Deep Learning for Business: Customer Shopping Preferences

Introduction

Business Problem and Significance

The business problem addressed in this project is understanding and predicting customer shopping preferences to enhance business decision-making. This is significant as it can help businesses tailor their products, marketing strategies, and overall customer experience to better meet customer needs.

Project Goals and Objectives

The goal of this project is to develop predictive models using deep learning to forecast customer purchase behaviors and trends. The specific objectives include:

1. Preprocessing the data to handle missing values, encode categorical variables, and scale numerical features.
2. Experimenting with various deep learning architectures, including Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), Bidirectional RNNs, and pretrained-inspired models.
3. Fine-tuning the models to optimize their performance.
4. Evaluating the models using appropriate metrics to determine their effectiveness in predicting customer shopping trends.

Benefits to the Business

The outcomes of this project will enable businesses to make informed decisions based on data-driven insights. By accurately predicting customer purchase behaviors, businesses can:

* Optimize their inventory management to reduce overstock and stockouts.
* Improve personalized marketing strategies, resulting in higher customer engagement and conversion rates.
* Enhance customer relationship management by understanding customer preferences and providing tailored experiences.

Implications on Decision-Making Processes

The predictive models developed in this project will provide valuable insights that can influence various business decisions, such as:

* **Inventory Management:** By predicting which products are likely to be in high demand, businesses can adjust their inventory levels accordingly.
* **Marketing Strategies:** Insights into customer preferences can help design more effective marketing campaigns, targeting the right customers with the right products.
* **Customer Relationship Management:** Understanding customer behavior patterns allows businesses to personalize their interactions with customers, leading to increased customer loyalty and satisfaction.

Dataset Source

The dataset used in this project is sourced from Kaggle, titled "Customer Shopping Trends Dataset" provided by Sourav Banerjee. This dataset contains updated information on customer shopping trends, including various attributes related to customer demographics, purchase behaviors, and product details. The dataset can be accessed [here](https://www.google.com/url?q=https%3A%2F%2Fwww.kaggle.com%2Fdatasets%2Fiamsouravbanerjee%2Fcustomer-shopping-trends-dataset%3Fselect%3Dshopping_trends_updated.csv). This dataset is crucial for developing and training the predictive models to achieve the project's objectives.

keyboard\_arrow\_down

Step 2: Describing Dataset

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

# Load the dataset

df = pd.read\_csv('/content/shopping\_trends.csv')

# Determine the dimensions of the dataset

print(f"Dataset Dimensions: {df.shape}")

# Check for missing values

missing\_values = df.isnull().sum()

print("Missing Values:\n", missing\_values)

# Descriptive statistics

descriptive\_stats = df.describe(include='all')

print("Descriptive Statistics:\n", descriptive\_stats)

# Visualize the data

# Scatter plot for age vs purchase amount

sns.scatterplot(x='Age', y='Purchase Amount (USD)', data=df)

plt.title('Age vs Purchase Amount')

plt.show()

# Encode categorical variables before calculating correlation matrix

label\_encoders = {}

categorical\_columns = ['Gender', 'Item Purchased', 'Category', 'Location', 'Size', 'Color', 'Season',

'Subscription Status', 'Shipping Type', 'Discount Applied', 'Promo Code Used',

'Payment Method', 'Frequency of Purchases', 'Preferred Payment Method']

for col in categorical\_columns:

le = LabelEncoder()

df[col] = le.fit\_transform(df[col])

label\_encoders[col] = le

# Correlation plot

correlation\_matrix = df.corr()

plt.figure(figsize=(12, 8))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')

plt.title('Correlation Matrix')

plt.show()

# Bar plots for categorical features

plt.figure(figsize=(20, 15))

for i, col in enumerate(categorical\_columns, 1):

plt.subplot(5, 3, i)

sns.countplot(data=df, x=col)

plt.title(f'Distribution of {col}')

plt.xticks(rotation=90)

plt.tight\_layout()

plt.show()

# Distribution plots for numerical features

numerical\_columns = ['Age', 'Purchase Amount (USD)', 'Previous Purchases', 'Review Rating']

plt.figure(figsize=(15, 10))

for i, col in enumerate(numerical\_columns, 1):

plt.subplot(2, 2, i)

sns.histplot(df[col], kde=True)

plt.title(f'Distribution of {col}')

plt.tight\_layout()

plt.show()

# Box plots for numerical features to identify outliers

plt.figure(figsize=(15, 10))

for i, col in enumerate(numerical\_columns, 1):

plt.subplot(2, 2, i)

sns.boxplot(data=df, y=col)

plt.title(f'Box plot of {col}')

plt.tight\_layout()

plt.show()

Dataset Dimensions: (3900, 19)

Missing Values:

Customer ID 0

Age 0

Gender 0

Item Purchased 0

Category 0

Purchase Amount (USD) 0

Location 0

Size 0

Color 0

Season 0

Review Rating 0

Subscription Status 0

Payment Method 0

Shipping Type 0

Discount Applied 0

Promo Code Used 0

Previous Purchases 0

Preferred Payment Method 0

Frequency of Purchases 0

dtype: int64

Descriptive Statistics:

