（离群点）敏感，这可能导致模型的不稳定性。同时，当自变量之间存在高度相关性（共线性）时，线性回归模型的系数估计可能不准确。

BP神经网络对异常值的鲁棒性相对较好，因为它可以通过调整权重来适应数据中的异常情况。但是，**BP神经网络**的参数选择（如学习率、激活函数的选择等）直接影响到网络的性能和训练结果。

4. 预测能力:

线性回归在变量之间存在线性关系时表现良好，但在面对非线性关系时可能无法提供准确的预测。

BP神经网络能够处理非线性关系，这使得它在数据拟合和预测方面具有强大的能力，尤其是在数据具有复杂模式时。

5. 训练过程中的问题：

线性回归通常不需要担心过拟合问题，因为它的模型结构相对简单。

BP神经网络在训练过程中可能会遇到局部极小化问题，即算法可能陷入局部极值，导致网络训练失败。此外，BP神经网络对初始网络权重非常敏感，不同的权重初始化可能导致不同的训练结果。

② 语音性别识别

题目：

根据声音样本来确定一个人的性别看似是一件很容易的事情。通常，人耳就可以很容易辨别出男性与女性声音的差异。那么，如何让计算机也能自动识别出男性或者女性的声音呢。

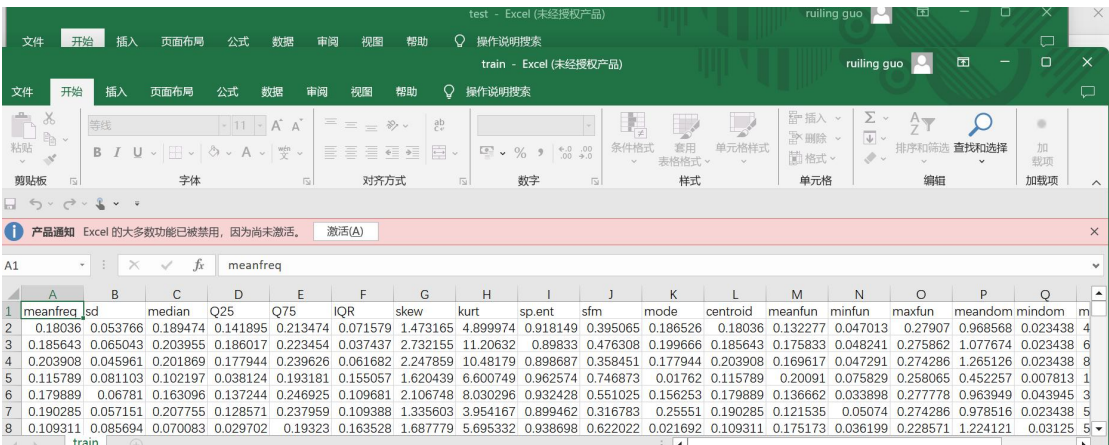
为了通过声音辨别性别，需要一个训练数据库。这个数据库由成千上万的男性和女性的声音样本建立起来，每个声音都以男性或女性的性别标记。

每个语音样本都存储为.WAV文件，然后对其进行预处理，测量声学信号的声学参数。预处理WAV文件的输出保存为一个CSV文件。

本实验，你将利用声音样本数据集，建立神经网络模型对声音进行分类。请按要求完成实验。

数据集：

文件train.csv为该实验的训练集，包含2536个声音样本，对应数据集的每行数据。每行数据包含每个样本的20个特征和样本的类别信息（男性或女性）。文件test.csv为测试集，用于评估算法性能。格式与训练集相同。



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	
1	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	mode	centroid	meanfun	minfun	maxfun	meandom	mindom	m
2	0.18036	0.053766	0.189474	0.141895	0.213474	0.071579	1.473165	4.899974	0.918149	0.395065	0.186526	0.18036	0.132277	0.047013	0.27907	0.968568	0.023438	4
3	0.185643	0.065043	0.203955	0.186017	0.223454	0.037437	2.732155	11.20632	0.89833	0.476308	0.199666	0.185643	0.175833	0.048241	0.275862	1.077674	0.023438	6
4	0.203908	0.045961	0.201869	0.177944	0.239626	0.061682	2.247859	10.48179	0.898687	0.358451	0.177944	0.203908	0.169617	0.047291	0.274286	1.265126	0.023438	8
5	0.115789	0.081103	0.102197	0.038124	0.193181	0.155057	1.620439	6.600749	0.962574	0.746873	0.01762	0.115789	0.20091	0.075829	0.258065	0.452257	0.007813	1
6	0.179889	0.06781	0.163096	0.137244	0.246925	0.109681	2.106748	8.030296	0.932428	0.551025	0.156253	0.179889	0.136662	0.033898	0.277778	0.963949	0.043945	3
7	0.190285	0.057151	0.207755	0.128571	0.237959	0.109388	1.335603	3.954167	0.899462	0.316783	0.25551	0.190285	0.121535	0.05074	0.274286	0.978516	0.023438	5
8	0.109311	0.085694	0.070083	0.029702	0.19323	0.163528	1.687779	5.695332	0.938698	0.622022	0.021692	0.109311	0.175173	0.036199	0.228571	1.224121	0.03125	5

