E-Commerce Customer Satisfaction Score Prediction

Project Summary

Name: Nilesh Ashok Wagh

Email: nilwagh8800@gmail.com

ANN Deep Learning Classifier Model Workflow

1. Know Your Data
2. Understanding Your Variable
3. Exploratory Data Analysis
4. Data Visualization & Storytelling
5. Hypothesis Testing
6. Data Preprocessing
7. Feature Engineering
8. Data Splitting for Training the Neural Network
9. Handling Imbalance Target Variable
10. Deep Learning Model Development and Architecture
11. Hyperparameter Tuning
12. Evaluation Metrics and Model Validation
13. Insight Generation from model prediction
14. Conclusion
15. References

Please paste the GitHub Repo link.

GitHub Profile Link: - https://github.com/nilwagh8800

GitHub Repository Link: - https://github.com/nilwagh8800/E-Commerce-Customer-Satisfaction-Score-Prediction-using-Artificial-Neural-Network--DL.git

E-Commerce Customer Satisfaction Score Prediction Summary

Overview

This project focuses on predicting Customer Satisfaction (CSAT) scores using Deep Learning Artificial Neural Networks (ANN). In the context of e-commerce, understanding customer satisfaction through their interactions and feedback is crucial for enhancing service quality, customer retention, and overall business growth. By leveraging advanced neural network models, we aim to accurately forecast CSAT scores based on a myriad of interaction-related features, providing actionable insights for service improvement.

Project Background

Customer satisfaction in the e-commerce sector is a pivotal metric that influences loyalty, repeat business, and word-of-mouth marketing. Traditionally, companies have relied on direct surveys to gauge customer satisfaction, which can be time-consuming and may not always capture the full spectrum of customer experiences. With the advent of deep learning, it's now possible to predict customer satisfaction scores in real-time, offering a granular view of service performance and identifying areas for immediate improvement.

Dataset Overview

The dataset encompasses customer satisfaction scores over a one-month period on an e-commerce platform named "Shopzilla." It consists of the following features:

- Unique id: Unique identifier for each record (integer).
- Channel name: Name of the customer service channel (object/string).
- Category: Category of the interaction (object/string).
- Sub-category: Subcategory of the interaction (object/string).
- Customer Remarks: Feedback provided by the customer (object/string).
- Order id: Identifier for the order associated with the interaction (integer).
- Order date time: Date and time of the order (datetime).
- Issue reported at: Timestamp when the issue was reported (datetime).
- Issue responded: Timestamp when the issue was responded to (datetime).
- Survey response date: Date of the customer survey response (datetime).
- Customer city: City of the customer (object/string).
- Product category: Category of the product (object/string).
- Item price: Price of the item (float).
- Connected handling time: Time taken to handle the interaction (float).
- Agent name: Name of the customer service agent (object/string).
- Supervisor: Name of the supervisor (object/string).
- Manager: Name of the manager (object/string).
- Tenure Bucket: Bucket categorizing agent tenure (object/string).
- Agent Shift: Shift timing of the agent (object/string).
- CSAT Score: Customer Satisfaction (CSAT) score (integer).

Project Goal

The primary goal of this project is to develop a deep learning model that can accurately predict the CSAT scores based on customer interactions and feedback. By doing so, we aim to provide e-commerce businesses with a powerful tool to monitor and enhance customer satisfaction in real-time, thereby improving service quality and fostering customer loyalty.

ANN Deep Learning Classifier Model Workflow

1. Know Your Data

The dataset given is a dataset from the E-Commerce industry, and we have to analyze the customer's satisfaction score and the insights behind it.

Customer Satisfaction Score (CSAT) is a key performance indicator (KPI) used to gauge the level of satisfaction customers have with a company's products, services, or overall experience. In the context of e-commerce, CSAT typically measures how happy customers are with their online shopping experience, including aspects like product quality, website usability, delivery speed, and customer service.

CSAT is an essential metric for e-commerce businesses, as it directly reflects the customers' perceptions and experiences, driving both immediate and long-term business success.

The above dataset has 85907 rows and 20 columns. There are no duplicate values in the dataset, but there are missing values in a few columns such as Customer_city,Product_category,item_price,order_id,order_date_time,customer remarks and connected_handling_time.

2. Understanding Your Variable

This dataset captures detailed information about customer service interactions. Each record is uniquely identified by an integer. The data includes the name of the customer service channel, which has three unique values, and the category and sub-category of the interaction, with 12 and 57 unique values, respectively. Customer feedback is recorded in the "Customer Remarks" field.