Customer ID Age Gender Item Purchased Category \

count 3900.000000 3900.000000 3900 3900 3900

unique NaN NaN 2 25 4

top NaN NaN Male Blouse Clothing

freq NaN NaN 2652 171 1737

mean 1950.500000 44.068462 NaN NaN NaN

std 1125.977353 15.207589 NaN NaN NaN

min 1.000000 18.000000 NaN NaN NaN

25% 975.750000 31.000000 NaN NaN NaN

50% 1950.500000 44.000000 NaN NaN NaN

75% 2925.250000 57.000000 NaN NaN NaN

max 3900.000000 70.000000 NaN NaN NaN

Purchase Amount (USD) Location Size Color Season Review Rating \

count 3900.000000 3900 3900 3900 3900 3900.000000

unique NaN 50 4 25 4 NaN

top NaN Montana M Olive Spring NaN

freq NaN 96 1755 177 999 NaN

mean 59.764359 NaN NaN NaN NaN 3.749949

std 23.685392 NaN NaN NaN NaN 0.716223

min 20.000000 NaN NaN NaN NaN 2.500000

25% 39.000000 NaN NaN NaN NaN 3.100000

50% 60.000000 NaN NaN NaN NaN 3.700000

75% 81.000000 NaN NaN NaN NaN 4.400000

max 100.000000 NaN NaN NaN NaN 5.000000

Subscription Status Payment Method Shipping Type Discount Applied \

count 3900 3900 3900 3900

unique 2 6 6 2

top No Credit Card Free Shipping No

freq 2847 696 675 2223

mean NaN NaN NaN NaN

std NaN NaN NaN NaN

min NaN NaN NaN NaN

25% NaN NaN NaN NaN

50% NaN NaN NaN NaN

75% NaN NaN NaN NaN

max NaN NaN NaN NaN

Promo Code Used Previous Purchases Preferred Payment Method \

count 3900 3900.000000 3900

unique 2 NaN 6

top No NaN PayPal

freq 2223 NaN 677

mean NaN 25.351538 NaN

std NaN 14.447125 NaN

min NaN 1.000000 NaN

25% NaN 13.000000 NaN

50% NaN 25.000000 NaN

75% NaN 38.000000 NaN

max NaN 50.000000 NaN

Frequency of Purchases

count 3900

unique 7

top Every 3 Months

freq 584

mean NaN

std NaN

min NaN

25% NaN

50% NaN

75% NaN

max NaN

A graph of blue dots

Description automatically generated

A screenshot of a graph

Description automatically generated

A group of blue rectangular bars

Description automatically generated

A graph of sales and distribution of purchase

Description automatically generated with medium confidence

A group of blue rectangular boxes

Description automatically generated

Step 3: Data Preprocessing

[2]

0s

import numpy as np  
from sklearn.preprocessing import StandardScaler  
  
# Handle missing values by filling with forward fill method  
df.fillna(method='ffill', inplace=True)  
  
# Encoding categorical variables using LabelEncoder  
label\_encoders = {}  
for col in categorical\_columns:  
    le = LabelEncoder()  
    df[col] = le.fit\_transform(df[col])  
    label\_encoders[col] = le  
  
# Feature scaling for numerical columns  
scaler = StandardScaler()  
df[['Age', 'Purchase Amount (USD)', 'Previous Purchases']] = scaler.fit\_transform(df[['Age', 'Purchase Amount (USD)', 'Previous Purchases']])  
  
# Prepare numerical features  
X\_numerical = df[['Age', 'Purchase Amount (USD)', 'Previous Purchases']].values  
y = df['Review Rating'].values  
  
# Display preprocessed data  
print(df.head())  
print(X\_numerical.shape, y.shape)

Customer ID Age Gender Item Purchased Category \

0 1 0.718913 1 2 1

1 2 -1.648629 1 23 1

2 3 0.390088 1 11 1

3 4 -1.517099 1 14 2

4 5 0.061263 1 2 1

Purchase Amount (USD) Location Size Color Season Review Rating \

0 -0.285629 16 0 7 3 3.1

1 0.178852 18 0 12 3 3.1

2 0.558882 20 2 12 1 3.1

3 1.276716 38 1 12 1 3.5

4 -0.454531 36 1 21 1 2.7

Subscription Status Payment Method Shipping Type Discount Applied \

0 1 2 1 1

1 1 0 1 1

2 1 1 2 1

3 1 4 3 1

4 1 1 2 1

Promo Code Used Previous Purchases Preferred Payment Method \

0 1 -0.785831 5

1 1 -1.616552 1

2 1 -0.162789 2

3 1 1.637107 4

4 1 0.391025 4

Frequency of Purchases

0 3

1 3

2 6

3 6

4 0

(3900, 3) (3900,)

addCode

addText

Step 4: Fitting Different Deep Learning Architectures

Imports and Data Preparation

[5]

0s

import numpy as np

from sklearn.preprocessing import StandardScaler, LabelEncoder

import pandas as pd

import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dense, Embedding, Flatten, Concatenate, LSTM, SimpleRNN, Conv1D, MaxPooling1D, Bidirectional

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

# Load the dataset

df = pd.read\_csv('shopping\_trends.csv')