步骤与要求:

(1) 通过对训练集进行学习, 建立神经网络模型并利用测试集测试算法分类的正确率(accuracy)。具体实验步骤应包括数据的可视化和预处理, 模型建立与训练以及测试结果等。

VoiceNN类定义了该神经网络的结构, 三层神经网络。第一层和第二层后跟有ReLU激活函数和Dropout正则化层, 以防止过拟合。最后一层使用Sigmoid 激活函数, 因为这是一个二分类问题。

```
class VoiceNN(nn.Cell):
    """Neural Network for voice gender classification"""
    def __init__(self, input_size):
        super(VoiceNN, self).__init__()
        self.layer1 = nn.Dense(input_size, 64)
        self.layer2 = nn.Dense(64, 32)
        self.layer3 = nn.Dense(32, 1)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.3)
        self.sigmoid = nn.Sigmoid()
    def construct(self, x):
        x = self.dropout(self.relu(self.layer1(x)))
        x = self.dropout(self.relu(self.layer2(x)))
        x = self.sigmoid(self.layer3(x))
        return x
```

NetWithLoss同样用于输出观察损失率。

visualize_data 方法用于数据可视化, 包括特征分布和特征之间的相关性矩阵。特征分布图展示了男性和女性在前五个特征上的分布情况。相关性矩阵图展示了特征之间的相关性, 帮助识别可能的共线性问题。

```
def train_neural_network(self, epochs=1000):
    """Train neural network model"""
    print("\nTraining Neural Network...")
    self.nn_model = VoiceNN(input_size=self.X_train.shape[1])
    loss_fn = nn.BCELoss()
    net_with_loss = NetWithLoss(self.nn_model, loss_fn)
    optimizer = nn.Adam(self.nn_model.trainable_params())
    def forward_fn(data, target):
        return net_with_loss(data, target)
    grad_fn = ops.value_and_grad(forward_fn, None, optimizer.parameters)
    for epoch in range(epochs):
        loss, grads = grad_fn(self.X_train_ms, self.y_train_ms.reshape(-1,
        optimizer(grads)
```


`train_neural_network`用于训练神经网络，创建一个 **VoiceNN** 实例，使用二元交叉熵损失函数`nn.BCELoss`和`Adam`优化器，定义前向传播函数 `forward_fn`，使用 `ops.value_and_grad` 计算损失函数的梯度。在训练循环中，对每个 `epoch` 计算损失和梯度，并更新网络权重，每100个 `epoch` 打印一次损失。训练完成后，计算模型在测试集上的准确率，并打印结果。

(2) 尝试使用其他机器学习方法进行实验，并作比较分析。

使用了逻辑回归、支持向量机、随机森林三种方法。

在`train_other_models`中，分别训练并计算精度，并与神经网络的精度比较。

```
124  def train_other_models(self):
125      """Train and evaluate other machine learning models"""
126      print("\nTraining Other Models...")
127
128      # Logistic Regression
129      self.lr_model = LogisticRegression()
130      self.lr_model.fit(self.X_train_scaled, self.y_train)
131      lr_pred = self.lr_model.predict(self.X_test_scaled)
132      self.lr_accuracy = accuracy_score(self.y_test, lr_pred)
133      print(f"Logistic Regression Accuracy: {self.lr_accuracy:.4f}")
134
135      # Support Vector Machine
136      self.svm_model = SVC()
137      self.svm_model.fit(self.X_train_scaled, self.y_train)
138      svm_pred = self.svm_model.predict(self.X_test_scaled)
139      self.svm_accuracy = accuracy_score(self.y_test, svm_pred)
140      print(f"SVM Accuracy: {self.svm_accuracy:.4f}")
141
142      # Random Forest
143      self.rf_model = RandomForestClassifier()
144      self.rf_model.fit(self.X_train_scaled, self.y_train)
```