For each interaction, there is an associated order identified by an integer, along with the order date and time. The dataset also records the timestamp when the issue was reported and when it was responded to. Additionally, the date of the customer survey response is included.

The customer's city is documented, with 1782 unique cities represented. The product category involved in the interaction, comprising 9 unique categories, and the price of the item, represented as a float, are also recorded. The "Connected handling time" field captures the time taken to handle the interaction, measured as a float.

Information about the customer service agents is detailed with their names (1371 unique names), the supervisors (40 unique names), and managers (6 unique names). Agents are also categorized by their tenure bucket and shift timing. The target variable in this dataset is the Customer Satisfaction (CSAT) score, an integer indicating the level of customer satisfaction with the service interaction.

This dataset can be utilized to analyze various aspects of customer service, such as the efficiency and effectiveness of agents, trends in customer issues, and overall customer satisfaction. This information is valuable for improving customer service strategies and enhancing customer experience.

3. Exploratory Data Analysis

Based on the provided data, we aimed to gain a clear understanding of customer satisfaction scores through graphical representations. However, it is crucial to delve deeper into the behavior of customers with varying satisfaction scores to uncover insights and hypothetical statements that might explain the reasons behind these scores. Thus, I focused on the data of customers with high satisfaction scores to identify patterns and potential reasons for their satisfaction.

Potential reasons for lower customer satisfaction scores are noted below based on the findings from the analysis:

Insights from Analysis:

Response Time: Identified that longer response times were correlated with lower customer satisfaction scores. This suggests a need for quicker response mechanisms.

Product Category: Found that certain product categories had consistently lower satisfaction scores, indicating potential issues with these products or their support processes.

Channel Name: Discovered that certain customer service channels were more effective at resolving issues satisfactorily, leading to higher CSAT scores.

Agent Tenure: Noted that agents with longer tenures tended to receive higher satisfaction scores, suggesting

that experience plays a crucial role in customer service effectiveness.

Shift Timings: Found variations in satisfaction scores based on agent shifts, with some shifts having lower scores, possibly due to higher workloads or fewer resources during those times.

Customer Feedback: Analyzed customer remarks to identify common themes and keywords associated with low satisfaction scores, providing qualitative insights into customer pain points.

4. Data Visualization & Storytelling

Chart - 1 - Pie Chart on Dependant Variable i.e., CSAT Score (Univariate)

Insights found from the chart:

Based on the chart, I observed that 59,617 customers rated the service with a CSAT Score of 5, which accounts for 69.4% of the total feedback in the dataset. Conversely, 1,283 customers were dissatisfied and gave a CSAT Score of 2, representing 1.5% of the total responses.

Additionally, 13.1% of customers gave a poor CSAT score of 1, another 13.1% rated it as 4, and 3% of customers provided a score of 3. This means nearly 15% of customers experienced poor service. Therefore, it is crucial to examine the factors contributing to this dissatisfaction.

Chart - 2 - Agent Vs. Average Response Time Percentage (Bivariate with Categorical - Numerical)

Insights found from the chart:

There are 10 agents with varying average response times.

The average response times by agent range from 2.09 to 4.09 hours. Elizabeth Rose and Donald Jordan have the shortest average response times, providing the best service to their clients through prompt action.

On the other hand, Christine Castro has the longest average response time for addressing client queries. Therefore, evaluating her performance and providing additional training is crucial to enhance the CSAT Score.

Chart - 3 - CSAT Score vs Item price (Bivariate)

Insights found from the chart:

From the bar graph, it is evident that the mean item price is highest when the CSAT score is 1 and lowest when the CSAT score is 5. This indicates an inverse correlation between item price and CSAT score, suggesting that higher item prices are generally associated with lower customer satisfaction scores.

5. Hypothesis Testing

Hypothetical Statement - 1

When the Mean Response Time is less than 2, the Customer Satisfaction Score is 5.

I have used t-Test as the statistical testing to obtain P-Value and found the result that the Null hypothesis has been rejected.

Based on the results of the one-sample t-test, the following findings can be made:

Mean Response Time:

The mean response time for customers who gave a CSAT Score of 5 is approximately 5706.44 seconds (about 1.58 hours).

T-statistic and P-value:

The t-statistic is -11.85, indicating that the observed mean response time is significantly different from the hypothesized mean of 7200 seconds (2 hours). The p-value is extremely small (2.30e-32), which is far below the significance level of 0.05.

Conclusion:

Given the p-value is much less than the significance level of 0.05, we reject the null hypothesis.

This means there is strong statistical evidence to conclude that the mean response time for customers who rated the service with a CSAT Score of 5 is significantly less than 2 hours.