# Handle missing values by filling with forward fill method

df.fillna(method='ffill', inplace=True)

# Encoding categorical variables using LabelEncoder

label\_encoders = {}

categorical\_columns = ['Gender', 'Item Purchased', 'Category', 'Location', 'Size', 'Color', 'Season',

                       'Subscription Status', 'Payment Method', 'Shipping Type', 'Discount Applied',

                       'Promo Code Used', 'Preferred Payment Method', 'Frequency of Purchases']

for col in categorical\_columns:

    le = LabelEncoder()

    df[col] = le.fit\_transform(df[col])

    label\_encoders[col] = le

# Feature scaling for numerical columns

scaler = StandardScaler()

df[['Age', 'Purchase Amount (USD)', 'Previous Purchases']] = scaler.fit\_transform(df[['Age', 'Purchase Amount (USD)', 'Previous Purchases']])

# Prepare numerical features and target variable

X\_numerical = df[['Age', 'Purchase Amount (USD)', 'Previous Purchases']].values

y = df['Review Rating'].values

# Split data into training and testing sets

X\_train\_num, X\_test\_num, y\_train, y\_test = train\_test\_split(X\_numerical, y, test\_size=0.2, random\_state=42)

# Split categorical data

X\_train\_cat = [df[col].values[:X\_train\_num.shape[0]] for col in categorical\_columns]

X\_test\_cat = [df[col].values[X\_train\_num.shape[0]:] for col in categorical\_columns]

X\_train = [X\_train\_num] + [np.array(cat) for cat in X\_train\_cat]

X\_test = [X\_test\_num] + [np.array(cat) for cat in X\_test\_cat]

# Define the MAPE metric

def mape(y\_true, y\_pred):

    y\_true = tf.convert\_to\_tensor(y\_true)

    y\_pred = tf.convert\_to\_tensor(y\_pred)

    return tf.reduce\_mean(tf.abs((y\_true - y\_pred) / y\_true)) \* 100

# Define embedding input layers for categorical features

def create\_embedding\_layers(input\_data, categorical\_columns, embedding\_dim=8):

    inputs = []

    embeddings = []

    for col in categorical\_columns:

        vocab\_size = input\_data[col].nunique()

        input\_layer = Input(shape=(1,), name=f'{col}\_input')

        embedding\_layer = Embedding(input\_dim=vocab\_size, output\_dim=embedding\_dim, input\_length=1)(input\_layer)

        flatten\_layer = Flatten()(embedding\_layer)

        inputs.append(input\_layer)

        embeddings.append(flatten\_layer)

    return inputs, embeddings

# Plot training and validation loss

def plot\_history(history, title):

    plt.plot(history.history['loss'], label='Training Loss')

    plt.plot(history.history['val\_loss'], label='Validation Loss')

    plt.title(title)

    plt.xlabel('Epochs')

    plt.ylabel('Loss')

    plt.legend()

    plt.show()

# Prepare data inputs for the model

numerical\_input\_data = X\_numerical

categorical\_input\_data = [df[col].values for col in categorical\_columns]

Simple Neural Network with Embeddings

[6]

12s

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau  
  
# Simple Neural Network with Embeddings and Improvements  
def create\_nn\_model\_with\_embeddings(input\_data, embedding\_dim=8):  
    numerical\_input = Input(shape=(X\_numerical.shape[1],))  
    inputs, embeddings = create\_embedding\_layers(input\_data, categorical\_columns, embedding\_dim)  
    merged\_embeddings = Concatenate()(embeddings)  
    merged\_inputs = Concatenate()([numerical\_input, merged\_embeddings])  
    x = Dense(128, activation='relu')(merged\_inputs)  
    x = Dense(64, activation='relu')(x)  
    x = Dense(32, activation='relu')(x)  
    output = Dense(1)(x)  
    model = Model(inputs=[numerical\_input] + inputs, outputs=output)  
    model.compile(optimizer='adam', loss='mse', metrics=[mape])  
    return model  
  
# Create and train the model with early stopping and learning rate reduction  
nn\_model\_with\_embeddings = create\_nn\_model\_with\_embeddings(df)  
early\_stopping = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)  
reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.2, patience=5, min\_lr=0.0001)  
nn\_history = nn\_model\_with\_embeddings.fit(  
    [X\_train\_num] + X\_train\_cat, y\_train, epochs=50, batch\_size=32, validation\_split=0.2,  
    callbacks=[early\_stopping, reduce\_lr]  
)  
nn\_loss, nn\_mape = nn\_model\_with\_embeddings.evaluate([X\_test\_num] + X\_test\_cat, y\_test)  
print(f"NN with Embeddings Mean Absolute Percentage Error: {nn\_mape}")  
  
# Plot training and validation loss for NN with Embeddings  
plot\_history(nn\_history, 'NN with Embeddings Training and Validation Loss')

Epoch 1/50

78/78 [==============================] - 3s 9ms/step - loss: 3.6805 - mape: 39.4027 - val\_loss: 0.7098 - val\_mape: 18.3785 - lr: 0.0010