比较：

1. 模型复杂度和灵活性

神经网络（**VoiceNN**）：具有较高的复杂度和灵活性，能够捕捉数据中的非线性关系。它通过学习特征之间的复杂交互来提高分类性能。

逻辑回归、支持向量机、随机森林：这些模型相对简单，适用于线性或可近似线性的数据关系。它们在处理非线性关系时可能不如神经网络灵活。

2. 训练时间和资源消耗

神经网络：通常需要更多的时间和计算资源来训练，特别是在深层或复杂的

网络结构中。

传统机器学习模型：通常训练时间较短，资源消耗较少，因为它们的算法和数学模型相对简单。

3. 预测准确性

神经网络：在数据集具有高度非线性特征时，神经网络可能提供更高的准确性。

逻辑回归、支持向量机、随机森林：在数据集线性可分或特征关系较为简单时，这些模型可能表现良好。特别是随机森林，它通过集成多个决策树来提高模型的稳定性和准确性。

4. 对数据规模和特征数量的敏感性

神经网络：对大规模数据和高维特征空间有很好的适应性，但需要足够的数据来避免过拟合。

传统机器学习模型：在特征数量较多或数据规模较大时，可能需要更多的特征选择或降维技术来提高性能。

5. 泛化能力

神经网络：如果训练得当，可以很好地泛化到未见过的数据，但风险是过拟合。

传统机器学习模型：通常泛化能力较好，特别是在数据集特征关系较为简单时。

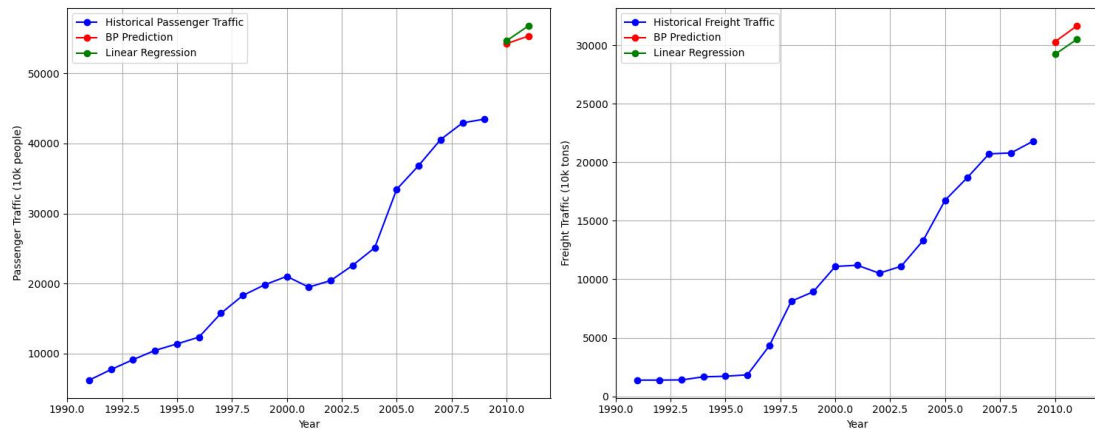
(3) 根据实验数据集，按实验要求给出相应的结果（截图）并对实验结果进行简要分析。

① 交通运量预测

BP 神经网络的丢失率：

Epoch 1600, Loss: 0.001685	Epoch 1600, Loss: 0.000271
Epoch 1700, Loss: 0.001663	Epoch 1700, Loss: 0.000254
Epoch 1800, Loss: 0.001651	Epoch 1800, Loss: 0.000240
Epoch 1900, Loss: 0.001645	Epoch 1900, Loss: 0.000228

模型训练到最后的时候丢失率都在 1%一下，拟合的效果不错。



BP Neural Network Predictions:

2010 Passenger Traffic: 54227 (10k people)
 2011 Passenger Traffic: 55310 (10k people)
 2010 Freight Traffic: 30302 (10k tons)
 2011 Freight Traffic: 31669 (10k tons)

