Model Selection for Customer Satisfaction Prediction

ELCE 455: Machine Learning with Python

School of Engineering and Digital Science

Instructor: Amin Zollavarni

Completed by: Dilnaz Issayeva, Aruzhan Tuletbekova

20.11.2024

Contents

Abstract

Forecasting customer satisfaction level is essential part of today's business environment, as this factor plays a crucial role in enhancing customer loyalty, retention and hence improving company's profitability over time. This project focuses on probabilistic models of customer satisfaction where the many characteristics are being used: income, service quality, product quality, age, and purchase frequency etc. By accessing the data with such features, four different machine learning techniques, namely, K-Nearest Neighbor, Random Forest, Lasso and Neural Network model will be implemented on predicting customer satisfaction. Results of this study will provide valuable insights for business by highlighting key factors that affect this target variable and identify machine learning technique with higher accuracy among selected models.

1 Introduction

Measuring customer satisfaction is crucial today specially for the successful operation of a business because it is a significant factor for success and longevity of a business. According to Smith and Kumar [1], machine learning algorithms are increasingly leveraged within the e-commerce sector to enhance customer engagement, optimize sales performance, lower operational costs, and provide tailored product or service recommendations by efficiently analyzing consumer data. The problem is that analyzing such large dataset with numerous entries which includes variety of features manually can be time-consuming and error-prone. This project's objective is to identify the most significant factors contributing to customer satisfaction. Additionally, we will conduct tests with four different machine learning models such as KNN, Random Forest, Lasso and Neural Network regressor and, by the end of the study, define the model selection with higher accuracy among chosen models. Based on this information, we will provide insights for the business regarding customer satisfaction and strategies to improve it.

2 Methodology

2.1 Data Collection

For this project, we used a publicly available dataset obtained through Kaggle, an online platform for data science competitions and datasets. The dataset, titled "Customer Feedback and Satisfaction", contains information about customer satisfaction and includes various features that can affect satisfaction metrics [2].

2.2 Library Imports and Setup

This project is focused on implementing various models and choosing the more accurate one, so it used both the **scikit-learn** library and **keras** API within Tensorflow. The full list of libraries and imports is shown in Figure 1.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import time
import random
from sklearn.model_selection import train_test_split, GridSearchCV, KFold
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Lasso
from sklearn.neural_network import MLPRegressor
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.preprocessing import StandardScaler
```

Figure 1: Imports

2.3 Data Description and Preprocessing

Before training our models, we tried to preview the data and perform certain actions to prepare our data for training and further steps. The original data look as shown in Figure 2.

```
df=pd.read_csv('satisfaction.csv')
df.head()
```

	CustomerID	Age	Gender	Country	Income	ProductQuality	ServiceQuality	PurchaseFrequency	FeedbackScore	LoyaltyLevel	SatisfactionScore
0	1	56	Male	UK	83094	5	8	5	Low	Bronze	100.0
1	2	69	Male	UK	86860	10	2	8	Medium	Gold	100.0
2	3	46	Female	USA	60173	8	10	18	Medium	Silver	100.0
3	4	32	Female	UK	73884	7	10	16	Low	Gold	100.0
4	5	60	Male	UK	97546	6	4	13	Low	Bronze	82.0

Figure 2: Initial data

The data was initially clean of duplicates and missing values. Figure 3 contains information about column types, data shape, and memory usage, which gives a broader view of the data. The "CustomerID" column has been deleted because it is a unique identifier and is not useful for modeling. Since the models used in this project are designed to work with numeric data only, columns with **object** data types should be encoded into numeric values to avoid problems when implementing models. The **Gender** column is binary, so LabelEncoder was used. The **Feedback Score** and **Loyalty Level** columns,

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38444 entries, 0 to 38443
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   CustomerID         38444 non-null  int64
 1   Age                38444 non-null  int64
 2   Gender             38444 non-null  object
 3   Country            38444 non-null  object
 4   Income             38444 non-null  int64
 5   ProductQuality     38444 non-null  int64
 6   ServiceQuality     38444 non-null  int64
 7   PurchaseFrequency  38444 non-null  int64
 8   FeedbackScore      38444 non-null  object
 9   LoyaltyLevel       38444 non-null  object
 10  SatisfactionScore  38444 non-null  float64
dtypes: float64(1), int64(6), object(4)
memory usage: 3.2+ MB
```