Epoch 2/50

78/78 [==============================] - 1s 8ms/step - loss: 0.5952 - mape: 18.1513 - val\_loss: 0.6197 - val\_mape: 17.4886 - lr: 0.0010

Epoch 3/50

78/78 [==============================] - 1s 7ms/step - loss: 0.5330 - mape: 17.3712 - val\_loss: 0.5562 - val\_mape: 18.0847 - lr: 0.0010

Epoch 4/50

78/78 [==============================] - 1s 8ms/step - loss: 0.5104 - mape: 17.0898 - val\_loss: 0.6024 - val\_mape: 17.5259 - lr: 0.0010

Epoch 5/50

78/78 [==============================] - 1s 10ms/step - loss: 0.5091 - mape: 17.0075 - val\_loss: 0.5476 - val\_mape: 17.4030 - lr: 0.0010

Epoch 6/50

78/78 [==============================] - 1s 8ms/step - loss: 0.4908 - mape: 16.7431 - val\_loss: 0.5650 - val\_mape: 17.6689 - lr: 0.0010

Epoch 7/50

78/78 [==============================] - 1s 11ms/step - loss: 0.4859 - mape: 16.7069 - val\_loss: 0.5605 - val\_mape: 18.0081 - lr: 0.0010

Epoch 8/50

78/78 [==============================] - 1s 6ms/step - loss: 0.4832 - mape: 16.6248 - val\_loss: 0.5666 - val\_mape: 17.7857 - lr: 0.0010

Epoch 9/50

78/78 [==============================] - 0s 6ms/step - loss: 0.4774 - mape: 16.5203 - val\_loss: 0.5709 - val\_mape: 18.1599 - lr: 0.0010

Epoch 10/50

78/78 [==============================] - 0s 6ms/step - loss: 0.4746 - mape: 16.4738 - val\_loss: 0.5808 - val\_mape: 17.9338 - lr: 0.0010

Epoch 11/50

78/78 [==============================] - 0s 6ms/step - loss: 0.4487 - mape: 16.0698 - val\_loss: 0.5979 - val\_mape: 17.6641 - lr: 2.0000e-04

Epoch 12/50

78/78 [==============================] - 0s 6ms/step - loss: 0.4468 - mape: 16.0167 - val\_loss: 0.5899 - val\_mape: 17.6680 - lr: 2.0000e-04

Epoch 13/50

78/78 [==============================] - 0s 5ms/step - loss: 0.4436 - mape: 15.9788 - val\_loss: 0.5870 - val\_mape: 17.6963 - lr: 2.0000e-04

Epoch 14/50

78/78 [==============================] - 0s 5ms/step - loss: 0.4404 - mape: 15.9415 - val\_loss: 0.5872 - val\_mape: 17.7192 - lr: 2.0000e-04

Epoch 15/50

78/78 [==============================] - 0s 6ms/step - loss: 0.4400 - mape: 15.8561 - val\_loss: 0.5744 - val\_mape: 17.7680 - lr: 2.0000e-04

25/25 [==============================] - 0s 2ms/step - loss: 0.5857 - mape: 17.9845

NN with Embeddings Mean Absolute Percentage Error: 17.984468460083008

A graph of a graph

Description automatically generated

Recurrent Neural Network (RNN) with Embeddings

[7]

28s

# Recurrent Neural Network with Embeddings and Improvements  
def create\_rnn\_model\_with\_embeddings(input\_data, embedding\_dim=8):  
    numerical\_input = Input(shape=(X\_numerical.shape[1],))  
    inputs, embeddings = create\_embedding\_layers(input\_data, categorical\_columns, embedding\_dim)  
    merged\_embeddings = Concatenate()(embeddings)  
    merged\_inputs = Concatenate()([numerical\_input, merged\_embeddings])  
    x = tf.expand\_dims(merged\_inputs, axis=1)  
    x = LSTM(128, return\_sequences=True)(x)  
    x = LSTM(64)(x)  
    x = Dense(32, activation='relu')(x)  
    output = Dense(1)(x)  
    model = Model(inputs=[numerical\_input] + inputs, outputs=output)  
    model.compile(optimizer='adam', loss='mse', metrics=[mape])  
    return model  
  
# Create and train the RNN model with early stopping and learning rate reduction  
rnn\_model\_with\_embeddings = create\_rnn\_model\_with\_embeddings(df)  
early\_stopping = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)  
reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.2, patience=5, min\_lr=0.0001)  
rnn\_history = rnn\_model\_with\_embeddings.fit(  
    [X\_train\_num] + X\_train\_cat, y\_train, epochs=50, batch\_size=32, validation\_split=0.2,  
    callbacks=[early\_stopping, reduce\_lr]  
)  
rnn\_loss, rnn\_mape = rnn\_model\_with\_embeddings.evaluate([X\_test\_num] + X\_test\_cat, y\_test)  
print(f"RNN with Embeddings Mean Absolute Percentage Error: {rnn\_mape}")

Epoch 1/50

78/78 [==============================] - 9s 35ms/step - loss: 6.5109 - mape: 55.5365 - val\_loss: 0.5723 - val\_mape: 17.5965 - lr: 0.0010