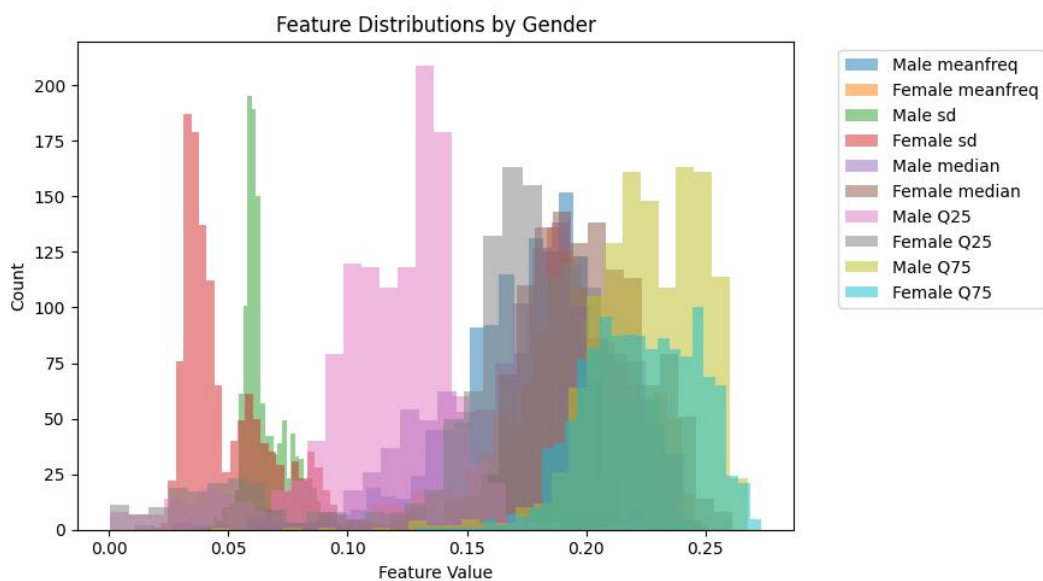
Linear Regression Predictions:

2010 Passenger Traffic: 54629 (10k people)
 2011 Passenger Traffic: 56774 (10k people)
 2010 Freight Traffic: 29232 (10k tons)
 2011 Freight Traffic: 30517 (10k tons)

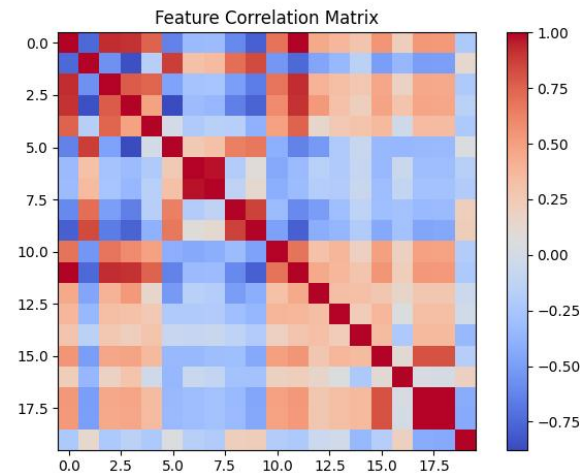
左图是对 Passenger Traffic 的历史数据和预测，右图是对 Freight Traffic 的。

两者的预测结果相差不多，对于 Passenger Traffic 都在 5400-5600 左右，对于 Freight Traffic 都在 2900-3100 左右。

② 语音性别识别

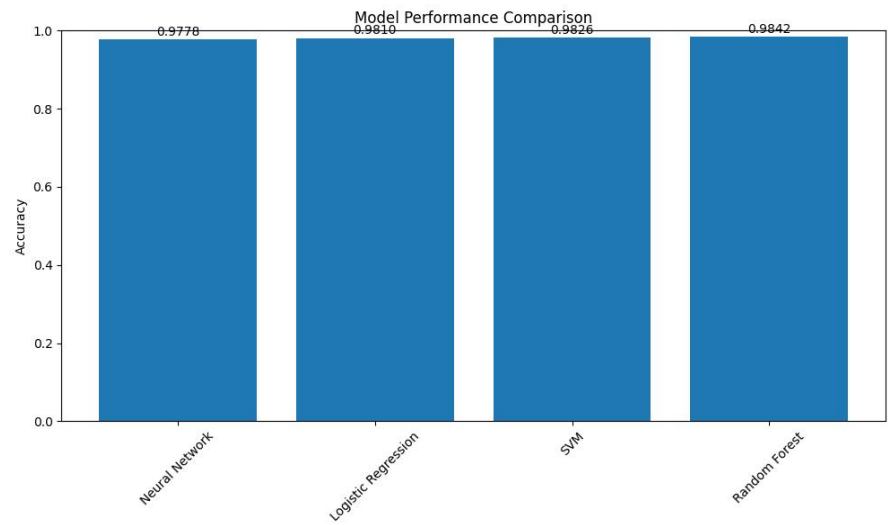


特征分布图展示了不同性别在各个特征上的分布情况。图中包含了多个重叠的直方图，每个直方图代表一个性别在特定特征上的值分布。我们可以看到 **male Q25**、**female sd**、**male sd** 等特征的集中性很高，还可以知道一些特征的平均性，比如 **female sd** 的平均值低于 **male sd**，**female Q25** 的平均值高于 **male Q25** 等。