Figure 3: Dataset Overview

containing ordinal values, were mapped accordingly. For the **Country** column with 5 unique values, OneHotEncoder was applied, increasing the feature count. The code is given in the Figure 4.

```
le = LabelEncoder()
loyalty_map = {'Bronze': 0, 'Silver': 1, 'Gold': 2}
feedback_map = {'Low':0, 'Medium':1, 'High':2}

df['Gender'] = LabelEncoder().fit_transform(df['Gender'])
df['LoyaltyLevel'] = df['LoyaltyLevel'].map(loyalty_map)
df['FeedbackScore'] = df['FeedbackScore'].map(feedback_map)

df = pd.get_dummies(df, columns=['Country'], drop_first=True).astype(int) #dropped column for Canada to avoid multicollinearity
print("Data after Encoding:")
display(df.head())
```

```
Data after Encoding:
```

	Age	Gender	Income	ProductQuality	ServiceQuality	PurchaseFrequency	FeedbackScore	LoyaltyLevel	SatisfactionScore	Country_France
0	56	1	83094	5	8	5	0	0	100	0
1	69	1	86860	10	2	8	1	2	100	0
2	46	0	60173	8	10	18	1	1	100	0
3	32	0	73884	7	10	16	0	2	100	0
4	60	1	97546	6	4	13	0	0	82	0

Figure 4: Encoding of categorical data

As a next step, an exploratory analysis was performed. The figure of distributions (Figure 5) shows that the country columns are left-skewed, the target column is right-skewed, and the other columns have nearly uniform distributions. Therefore, after dividing the data into training and test parts, the necessary transformations should be carried out. Another important graph is the heatmap (Figure 6), which shows the correlation co-

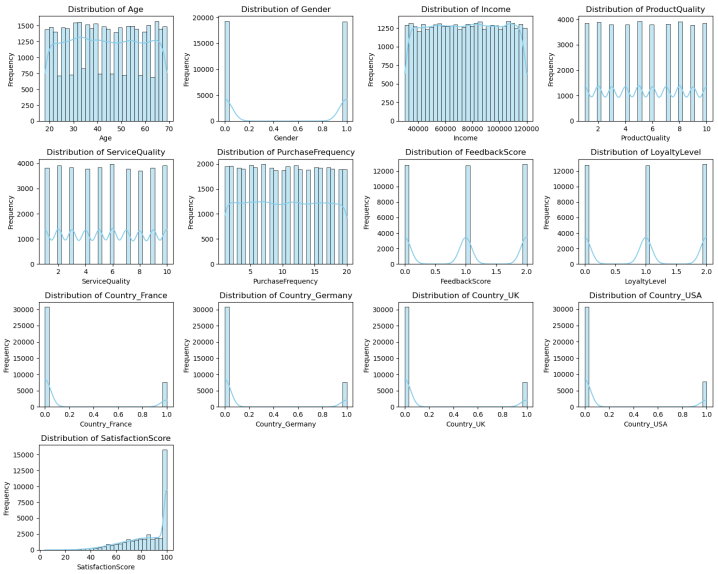

Figure 5: Distribution of all columns

efficients between different variables. This map is useful for selecting relevant features to train the model, as irrelevant features can introduce noise and degrade model performance on unseen data. The provided heatmap shows that only 5 columns—'ServiceQuality', 'ProductQuality', 'Income', 'Age', and 'PurchaseFrequency'—have a notible impact on predicting 'SatisfactionScore'. Hence, we selected these features to train our models.