Epoch 2/50

78/78 [==============================] - 2s 22ms/step - loss: 0.5112 - mape: 17.3638 - val\_loss: 0.5827 - val\_mape: 17.5029 - lr: 0.0010

Epoch 3/50

78/78 [==============================] - 3s 44ms/step - loss: 0.5030 - mape: 17.1216 - val\_loss: 0.6489 - val\_mape: 17.7247 - lr: 0.0010

Epoch 4/50

78/78 [==============================] - 2s 27ms/step - loss: 0.4963 - mape: 16.9237 - val\_loss: 0.5750 - val\_mape: 17.6821 - lr: 0.0010

Epoch 5/50

78/78 [==============================] - 1s 12ms/step - loss: 0.5033 - mape: 17.0506 - val\_loss: 0.6629 - val\_mape: 17.8683 - lr: 0.0010

Epoch 6/50

78/78 [==============================] - 1s 13ms/step - loss: 0.4991 - mape: 16.9545 - val\_loss: 0.6724 - val\_mape: 17.9255 - lr: 0.0010

Epoch 7/50

78/78 [==============================] - 1s 12ms/step - loss: 0.4828 - mape: 16.7123 - val\_loss: 0.6210 - val\_mape: 17.7369 - lr: 2.0000e-04

Epoch 8/50

78/78 [==============================] - 1s 11ms/step - loss: 0.4817 - mape: 16.6743 - val\_loss: 0.5915 - val\_mape: 17.7024 - lr: 2.0000e-04

Epoch 9/50

78/78 [==============================] - 1s 11ms/step - loss: 0.4804 - mape: 16.7054 - val\_loss: 0.6167 - val\_mape: 17.7516 - lr: 2.0000e-04

Epoch 10/50

78/78 [==============================] - 1s 11ms/step - loss: 0.4820 - mape: 16.7048 - val\_loss: 0.6359 - val\_mape: 17.7955 - lr: 2.0000e-04

Epoch 11/50

78/78 [==============================] - 1s 11ms/step - loss: 0.4824 - mape: 16.7039 - val\_loss: 0.6099 - val\_mape: 17.7305 - lr: 2.0000e-04

25/25 [==============================] - 0s 6ms/step - loss: 0.5927 - mape: 17.6449

RNN with Embeddings Mean Absolute Percentage Error: 17.644874572753906

A graph of a graph

Description automatically generated

# Plot training and validation loss for RNN with Embeddings  
plot\_history(rnn\_history, 'RNN with Embeddings Training and Validation Loss')

Convolutional Neural Network (CNN) with Embeddings

[8]

25s

# Convolutional Neural Network with Embeddings and Improvements

def create\_cnn\_model\_with\_embeddings(input\_data, embedding\_dim=8):

    numerical\_input = Input(shape=(X\_numerical.shape[1],))

    inputs, embeddings = create\_embedding\_layers(input\_data, categorical\_columns, embedding\_dim)

    merged\_embeddings = Concatenate()(embeddings)

    merged\_inputs = Concatenate()([numerical\_input, merged\_embeddings])

    x = tf.expand\_dims(merged\_inputs, axis=2)

    x = Conv1D(64, kernel\_size=3, activation='relu', padding='same')(x)

    x = MaxPooling1D(pool\_size=2)(x)

    x = Conv1D(32, kernel\_size=3, activation='relu', padding='same')(x)

    x = MaxPooling1D(pool\_size=2)(x)

    x = Flatten()(x)

    x = Dense(32, activation='relu')(x)

    output = Dense(1)(x)

    model = Model(inputs=[numerical\_input] + inputs, outputs=output)

    model.compile(optimizer='adam', loss='mse', metrics=[mape])

    return model

# Create and train the CNN model with early stopping and learning rate reduction

cnn\_model\_with\_embeddings = create\_cnn\_model\_with\_embeddings(df)

cnn\_history = cnn\_model\_with\_embeddings.fit(

    [X\_train\_num] + X\_train\_cat, y\_train, epochs=50, batch\_size=32, validation\_split=0.2,

    callbacks=[early\_stopping, reduce\_lr]

)

cnn\_loss, cnn\_mape = cnn\_model\_with\_embeddings.evaluate([X\_test\_num] + X\_test\_cat, y\_test)

print(f"CNN with Embeddings Mean Absolute Percentage Error: {cnn\_mape}")

# Plot training and validation loss for CNN with Embeddings

plot\_history(cnn\_history, 'CNN with Embeddings Training and Validation Loss')

Epoch 1/50

78/78 [==============================] - 6s 27ms/step - loss: 2.3210 - mape: 30.4766 - val\_loss: 0.5638 - val\_mape: 18.1505 - lr: 0.0010

Epoch 2/50

78/78 [==============================] - 1s 14ms/step - loss: 0.5407 - mape: 17.5640 - val\_loss: 0.5458 - val\_mape: 18.1235 - lr: 0.0010

Epoch 3/50

78/78 [==============================] - 1s 15ms/step - loss: 0.5158 - mape: 17.2594 - val\_loss: 0.5810 - val\_mape: 17.2989 - lr: 0.0010