相关性矩阵展示了不同特征之间的相关性。图中的颜色表示相关系数的大小和方向：深红色：表示特征之间有强烈的正相关关系。深蓝色：表示特征之间有强烈的负相关关系。白色或浅色：表示特征之间的相关性较低或无相关性。

通过这张图，我们可以识别特征之间可能存在的线性关系，如果两个特征之间的相关性非常高，可能意味着它们提供的信息是冗余的，可以考虑在模型中只使用其中一个。（但是在本模型中使用了全部的特征进行训练，画图的原因是题目要求）



```
Epoch 800, Loss: 0.0058
Epoch 900, Loss: 0.0039
Neural Network Accuracy: 0.9778

Training Other Models...
Logistic Regression Accuracy: 0.9810
SVM Accuracy: 0.9826
Random Forest Accuracy: 0.9842
```

这是对几种不同方法的正确率(accuracy)比较,可以看到其实正确率都是很高的,相差不大。

四、心得

五、程序文件名清单

main.py: 交通运量预测源代码

main.py.pdf: 交通运量预测源代码 pdf 版

main2.py: 语音性别识别源代码

main2.py.pdf: 语音性别识别源代码 pdf 版

六、附录

代码如下。

main.py

```
1  import numpy as np
2  import pandas as pd
3  from sklearn.preprocessing import MinMaxScaler
4  from sklearn.linear_model import LinearRegression
5  import matplotlib.pyplot as plt
6  import mindspore as ms
7  import mindspore.nn as nn
8  import mindspore.ops as ops
9  from mindspore import context, Tensor
10
11 # 设置MindSpore运行模式
12 context.set_context(mode=context.PYNATIVE_MODE, device_target="CPU")
13
14
15 class BPNeuralNetwork(nn.Cell):
16     """BP神经网络模型"""
17
18     def __init__(self):
19         super(BPNeuralNetwork, self).__init__()
20         # 使用三层神经网络结构
21         self.layer1 = nn.Dense(3, 16)
22         self.layer2 = nn.Dense(16, 8)
23         self.layer3 = nn.Dense(8, 1)
24         # 使用ReLU激活函数
25         self.relu = nn.ReLU()
26         # 添加BatchNorm层
27         self.bn1 = nn.BatchNorm1d(16)
28         self.bn2 = nn.BatchNorm1d(8)
29
30     def construct(self, x):
31         x = self.bn1(self.relu(self.layer1(x)))
32         x = self.bn2(self.relu(self.layer2(x)))
33         x = self.layer3(x)
34         return x
35
36
37 class NetWithLoss(nn.Cell):
38     def __init__(self, backbone, loss_fn):
39         super(NetWithLoss, self).__init__()
40         self.backbone = backbone
41         self.loss_fn = loss_fn
42
43     def construct(self, data, label):
44         out = self.backbone(data)
45         return self.loss_fn(out, label)
46
47
48 class TrafficPredictor:
49     def __init__(self):
50         try:
51             # 读取Excel文件中的历史数据 (list1)
52             df = pd.read_excel('list1.xls')
53
54             # 假设Excel文件的列顺序为: 年份, 人口数量, 机动车数量, 公路面积, 客运量, 货运量
55             self.years = df.iloc[:, 0].values # 第一列是年份
56             self.population = df.iloc[:, 1].values # 人口数量
57             self.vehicles = df.iloc[:, 2].values # 机动车数量
58             self.road_area = df.iloc[:, 3].values # 公路面积
59             self.passengers = df.iloc[:, 4].values # 客运量
60             self.freight = df.iloc[:, 5].values # 货运量
61
62             # 2010和2011年的预测数据
63             self.predict_population = np.array([73.39, 75.55]) # 人口数量
64             self.predict_vehicles = np.array([3.9635, 4.0975]) # 机动车数量
65             self.predict_road_area = np.array([0.9880, 1.0268]) # 公路面积
66
67             # 初始化标准化器
68             self.scaler_X = MinMaxScaler()
69             self.scaler_y1 = MinMaxScaler()
70             self.scaler_y2 = MinMaxScaler()
71
72             # 准备训练数据
73             self.X = np.column_stack((self.population, self.vehicles, self.road_area))
74             self.X_scaled = self.scaler_X.fit_transform(self.X)
75             self.y1_scaled = self.scaler_y1.fit_transform(self.passengers.reshape(-1, 1))
76             self.y2_scaled = self.scaler_y2.fit_transform(self.freight.reshape(-1, 1))
77
78             # 准备预测数据
79             self.X_pred = np.column_stack((self.predict_population, self.predict_vehicles, self.predict_road_area))