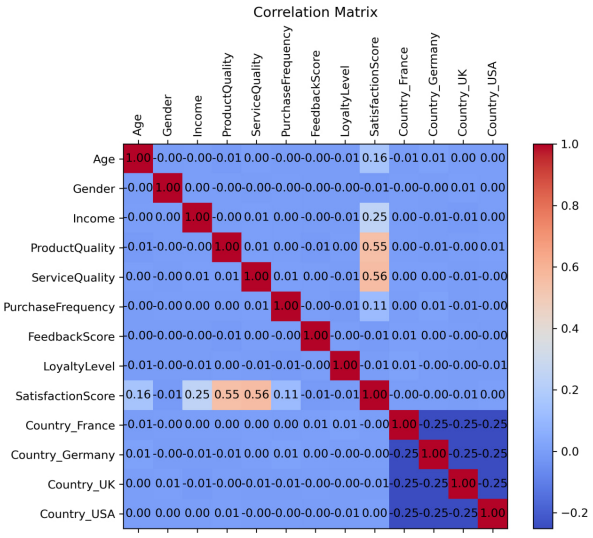

Figure 6: Correlation Matrix

2.4 Data Splitting

The train test split method was used twice to split the data into three sets: training, validation, and test sets, as shown in the Figure 7.

We have divided the data into three sets, as the project aims to select a model that will work better. That is why the validation set is important here, which is used to tune hyperparameters and select the appropriate model. This is important because it helps to evaluate how well the model performs on unseen data, which it has not encountered during training. As a result, the final dataset distribution is approximately 56.25% for training, 18.75% for validation, and 25% for testing.

```
X= df[['ServiceQuality', 'ProductQuality', 'Income', 'Age', 'PurchaseFrequency']]
y = df.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, random_state=42)
```

Figure 7: Data Splitting

2.5 Transformation of Data

As presented before, the target variable has a right-skewed distribution, which can bias the predictions of the model and affect performance. To address this, a logarithmic transformation was applied to the target variable (y), reducing the impact of extreme values. Additionally, feature scaling was applied to ensure that all features contribute equally to the model. Without scaling, features with larger values could dominate the learning process, leading to biased results. The StandardScaler() transformation was fitted on the training data, because both the validation and test sets are considered to be unseen (Figure 8).

```
if abs(y.skew()) > 1:
    y = np.log1p(y)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

Figure 8: Data Transformation

2.6 Model Definitions and Hyperparameter Grid Setup

For this project, four different models such as K-Nearest Neighbors, Random Forest, Lasso regression and Neural Network Regressor were introduced. The listed models provide a combination of traditional, ensemble-based and deep learning approaches. As a next step,

we specified the hyperparameters to be tuned for each one to enhance their performance ensuring reliability and robustness.

```
models = {
    'KNN Regressor': KNeighborsRegressor(),
    'Random Forest Regressor': RandomForestRegressor(),
    'Lasso Regression': Lasso()
}

param_grids = {
    'KNN Regressor': {
        'n_neighbors': [3, 5, 7, 10],
        'weights': ['uniform', 'distance'],
        'p': [1, 2]
    },
    'Random Forest Regressor': {
        'n_estimators': [50, 100, 200],
        'max_depth': [10, 20, None],
        'min_samples_split': [2, 5, 10],
        'max_features': ['auto', 'sqrt', 'log2', None]
    },
    'Lasso Regression': {
        'alpha': [0.01, 0.1, 1, 10]
    }
}
```

Figure 9: Models and Hyperparameters for grid search

2.7 Grid Search Setup

Among the defined models and hyperparameters, the combination that performed the best based on the validation set should be selected. To implement this, we performed a grid search for each model (Figure 10), which involves testing various combinations of hyperparameters within a specified range to determine the most optimal configuration for the given parameters and models.

For models that require scaling (for example, the KNN regressor), we use grid search based on scaled training data. Optimal hyperparameters and corresponding performance indicators (negative MSE) were recorded for each model. We will also evaluate the performance of each model in the validation set using the MSE and R^2 estimates, and selected the model that showed the best results from the proposed options based on the validation results. This model was then used for evaluation in a test set.