Epoch 4/50

78/78 [==============================] - 1s 15ms/step - loss: 0.5130 - mape: 17.1466 - val\_loss: 0.5287 - val\_mape: 17.6574 - lr: 0.0010

Epoch 5/50

78/78 [==============================] - 1s 14ms/step - loss: 0.4981 - mape: 17.0324 - val\_loss: 0.5855 - val\_mape: 17.3440 - lr: 0.0010

Epoch 6/50

78/78 [==============================] - 1s 15ms/step - loss: 0.5080 - mape: 17.0912 - val\_loss: 0.5492 - val\_mape: 17.3688 - lr: 0.0010

Epoch 7/50

78/78 [==============================] - 2s 22ms/step - loss: 0.4999 - mape: 17.0200 - val\_loss: 0.5322 - val\_mape: 17.6694 - lr: 0.0010

Epoch 8/50

78/78 [==============================] - 2s 23ms/step - loss: 0.4945 - mape: 16.9352 - val\_loss: 0.5437 - val\_mape: 18.1622 - lr: 0.0010

Epoch 9/50

78/78 [==============================] - 1s 18ms/step - loss: 0.5009 - mape: 16.9637 - val\_loss: 0.5382 - val\_mape: 17.9320 - lr: 0.0010

Epoch 10/50

78/78 [==============================] - 1s 14ms/step - loss: 0.4821 - mape: 16.7802 - val\_loss: 0.5491 - val\_mape: 17.4664 - lr: 2.0000e-04

Epoch 11/50

78/78 [==============================] - 1s 14ms/step - loss: 0.4826 - mape: 16.7623 - val\_loss: 0.5374 - val\_mape: 17.6254 - lr: 2.0000e-04

Epoch 12/50

78/78 [==============================] - 1s 14ms/step - loss: 0.4841 - mape: 16.7484 - val\_loss: 0.5448 - val\_mape: 17.5035 - lr: 2.0000e-04

Epoch 13/50

78/78 [==============================] - 1s 14ms/step - loss: 0.4785 - mape: 16.6833 - val\_loss: 0.5373 - val\_mape: 17.8592 - lr: 2.0000e-04

Epoch 14/50

78/78 [==============================] - 1s 13ms/step - loss: 0.4780 - mape: 16.6183 - val\_loss: 0.5392 - val\_mape: 17.9425 - lr: 2.0000e-04

25/25 [==============================] - 0s 4ms/step - loss: 0.5570 - mape: 18.3849

CNN with Embeddings Mean Absolute Percentage Error: 18.384910583496094

A graph of a graph

Description automatically generated

Bidirectional Recurrent Neural Network (BiRNN) with Embeddings

[9]

52s

# Bidirectional RNN with Embeddings and Improvements

def create\_birnn\_model\_with\_embeddings(input\_data, embedding\_dim=8):

    numerical\_input = Input(shape=(X\_numerical.shape[1],))

    inputs, embeddings = create\_embedding\_layers(input\_data, categorical\_columns, embedding\_dim)

    merged\_embeddings = Concatenate()(embeddings)

    merged\_inputs = Concatenate()([numerical\_input, merged\_embeddings])

    x = tf.expand\_dims(merged\_inputs, axis=1)

    x = Bidirectional(LSTM(128, return\_sequences=True))(x)

    x = Bidirectional(LSTM(64))(x)

    x = Dense(32, activation='relu')(x)

    output = Dense(1)(x)

    model = Model(inputs=[numerical\_input] + inputs, outputs=output)

    model.compile(optimizer='adam', loss='mse', metrics=[mape])

    return model

# Create and train the BiRNN model with early stopping and learning rate reduction

birnn\_model\_with\_embeddings = create\_birnn\_model\_with\_embeddings(df)

birnn\_history = birnn\_model\_with\_embeddings.fit(

    [X\_train\_num] + X\_train\_cat, y\_train, epochs=50, batch\_size=32, validation\_split=0.2,

    callbacks=[early\_stopping, reduce\_lr]

)

birnn\_loss, birnn\_mape = birnn\_model\_with\_embeddings.evaluate([X\_test\_num] + X\_test\_cat, y\_test)

print(f"BiRNN with Embeddings Mean Absolute Percentage Error: {birnn\_mape}")

# Plot training and validation loss for BiRNN with Embeddings

plot\_history(birnn\_history, 'BiRNN with Embeddings Training and Validation Loss')

Epoch 1/50

78/78 [==============================] - 13s 45ms/step - loss: 4.8589 - mape: 45.6719 - val\_loss: 0.6680 - val\_mape: 17.8866 - lr: 0.0010

Epoch 2/50

78/78 [==============================] - 2s 26ms/step - loss: 0.5144 - mape: 17.2803 - val\_loss: 0.5795 - val\_mape: 17.7173 - lr: 0.0010

Epoch 3/50

78/78 [==============================] - 2s 23ms/step - loss: 0.5077 - mape: 17.2120 - val\_loss: 0.6086 - val\_mape: 17.7182 - lr: 0.0010

Epoch 4/50

78/78 [==============================] - 1s 17ms/step - loss: 0.4995 - mape: 16.9280 - val\_loss: 0.6708 - val\_mape: 17.8399 - lr: 0.0010