```

```

80         self.X_pred_scaled = self.scaler_X.transform(self.X_pred)
81
82         # 转换为MindSpore张量
83         self.X_scaled_ms = Tensor(self.X_scaled.astype(np.float32))
84         self.y1_scaled_ms = Tensor(self.y1_scaled.astype(np.float32))
85         self.y2_scaled_ms = Tensor(self.y2_scaled.astype(np.float32))
86         self.X_pred_scaled_ms = Tensor(self.X_pred_scaled.astype(np.float32))
87
88     except Exception as e:
89         print(f"读取数据时发生错误: {str(e)}")
90         raise
91
92     def train_single_model(self, model, X, y, learning_rate=0.001, epochs=2000):
93         """训练单个模型"""
94         loss_fn = nn.MSELoss()
95         net_with_loss = NetWithLoss(model, loss_fn)
96         optimizer = nn.Adam(model.trainable_params(), learning_rate=learning_rate)
97
98         def forward_fn(data, target):
99             return net_with_loss(data, target)
100
101         grad_fn = ops.value_and_grad(forward_fn, None, optimizer.parameters)
102
103         for epoch in range(epochs):
104             loss, grads = grad_fn(X, y)
105             optimizer(grads)
106
107             if epoch % 100 == 0:
108                 print(f"Epoch {epoch}, Loss: {loss.asnumpy():.6f}")
109
110         return model
111
112     def train_models(self):
113         """训练BP神经网络和线性回归模型"""
114         # 训练BP神经网络模型
115         print("\n训练BP神经网络模型...")
116
117         # 客运量预测模型
118         self.bp_model_passengers = BPNeuralNetwork()
119         print("训练客运量预测模型...")
120         self.bp_model_passengers = self.train_single_model(
121             self.bp_model_passengers,
122             self.X_scaled_ms,
123             self.y1_scaled_ms
124         )
125
126         # 货运量预测模型
127         self.bp_model_freight = BPNeuralNetwork()
128         print("\n训练货运量预测模型...")
129         self.bp_model_freight = self.train_single_model(
130             self.bp_model_freight,
131             self.X_scaled_ms,
132             self.y2_scaled_ms
133         )
134
135         # BP预测结果
136         self.bp_pred_passengers = self.scaler_y1.inverse_transform(
137             self.bp_model_passengers(self.X_pred_scaled_ms).asnumpy().reshape(-1, 1))
138         self.bp_pred_freight = self.scaler_y2.inverse_transform(
139             self.bp_model_freight(self.X_pred_scaled_ms).asnumpy().reshape(-1, 1))
140
141         # 训练线性回归模型
142         print("\n训练线性回归模型...")
143         self.lr_passengers = LinearRegression()
144         self.lr_freight = LinearRegression()
145
146         self.lr_passengers.fit(self.X_scaled, self.y1_scaled)
147         self.lr_freight.fit(self.X_scaled, self.y2_scaled)
148
149         # 线性回归预测结果
150         self.lr_pred_passengers = self.scaler_y1.inverse_transform(
151             self.lr_passengers.predict(self.X_pred_scaled).reshape(-1, 1))
152         self.lr_pred_freight = self.scaler_y2.inverse_transform(
153             self.lr_freight.predict(self.X_pred_scaled).reshape(-1, 1))
154
155     def print_predictions(self):
156         """打印预测结果"""
157         print("\nBP Neural Network Predictions:")
158         print("2010 Passenger Traffic: {:.0f} (10k people)".format(self.bp_pred_passengers[0][0]))
159         print("2011 Passenger Traffic: {:.0f} (10k people)".format(self.bp_pred_passengers[1][0]))
160         print("2010 Freight Traffic: {:.0f} (10k tons)".format(self.bp_pred_freight[0][0]))
161         print("2011 Freight Traffic: {:.0f} (10k tons)".format(self.bp_pred_freight[1][0]))