```python
# Grid search for each model
for model_name, model in models.items():
    print(f"Grid search for {model_name}...")

    grid_search = GridSearchCV(estimator=model,
                               param_grid=param_grids[model_name],
                               cv=cv,
                               scoring='neg_mean_squared_error',
                               n_jobs=-1,
                               refit=True)

    # Fit model on scaled data for models that require scaling
    if model_name == 'KNN Regressor':  # KNN requires scaling
        grid_search.fit(X_train_scaled, y_train)
    else:
        grid_search.fit(X_train, y_train)

    print(f"Best parameters for {model_name}: {grid_search.best_params_}")

    # Record the best score and parameters
    best_model_score = grid_search.best_score_
    best_model_params = grid_search.best_params_

    model_results[model_name] = {
        'best_params': best_model_params,
        'best_score': best_model_score
    }

    # Evaluate on validation set
    y_val_pred = grid_search.best_estimator_.predict(X_val_scaled if model_name == 'KNN Regressor' else X_val)

    mse = mean_squared_error(y_val, y_val_pred)
    r2 = r2_score(y_val, y_val_pred)

    print(f"Validation Set - {model_name}: MSE = {mse:.2f}, R² = {r2:.2f}")

    # Track the best model based on the best validation score
    if best_model_score > best_score:
        best_score = best_model_score
        best_params = best_model_params
        best_model = grid_search.best_estimator_
```

Figure 10: Grid Search Setup

3 Numerical Analysis

3.1 Grid Search Results

The results of grid search for each model are presented in Figure 11. Here are the best hyperparameters with their corresponding performance metrics (MSE and R^2) on the validation set. Equations (1) and (2) are adapted from the textbook by Zollanvari [3] , where where lower MSE indicates smaller prediction errors, and higher R^2 reflects better alignment between predicted and actual values, signifying a more accurate model.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{1}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{2}$$

```
Grid search for KNN Regressor...
Best parameters for KNN Regressor: {'n_neighbors': 10, 'p': 1, 'weights': 'uniform'}
Validation Set - KNN Regressor: MSE = 0.01, R² = 0.75
Grid search for Random Forest Regressor...
Best parameters for Random Forest Regressor: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_split': 10, 'n_estimators': 200}
Validation Set - Random Forest Regressor: MSE = 0.01, R² = 0.77
Grid search for Lasso Regression...
Best parameters for Lasso Regression: {'alpha': 0.01}
Validation Set - Lasso Regression: MSE = 0.02, R² = 0.65
```

Figure 11: Grid Search Results

Model	MSE	R^2
KNN Regressor	0.01	0.75
Random Forest Regressor	0.01	0.77
Lasso Regression	0.02	0.65

Table 1: Model Performance on Validation Set

From the Figure 11 and Table 1, it can be seen that the Random Forest Regressor performed better, achieving a higher R^2 value of 0.77 and less MSE value of 0.01, indicating better predictive accuracy compared to the other models.

3.2 Model Evaluation on Test Set

Since the results showed that the Random Forest Regressor performed better than the other models, we used it to make predictions on the test set. We then evaluated the model on the test set to check if its performance was consistent with the validation set. The results given in Figure 13 show that the model worked similarly to the validation dataset, indicating that the model can be well generalized to new invisible data.

```
print(f"\nBest Model: {best_model}")
y_test_pred = best_model.predict(X_test_scaled if isinstance(best_model, KNeighborsRegressor) else X_test)

test_mse = mean_squared_error(y_test, y_test_pred)
test_r2 = r2_score(y_test, y_test_pred)

print(f"Test Set - Best Model: MSE = {test_mse:.2f}, R² = {test_r2:.2f}")

model_results_df = pd.DataFrame(model_results).T
print("\nModel Selection Results:")
print(model_results_df)
```

Figure 12: Model Evaluation on Test Set

```
Best Model: RandomForestRegressor(max_depth=10, max_features='sqrt', min_samples_split=10,
                      n_estimators=200)
Test Set - Best Model: MSE = 0.01, R² = 0.77