Epoch 5/50

78/78 [==============================] - 1s 16ms/step - loss: 0.5018 - mape: 16.9503 - val\_loss: 0.6678 - val\_mape: 17.9324 - lr: 0.0010

Epoch 6/50

78/78 [==============================] - 1s 15ms/step - loss: 0.5048 - mape: 17.0493 - val\_loss: 0.6907 - val\_mape: 17.9676 - lr: 0.0010

Epoch 7/50

78/78 [==============================] - 1s 15ms/step - loss: 0.4962 - mape: 16.8654 - val\_loss: 0.5958 - val\_mape: 17.6876 - lr: 0.0010

Epoch 8/50

78/78 [==============================] - 1s 17ms/step - loss: 0.4810 - mape: 16.6766 - val\_loss: 0.5904 - val\_mape: 17.7415 - lr: 2.0000e-04

Epoch 9/50

78/78 [==============================] - 1s 18ms/step - loss: 0.4788 - mape: 16.6358 - val\_loss: 0.6035 - val\_mape: 17.7239 - lr: 2.0000e-04

Epoch 10/50

78/78 [==============================] - 1s 18ms/step - loss: 0.4803 - mape: 16.6820 - val\_loss: 0.6024 - val\_mape: 17.7057 - lr: 2.0000e-04

Epoch 11/50

78/78 [==============================] - 4s 48ms/step - loss: 0.4785 - mape: 16.5833 - val\_loss: 0.5967 - val\_mape: 17.6953 - lr: 2.0000e-04

Epoch 12/50

78/78 [==============================] - 2s 19ms/step - loss: 0.4798 - mape: 16.6515 - val\_loss: 0.6111 - val\_mape: 17.7359 - lr: 2.0000e-04

25/25 [==============================] - 0s 5ms/step - loss: 0.5886 - mape: 17.5401

BiRNN with Embeddings Mean Absolute Percentage Error: 17.540119171142578

A graph of a graph

Description automatically generated with medium confidence

Experiment with Pretrained DL Models

[10]

0s

#Import Necessary Libraries  
import tensorflow as tf  
from tensorflow.keras.layers import Input, Dense, Flatten, Concatenate, Embedding  
from tensorflow.keras.models import Model  
import matplotlib.pyplot as plt  
  
# Define the MAPE metric using TensorFlow operations  
def mape(y\_true, y\_pred):  
    y\_true = tf.convert\_to\_tensor(y\_true)  
    y\_pred = tf.convert\_to\_tensor(y\_pred)  
    return tf.reduce\_mean(tf.abs((y\_true - y\_pred) / y\_true)) \* 100  
  
# Define embedding input layers for categorical features  
def create\_embedding\_layers(input\_data, categorical\_columns, embedding\_dim=8):  
    inputs = []  
    embeddings = []  
    for col in categorical\_columns:  
        vocab\_size = input\_data[col].nunique()  
        input\_layer = Input(shape=(1,))  
        embedding\_layer = Embedding(input\_dim=vocab\_size, output\_dim=embedding\_dim, input\_length=1)(input\_layer)  
        flatten\_layer = Flatten()(embedding\_layer)  
        inputs.append(input\_layer)  
        embeddings.append(flatten\_layer)  
    return inputs, embeddings

Create and Train the Pretrained-Inspired Model

[11]

16s

# Pretrained-Inspired Model  
def create\_pretrained\_inspired\_model(input\_data, embedding\_dim=8):  
    numerical\_input = Input(shape=(X\_numerical.shape[1],))  
    inputs, embeddings = create\_embedding\_layers(input\_data, categorical\_columns, embedding\_dim)  
    merged\_embeddings = Concatenate()(embeddings)  
    merged\_inputs = Concatenate()([numerical\_input, merged\_embeddings])  
    x = Dense(128, activation='relu')(merged\_inputs)  
    x = Dense(64, activation='relu')(x)  
    x = Dense(32, activation='relu')(x)  
    x = Dense(16, activation='relu')(x)  
    output = Dense(1)(x)  
    model = Model(inputs=[numerical\_input] + inputs, outputs=output)  
    model.compile(optimizer='adam', loss='mse', metrics=[mape])  
    return model  
  
# Prepare inputs for the model  
numerical\_input\_data = X\_numerical  
categorical\_input\_data = [df[col].values for col in categorical\_columns]  
  
# Create and train the model  
pretrained\_model = create\_pretrained\_inspired\_model(df)  
early\_stopping = tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)  
reduce\_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val\_loss', factor=0.2, patience=5, min\_lr=0.0001)  
pretrained\_history = pretrained\_model.fit(  
    [X\_train\_num] + X\_train\_cat, y\_train, epochs=50, batch\_size=32, validation\_split=0.2,  
    callbacks=[early\_stopping, reduce\_lr]  
)  
pretrained\_loss, pretrained\_mape = pretrained\_model.evaluate([X\_test\_num] + X\_test\_cat, y\_test)  
print(f"Pretrained Inspired Model Mean Absolute Percentage Error: {pretrained\_mape}")  
  