Business Implication:

The significantly lower response time for customers with a high satisfaction score suggests that prompt

response times are correlated with higher customer satisfaction.

Focusing on reducing response times could be a key strategy to enhance overall customer satisfaction.

This analysis indicates that improving response times can positively impact customer satisfaction scores, supporting efforts to maintain or enhance quick response rates in customer service operations.

Hypothetical Statement - 2

When the price of an item above 5660, does it result in customer satisfaction scores to go below 3

Findings Interpretation

Fail to Reject Null Hypothesis: The p-value is greater than 0.05, it means there is not enough evidence to suggest that items priced above 5660 significantly affect customer satisfaction scores to be below 3.

Explanation of Output

T-statistic: This value indicates how many standard deviations the sample mean is away from the hypothesized mean. A negative t-statistic would support the alternative hypothesis that the sample mean is less than the hypothesized mean.

P-value: This value tells us the probability of obtaining a result at least as extreme as the one observed, assuming the null hypothesis is true. Since we are performing a one-tailed test, the p-value is divided by 2.

Decision Rule: If the p-value is less than the significance level (0.05) and the t-statistic is negative, we reject the null hypothesis, indicating that the mean CSAT score for high-priced items is significantly less than 3.

6. Data Preprocessing

1. Handling Missing Values

We employed various missing value imputation techniques based on the nature of the features and the distribution of the data:

Order_id: As this feature is not significant for our analysis and the number of missing values is minimal, we opted to drop this column entirely.

Customer Remarks: With a substantial number of missing values (57165), we couldn't discard this feature as it holds crucial information. Instead, we replaced the NaN values with "Missing Reviews" to ensure we retain the textual data for analysis.

Categorical Column Imputation (Customer city and Product Category): Since these categorical features are vital for our analysis, we used mode imputation to fill in the missing values. Mode imputation was chosen as it replaces missing values with the most frequently occurring category, thereby preserving the distribution of the data.

Numerical Column Imputation (connected_handling_time and item_price): For connected_handling_time, which follows a normal distribution with minimal outliers, we applied mean imputation to replace missing values. Conversely, for item_price, where outliers are more prominent, median imputation was utilized to ensure robustness against outliers.

order_date_time: Mode imputation was applied to handle missing values in this feature, as it represents datetime data. Subsequently, we converted it into datetime format to extract additional temporal features like day and month.

These techniques were selected to effectively manage missing data while preserving the integrity and utility of the dataset for subsequent analysis.

2. Handling Outliers

Outlier handling is crucial for ensuring the accuracy and reliability of data analysis. The following code provides an approach to identify and treat outliers in a dataset by distinguishing between symmetric and non-symmetric distributed features.

Initially, the code converts the "CSAT Score" to a string type to exclude it from numerical operations. It then classifies features into symmetric and non-symmetric distributions based on the difference between the mean and the median of each feature. If the absolute difference is less than 0.2, the feature is considered symmetric;

otherwise, it is non-symmetric. This classification is essential for applying appropriate outlier treatment methods.

For symmetric distributed features, no specific outlier treatment is applied in this code snippet. However, for non-symmetric features, the code defines an outlier treatment function. This function calculates the upper and lower boundaries as the mean plus or minus three times the standard deviation, respectively. These boundaries are used to cap the values of the non-symmetric features, effectively restricting the data within these limits to mitigate the impact of outliers.

The outlier treatment is applied using a loop that iterates over the non-symmetric features. For each feature, values below the lower boundary are set to the lower boundary, and values above the upper boundary are set to the upper boundary. This ensures that the extreme values are handled appropriately without removing any data points.

Finally, the code visualizes the distribution of the numerical columns using strip plots to show the effect of outlier treatment. The sns.stripplot function from the seaborn library is used to generate these plots, providing a clear visual representation of the data before and after outlier treatment.

This approach ensures that the data is cleaned by treating outliers appropriately, which can significantly improve the performance of subsequent data analysis and machine learning models.

3. Categorical Encoding

To handle categorical data, the provided code uses one-hot encoding, a common technique for converting categorical variables into a format that can be provided to machine learning algorithms.

First, the code identifies the categorical columns in the dataset by excluding numerical columns. The CSAT Score is converted back to an integer type since it's a numerical target variable. Categorical columns are determined.the code applies one-hot encoding to the identified categorical columns using the pd.get_dummies function from pandas. One-hot encoding converts categorical variables into a series of binary columns, each representing a possible category.

One-hot encoding is appropriate for the given categorical features because these are likely nominal categorical variables with no inherent order or ranking among categories. This encoding ensures that the machine learning model treats each category as a distinct and separate entity without implying any ordinal relationship.