Model Selection Results:
                                                          best_params  \
KNN Regressor            {'n_neighbors': 10, 'p': 1, 'weights': 'uniform'}
Random Forest Regressor  {'max_depth': 10, 'max_features': 'sqrt', 'min...
Lasso Regression                                        {'alpha': 0.01}
```

Figure 13: Evaluation Results

3.3 Neural Network Model

For the neural network model, we build it separately using the **Keras** API. Similar steps have been taken for this model too.

3.3.1 Set Random Seed for Reproducibility

Setting a random seed is important to ensure the results are reproducible. By fixing the seed, the random processes in the code (like splitting the data and initializing weights in the neural network) will generate the same results every time the code is run.

3.3.2 Building the Neural Network Model

This model consists of three layers: two hidden layers with ReLU activation functions and dropout layers to prevent overfitting, followed by one output layer for regression. The dropout layers randomly set some of the input units to 0 during training to improve generalization. Later, the model is compiled using the Adam optimizer and the standard error loss function (MSE). The Mean Absolute Error (MAE) was chosen as an evaluation metric, which measures the average magnitude of errors in predictions.

```
regression_model = keras.Sequential([
    layers.Dense(64, activation="relu", input_shape=(X_train_scaled.shape[1],)),
    layers.Dropout(0.3),
    layers.Dense(32, activation="relu"),
    layers.Dropout(0.3),
    layers.Dense(1)  # Output layer for regression
])

regression_model.compile(optimizer="adam", loss="mse", metrics=["mae"])
```

Figure 14: Neural Network Model

3.3.3 Defining Callbacks and Training Model

As a next step, we have defined callbacks, which is an important step to overcome potential problems such as overfitting and long duration of training. Here we have defined an early stop of the callback to prevent overfitting by stopping the learning process if the validation loss does not improve over a certain number of periods. In addition, we have defined model checkpoints to keep the weights of the model with the best performance in the validation set, ensuring that the best model is used. The training process is monitored using the defined callbacks. The duration of the training is also measured and printed.

```python
# Defining callbacks for early stopping and model checkpoint
my_callbacks = [
    keras.callbacks.EarlyStopping(
        monitor="val_loss",
        patience=20,
        restore_best_weights=True
    ),
    keras.callbacks.ModelCheckpoint(
        filepath="best_regression_model.keras",
        monitor="val_loss",
        save_best_only=True,
        verbose=1
    )
]

# Training the model
start = time.time()
history = regression_model.fit(
    x=X_train_scaled,
    y=y_train,
    batch_size=32,
    epochs=200,
    validation_data=(X_val_scaled, y_val),
    callbacks=my_callbacks
)
end = time.time()
training_duration = end - start
print("Training duration = {:.3f} seconds".format(training_duration))
print(history.history.keys())
print(regression_model.summary())
```

Figure 15: Defining Callback and Training Model

The results presented in Figure 16 show that the training process stopped after 149 epochs due to the early stopping callback, indicating that the validation loss did not improve for 20 consecutive epochs. The final validation loss was approximately 0.0119, while the training loss was slightly lower at 0.0145, showing no signs of overfitting.

As a result, we plotted the graph showing both the training and validation performance of the neural network model over the training epochs. Two plots are presented in Figure 17: the left plot illustrates the MSE loss, while the right plot depicts the MAE. It can be observed that both the training and validation losses decrease over time, indicating that the model is learning effectively during training. At later epochs, the curves stabilize and converge with minimal differences between the two losses, suggesting no significant overfitting. The similarity between the two graphs further confirms the model's consistent learning and generalization capabilities.

Using the validation set, we assessed the model's performance at each epoch. With the combination of the early stopping callback and the model checkpoint, we were able to save the version of the model that achieved the lowest validation loss, representing the potential best version of the model. This best-performing model was restored as our