```



```
162
163     print("\nLinear Regression Predictions:")
164     print("2010 Passenger Traffic: {:.0f} (10k people)".format(self.lr_pred_passengers[0][0]))
165     print("2011 Passenger Traffic: {:.0f} (10k people)".format(self.lr_pred_passengers[1][0]))
166     print("2010 Freight Traffic: {:.0f} (10k tons)".format(self.lr_pred_freight[0][0]))
167     print("2011 Freight Traffic: {:.0f} (10k tons)".format(self.lr_pred_freight[1][0]))
168
169     def plot_results(self):
170         """Visualization of results"""
171         # Set font to DejaVu Sans explicitly
172         plt.rcParams['font.family'] = 'DejaVu Sans'
173         # 设置中文字体
174         plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签
175         plt.rcParams['axes.unicode_minus'] = False # 用来正常显示负号
176         plt.figure(figsize=(15, 6))
177
178         # Passenger traffic plot
179         plt.subplot(1, 2, 1)
180         plt.plot(self.years, self.passengers, 'bo-', label='Historical Passenger Traffic')
181         plt.plot([2010, 2011], self.bp_pred_passengers, 'ro-', label='BP Prediction')
182         plt.plot([2010, 2011], self.lr_pred_passengers, 'go-', label='Linear Regression')
183         plt.xlabel('Year')
184         plt.ylabel('Passenger Traffic (10k people)')
185         plt.legend()
186         plt.grid(True)
187
188         # Freight traffic plot
189         plt.subplot(1, 2, 2)
190         plt.plot(self.years, self.freight, 'bo-', label='Historical Freight Traffic')
191         plt.plot([2010, 2011], self.bp_pred_freight, 'ro-', label='BP Prediction')
192         plt.plot([2010, 2011], self.lr_pred_freight, 'go-', label='Linear Regression')
193         plt.xlabel('Year')
194         plt.ylabel('Freight Traffic (10k tons)')
195         plt.legend()
196         plt.grid(True)
197
198         plt.tight_layout()
199         plt.show()
200
201
202     def main():
203         try:
204             predictor = TrafficPredictor()
205             predictor.train_models()
206             predictor.print_predictions()
207             predictor.plot_results()
208         except Exception as e:
209             print(f"错误: {str(e)}")
210             raise
211
212
213     if __name__ == "__main__":
214         main()
```

main2.py

```
1  import numpy as np
2  import pandas as pd
3  from sklearn.preprocessing import StandardScaler
4  from sklearn.metrics import accuracy_score, confusion_matrix
5  import matplotlib.pyplot as plt
6  from sklearn.linear_model import LogisticRegression
7  from sklearn.svm import SVC
8  from sklearn.ensemble import RandomForestClassifier
9  import mindspore as ms
10 import mindspore.nn as nn
11 import mindspore.ops as ops
12 from mindspore import context, Tensor
13
14 # Set MindSpore context
15 context.set_context(mode=context.PYNATIVE_MODE, device_target="CPU")
16
17
18 class VoiceNN(nn.Cell):
19     """Neural Network for voice gender classification"""
20
21     def __init__(self, input_size):
22         super(VoiceNN, self).__init__()
23         self.layer1 = nn.Dense(input_size, 64)
24         self.layer2 = nn.Dense(64, 32)
25         self.layer3 = nn.Dense(32, 1)
26         self.relu = nn.ReLU()
27         self.dropout = nn.Dropout(0.3)
28         self.sigmoid = nn.Sigmoid()
29
30     def construct(self, x):
31         x = self.dropout(self.relu(self.layer1(x)))
32         x = self.dropout(self.relu(self.layer2(x)))
33         x = self.sigmoid(self.layer3(x))
34         return x
35
36
37 class NetWithLoss(nn.Cell):
38     def __init__(self, backbone, loss_fn):
39         super(NetWithLoss, self).__init__()
40         self.backbone = backbone
41         self.loss_fn = loss_fn
42
43     def construct(self, data, label):
44         out = self.backbone(data)
45         return self.loss_fn(out, label)
46
47
48 class VoiceGenderClassifier:
49     def __init__(self):
50         # Load and preprocess data
51         self.train_data = pd.read_csv('train.csv')
52         self.test_data = pd.read_csv('test.csv')
53
54         # Separate features and labels
55         self.X_train = self.train_data.iloc[:, :-1]
56         self.y_train = (self.train_data.iloc[:, -1] == 'male').astype(int)
57         self.X_test = self.test_data.iloc[:, :-1]
58         self.y_test = (self.test_data.iloc[:, -1] == 'male').astype(int)
59
60         # Scale features
61         self.scaler = StandardScaler()
62         self.X_train_scaled = self.scaler.fit_transform(self.X_train)
63         self.X_test_scaled = self.scaler.transform(self.X_test)
64
65         # Convert to MindSpore tensors
66         self.X_train_ms = Tensor(self.X_train_scaled.astype(np.float32))
```



```

67         self.y_train_ms = Tensor(self.y_train.values.astype(np.float32))
68         self.X_test_ms = Tensor(self.X_test_scaled.astype(np.float32))
69
70     def visualize_data(self):
71         """Data visualization"""
72         plt.figure(figsize=(15, 5))
73
74         # Plot feature distributions
75         plt.subplot(121)
76         features = self.train_data.columns[:-1]
77         for i, feature in enumerate(features[:5]): # Plot first 5 features
78             male_data = self.train_data[self.train_data.iloc[:, -1] == 'male'][feature]
79             female_data = self.train_data[self.train_data.iloc[:, -1] == 'female'][feature]
80
81             plt.hist(male_data, alpha=0.5, bins=30, label=f'Male {feature}')
82             plt.hist(female_data, alpha=0.5, bins=30, label=f'Female {feature}')
83
84         plt.title('Feature Distributions by Gender')
85         plt.xlabel('Feature Value')
86         plt.ylabel('Count')
87         plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
88
89         # Plot correlation matrix
90         plt.subplot(122)
91         corr_matrix = self.train_data.iloc[:, :-1].corr()
92         plt.imshow(corr_matrix, cmap='coolwarm', aspect='equal')
93         plt.colorbar()
94         plt.title('Feature Correlation Matrix')
95
96         plt.tight_layout()
97         plt.show()
98
99     def train_neural_network(self, epochs=1000):
100         """Train neural network model"""
101         print("\nTraining Neural Network...")
102         self.nn_model = VoiceNN(input_size=self.X_train.shape[1])
103         loss_fn = nn.BCELoss()
104         net_with_loss = NetWithLoss(self.nn_model, loss_fn)
105         optimizer = nn.Adam(self.nn_model.trainable_params())
106
107         def forward_fn(data, target):
108             return net_with_loss(data, target)
109
110         grad_fn = ops.value_and_grad(forward_fn, None, optimizer.parameters)
111
112         for epoch in range(epochs):
113             loss, grads = grad_fn(self.X_train_ms, self.y_train_ms.reshape(-1, 1))
114             optimizer(grads)
115
116             if epoch % 100 == 0:
117                 print(f"Epoch {epoch}, Loss: {loss.asnumpy():.4f}")
118
119         # Calculate accuracy
120         predictions = (self.nn_model(self.X_test_ms).asnumpy() > 0.5).astype(int)
121         self.nn_accuracy = accuracy_score(self.y_test, predictions)
122         print(f"Neural Network Accuracy: {self.nn_accuracy:.4f}")
123
124     def train_other_models(self):
125         """Train and evaluate other machine learning models"""
126         print("\nTraining Other Models...")
127
128         # Logistic Regression
129         self.lr_model = LogisticRegression()
130         self.lr_model.fit(self.X_train_scaled, self.y_train)
131         lr_pred = self.lr_model.predict(self.X_test_scaled)
132         self.lr_accuracy = accuracy_score(self.y_test, lr_pred)
133         print(f"Logistic Regression Accuracy: {self.lr_accuracy:.4f}")
134
135         # Support Vector Machine