# Plot training and validation loss for the pretrained-inspired model  
def plot\_history(history, title):  
    plt.plot(history.history['loss'], label='Training Loss')  
    plt.plot(history.history['val\_loss'], label='Validation Loss')  
    plt.title(title)  
    plt.xlabel('Epochs')  
    plt.ylabel('Loss')  
    plt.legend()  
    plt.show()  
  
plot\_history(pretrained\_history, 'Pretrained Inspired Model Training and Validation Loss')

Epoch 1/50

78/78 [==============================] - 6s 12ms/step - loss: 3.1416 - mape: 36.2485 - val\_loss: 0.7271 - val\_mape: 18.8861 - lr: 0.0010

Epoch 2/50

78/78 [==============================] - 0s 6ms/step - loss: 0.5857 - mape: 18.0133 - val\_loss: 0.6483 - val\_mape: 17.9260 - lr: 0.0010

Epoch 3/50

78/78 [==============================] - 0s 5ms/step - loss: 0.5285 - mape: 17.3264 - val\_loss: 0.7377 - val\_mape: 18.3091 - lr: 0.0010

Epoch 4/50

78/78 [==============================] - 0s 5ms/step - loss: 0.5122 - mape: 17.0933 - val\_loss: 0.5977 - val\_mape: 17.8294 - lr: 0.0010

Epoch 5/50

78/78 [==============================] - 0s 5ms/step - loss: 0.4996 - mape: 16.8266 - val\_loss: 0.6320 - val\_mape: 17.8619 - lr: 0.0010

Epoch 6/50

78/78 [==============================] - 0s 5ms/step - loss: 0.4959 - mape: 16.7694 - val\_loss: 0.6739 - val\_mape: 18.0377 - lr: 0.0010

Epoch 7/50

78/78 [==============================] - 0s 4ms/step - loss: 0.4950 - mape: 16.7565 - val\_loss: 0.8223 - val\_mape: 18.7897 - lr: 0.0010

Epoch 8/50

78/78 [==============================] - 0s 5ms/step - loss: 0.4897 - mape: 16.6993 - val\_loss: 0.6597 - val\_mape: 17.9815 - lr: 0.0010

Epoch 9/50

78/78 [==============================] - 0s 6ms/step - loss: 0.4927 - mape: 16.6539 - val\_loss: 0.6011 - val\_mape: 17.9988 - lr: 0.0010

Epoch 10/50

78/78 [==============================] - 0s 5ms/step - loss: 0.4675 - mape: 16.4818 - val\_loss: 0.6464 - val\_mape: 17.9877 - lr: 2.0000e-04

Epoch 11/50

78/78 [==============================] - 0s 6ms/step - loss: 0.4611 - mape: 16.2638 - val\_loss: 0.6508 - val\_mape: 18.0363 - lr: 2.0000e-04

Epoch 12/50

78/78 [==============================] - 0s 6ms/step - loss: 0.4574 - mape: 16.2552 - val\_loss: 0.7025 - val\_mape: 18.1547 - lr: 2.0000e-04

Epoch 13/50

78/78 [==============================] - 0s 5ms/step - loss: 0.4601 - mape: 16.2546 - val\_loss: 0.6718 - val\_mape: 18.0677 - lr: 2.0000e-04

Epoch 14/50

78/78 [==============================] - 0s 5ms/step - loss: 0.4567 - mape: 16.1662 - val\_loss: 0.7118 - val\_mape: 18.2527 - lr: 2.0000e-04

25/25 [==============================] - 0s 3ms/step - loss: 0.6296 - mape: 17.7244

Pretrained Inspired Model Mean Absolute Percentage Error: 17.72443199157715

A graph of a graph with blue line and orange line

Description automatically generated

Step 5 Evaluation of all models

Evaluation of All Models

# Store results in a dictionary

results = {

"NN with Embeddings": nn\_mape,

"RNN with Embeddings": rnn\_mape,

"CNN with Embeddings": cnn\_mape,

"BiRNN with Embeddings": birnn\_mape,

"Pretrained Inspired Model": pretrained\_mape

}

# Display results

for model\_name, mape\_value in results.items():

print(f"{model\_name} Test MAPE: {mape\_value}")

# Plot training and validation loss for each model

def plot\_history(history, title):

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title(title)

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

plot\_history(nn\_history, 'NN with Embeddings Training and Validation Loss')

plot\_history(rnn\_history, 'RNN with Embeddings Training and Validation Loss')

plot\_history(cnn\_history, 'CNN with Embeddings Training and Validation Loss')

plot\_history(birnn\_history, 'BiRNN with Embeddings Training and Validation Loss')

plot\_history(pretrained\_history, 'Pretrained Inspired Model Training and Validation Loss')

NN with Embeddings Test MAPE: 17.984468460083008

RNN with Embeddings Test MAPE: 17.644874572753906

CNN with Embeddings Test MAPE: 18.384910583496094

BiRNN with Embeddings Test MAPE: 17.540119171142578

Pretrained Inspired Model Test MAPE: 17.72443199157715

A graph of a graph

Description automatically generated

A graph of a graph

Description automatically generated

A graph of a graph

Description automatically generated

A graph of a graph

Description automatically generated with medium confidence

A graph of a graph with blue line and orange line

Description automatically generated