```
657/676 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0145 - mae: 0.0833
Epoch 140: val_loss did not improve from 0.01192
676/676 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 0.0145 - mae: 0.0833 - val_loss: 0.0120 - val_mae: 0.0749
Epoch 141/200
660/676 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0146 - mae: 0.0830
Epoch 141: val_loss did not improve from 0.01192
676/676 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0146 - mae: 0.0830 - val_loss: 0.0120 - val_mae: 0.0733
Epoch 142/200
655/676 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0145 - mae: 0.0832
Epoch 142: val_loss did not improve from 0.01192
676/676 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - loss: 0.0145 - mae: 0.0832 - val_loss: 0.0120 - val_mae: 0.0738
Epoch 143/200
650/676 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0144 - mae: 0.0833
Epoch 143: val_loss did not improve from 0.01192
676/676 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0145 - mae: 0.0833 - val_loss: 0.0120 - val_mae: 0.0733
Epoch 144/200
646/676 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0143 - mae: 0.0826
Epoch 144: val_loss did not improve from 0.01192
676/676 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.0143 - mae: 0.0826 - val_loss: 0.0119 - val_mae: 0.0723
Epoch 145/200
655/676 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0147 - mae: 0.0830
Epoch 145: val_loss did not improve from 0.01192
676/676 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.0147 - mae: 0.0830 - val_loss: 0.0120 - val_mae: 0.0739
Epoch 146/200
656/676 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0146 - mae: 0.0834
Epoch 146: val_loss did not improve from 0.01192
676/676 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 0.0146 - mae: 0.0834 - val_loss: 0.0122 - val_mae: 0.0766
Epoch 147/200
650/676 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0146 - mae: 0.0837
Epoch 147: val_loss did not improve from 0.01192
676/676 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0146 - mae: 0.0837 - val_loss: 0.0121 - val_mae: 0.0740
Epoch 148/200
660/676 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0148 - mae: 0.0832
Epoch 148: val_loss did not improve from 0.01192
676/676 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0148 - mae: 0.0832 - val_loss: 0.0119 - val_mae: 0.0735
Epoch 149/200
668/676 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0146 - mae: 0.0830
Epoch 149: val_loss did not improve from 0.01192
676/676 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - loss: 0.0146 - mae: 0.0830 - val_loss: 0.0121 - val_mae: 0.0728
Training duration = 317.134 seconds
dict_keys(['loss', 'mae', 'val_loss', 'val_mae'])
```

Figure 16: Results of Neural Network Model Training

final model. We then used it to make predictions on the unseen test set and evaluated its performance. The results showed an MSE of 0.01 and an R^2 of 0.78, which is slightly better than the most accurate model we selected during the initial comparison of different algorithms.

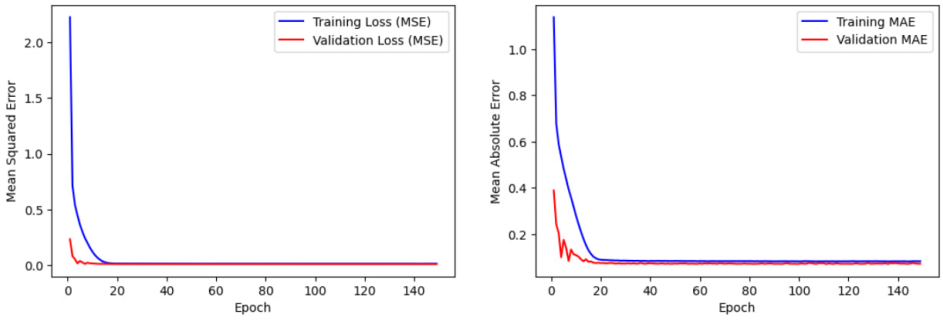

Figure 17: Training and Validation Loss Plots

4 Discussion

Based on the correlation heatmap, features that contribute to customer satisfaction estimation are ServiceQuality, ProductQuality, Income, Age, and PurchaseFrequency. These attributes show a high percentage of association with the target variable.

4.1 Neural Network model vs Random Forest

Analyzing numerical results from previous section, it is evident that Neural Network model outperformed other selected models such as K-Nearest Neighbor, Random Forest and Lasso Regressor, by demonstrating the lowest MSE of 0.01 and the highest R^2 of 0.78. In terms of accuracy Neural Network Model is followed by Random Forest, as the latter showed slightly lower R^2 of 0.77. According to Asif et al. [4], Random Forest model also provided high accuracy predictions for similar e-commerce customer satisfaction studies, as it efficiently handles large and complex datasets, captures intricate patterns, and provides robust predictions.

An exceptional high accuracy of the NN regressor could be explained by its ability to capture complex and nonlinear relationships in dataset. Unlike Random Forest, which splits data based on fixed thresholds, Neural Network model uses activation functions to identify and model intricate relationships. This allows NNs to capture details and connections that Random Forest might miss. Additionally, this model avoids overfitting by applying iterative techniques like dropout layers and early stopping.