```

```
136         self.svm_model = SVC()
137         self.svm_model.fit(self.X_train_scaled, self.y_train)
138         svm_pred = self.svm_model.predict(self.X_test_scaled)
139         self.svm_accuracy = accuracy_score(self.y_test, svm_pred)
140         print(f"SVM Accuracy: {self.svm_accuracy:.4f}")
141
142         # Random Forest
143         self.rf_model = RandomForestClassifier()
144         self.rf_model.fit(self.X_train_scaled, self.y_train)
145         rf_pred = self.rf_model.predict(self.X_test_scaled)
146         self.rf_accuracy = accuracy_score(self.y_test, rf_pred)
147         print(f"Random Forest Accuracy: {self.rf_accuracy:.4f}")
148
149     def plot_results(self):
150         """Plot comparison of model performances"""
151         models = ['Neural Network', 'Logistic Regression', 'SVM', 'Random Forest']
152         accuracies = [self.nn_accuracy, self.lr_accuracy, self.svm_accuracy, self.rf_accuracy]
153
154         plt.figure(figsize=(10, 6))
155         plt.bar(models, accuracies)
156         plt.title('Model Performance Comparison')
157         plt.ylabel('Accuracy')
158         plt.ylim(0, 1)
159         for i, v in enumerate(accuracies):
160             plt.text(i, v + 0.01, f'{v:.4f}', ha='center')
161         plt.xticks(rotation=45)
162         plt.tight_layout()
163         plt.show()
164
165
166 def main():
167     # Set matplotlib font settings
168     plt.rcParams['font.family'] = 'DejaVu Sans'
169     plt.rcParams['font.size'] = 10
170
171     try:
172         # Create classifier instance
173         classifier = VoiceGenderClassifier()
174
175         # Visualize data
176         classifier.visualize_data()
177
178         # Train and evaluate models
179         classifier.train_neural_network()
180         classifier.train_other_models()
181
182         # Plot results comparison
183         classifier.plot_results()
184
185     except Exception as e:
186         print(f"Error: {str(e)}")
187         raise
188
189
190 if __name__ == "__main__":
191     main()
```