4.2 Lasso and KNN regressor

The KNN regressor had the least accuracy in this study, but the result was better than that of the Lasso model. There were drawnbacks due to moderate size of the dataset and high dimensionality due to one-hot encoding of features i.e., ordinal categoricals like Country. Even though feature scaling seems to improve the progression of the model by a widening of the boundaries and range of variance, the higher dimensionality led to the problem of more distortion with the result that individual neighbours could no longer be easily discernible. Subsequently, one shortcoming of KNN is that it is incapable of making use of inherent feature importance or relations in the course of training, unlike other models, for that matter, such as Random Forest or Neural Networks, that are capable of identifying such structures better.

Out of all models, absolute lowest accuracy score was observed from Lasso Regression, which could be due to its linear assumption. The results for this model showed noticeable lower R^2 and higher MSE of 0.02. Lasso has the property of assuming a straight line between the features and the target variable happily enough. But as seen from the

exploratory data analysis and the heatmap, the current dataset had strong non-linear relationship with the target variable, namely the SatisfactionScore. Lasso have its uses in choosing features and regularization; however, this dataset does not contain much feature noise and is not very high in dimensionality; hence, it does not show much advantage in Lasso penalty.Thus, Lasso's attempt to shrink the model meant that some of the important non-interaction terms between some identified features were missed, and this might led to relatively lower R^2 score.

4.3 Suggestions for Future Improvements

Despite the fact that four different types of models were selected to predict customer satisfaction level, the higher accuracy for prediction can be achieved by involving more machine learning models such as AdaBoost/XGBoost. Additionally, since no feature engineering was compeleted on this project advanced feature engineering strategies like interaction terms, polynomial features could lead to higher predictive value.

5 Conclusion

In conclusion, the focus of the conducted research was to develop machine learning models for determining customer satisfaction levels taking into consideration income, service quality, product quality, age and purchase frequency as primary parameters. In this project, we compared and analyzed the performance of four models, namely KNN, Random Forest, Lasso Regression, and Neural Network Regressor by choosing MSE and R^2 as a primary evaluation metrics. The findings showed that, Neural Network Regressor was the best mode of the four models tested since it identified the relationships in the data and produced the highest accuracy. Random Forest also yield high accuracy, however, it is marginally less accurate than Neural Network model because it lacks ability to capture micro nonlinear relationship. Among the weak performances, models such as Lasso Regression and KNN were identified, and in case of datasets with interactions and transformed data, the role of model selection was noted.

It's noteworthy that this study highlights the role of machine learning when it comes to defining important factors that make key to customer satisfaction and enhancing predictive capability as well. This is where future work could extend the current contribution in feature engineering or the use of other models and fine-tuning of the neural network models to improve the results. Such advancements would give additional insights and enhanced working tools that businesses could use for acquiring even better perception and managing customer experiences.

References

[1] M. Asif, J. A. Grönlund, and M. H. Joensuu, "A comprehensive machine learning model to predict customer satisfaction and enhance business performance," *Journal of Retailing and Consumer Services*, vol. 78, p. 103865, 2024.

[2] J. Paliwal, "Customer feedback and satisfaction," Kaggle, 2023, accessed: Nov. 10, 2024. [Online]. Available: https://www.kaggle.com/datasets/jahnavipaliwal/customer-feedback-and-satisfaction

[3] A. Zollanvari, *Introduction to Machine Learning for Engineers*, 2nd ed. New York, NY: Springer, 2024.

[4] M. Asif, J. A. Grönlund, and M. H. Joensuu, "A comprehensive machine learning model to predict customer satisfaction and enhance business performance," *Journal of Retailing and Consumer Services*, vol. 78, p. 103865, 2024, accessed: Nov. 10, 2024.