This method effectively transforms the categorical data into a numerical format suitable for machine learning algorithms, ensuring that the categorical information is accurately represented in the analysis.

7. Feature Engineering

1. Feature Manipulation & Selection

Created Some new features like
Response_Time_seconds,day_number_order_date,weekday_number_order_date,
weekday_num_response_date and day_num_response_date

For feature selection, we applied several techniques to refine our dataset and ensure the chosen features contribute meaningful information. Initially, we dropped columns with constant or quasi-constant variance as they provide minimal information. Then, we employed Pearson correlation to identify and remove highly correlated features, addressing multicollinearity.

Pearson correlation values range from -1 to 1, indicating the degree of linear relationship between two variables. Values close to 0 suggest a weak correlation, while values close to 1 or -1 indicate a strong positive or negative correlation, respectively. By creating a correlation matrix, we detected variables with high absolute correlation values, which indicated collinearity.

To further validate and refine our features, we used the Variance Inflation Factor (VIF). VIF assesses the extent of multicollinearity by regressing each feature against all others. A VIF value between 5 and 10 suggests significant multicollinearity, warranting consideration for feature removal. We calculated VIF for each predictor

variable and systematically dropped features with VIFs greater than 8, iterating this process to minimize multicollinearity.

Through these steps, we reduced our feature set from 77 to 10, ensuring that the remaining features were informative and free from redundancy. This process involved dropping constant and quasi-constant features, visualizing correlation heatmaps, and removing highly correlated features to create a robust and efficient feature set.

2. Data Transformation

From the feature analysis, it was evident that two features did not follow a symmetric, Gaussian distribution, while the rest exhibited a symmetric curve. To address this, I applied exponential transformation to these two non-symmetric features to achieve a Gaussian distribution.

I experimented with various transformation techniques and found that exponential transformation worked best, producing no infinity values and yielding satisfactory results. Therefore, I decided to proceed with the exponential transformation, using a power of 0.25, to normalize these features effectively.

3. Data Scaling

When you are using an algorithm that assumes your features have a similar range, you should use feature scaling.

If the ranges of your features differ much then you should use feature scaling. If the range does not vary a lot like one of them is between 0 and 2 and the other one is between -1 and 0.5 then you can leave them as it's. However, you should use feature scaling if the ranges are, for example, between -2 and 2 and between -100 and 100.

Use Standardization when your data follows Gaussian distribution. Use Normalization when your data does not follow Gaussian distribution.

So, in my data the numerical columns have large data differences and the following gaussian distribution. That's why, I have used standardization using Standardscaler.

8. Data Splitting for Training the Neural Network

There are two competing concerns: with less training data, your parameter estimates have greater variance. With less testing data, your performance statistic will have greater variance. Broadly speaking you should be concerned with dividing data such that neither variance is too high, which is more to do with the absolute number of instances in each category rather than the percentage.

If you have a total of 100 instances, you're probably stuck with cross validation as no single split is going to give you satisfactory variance in your estimates. If you have 100,000 instances, it doesn't really matter whether you choose an 80:20 split or a 90:10 split (indeed you may choose to use less training data if your method is particularly computationally intensive).

You'd be surprised to find out that 80/20 is quite a commonly occurring ratio, often referred to as the Pareto principle. It's usually a safe bet if you use that ratio.

In this case the training dataset is small, that's why I have taken a 70:30 ratio.

9. Handling Imbalance Target Variable

In the realm of supervised machine learning with multiple classes, dealing with imbalanced datasets is crucial. Imbalance occurs when the number of data points for different classes is uneven. A balanced dataset typically means an equal distribution, such as 50% for each class in a binary classification. Minor imbalances, like a 60:40 ratio, generally do not pose significant issues. However, substantial imbalances, such as 90% for one class and 10% for another, can lead to biased models and ineffective performance measures.

In our dataset, the class distribution is imbalanced at an 85:15 ratio. This imbalance likely causes the model to be biased towards the majority class, predicting it more frequently. To mitigate this issue, it is essential to balance the dataset before model training. I employed the Synthetic Minority Over-sampling Technique (SMOTE) to address the 85:15 imbalance.

SMOTE is a machine learning technique designed to handle imbalanced datasets. Imbalanced data is common and many machine learning algorithms struggle with it, so techniques like SMOTE are used to enhance

performance. SMOTE works by generating synthetic samples of the minority class, rather than simply duplicating existing ones or undersampling the majority class. This approach creates new data points by interpolating between existing minority samples, thus increasing the minority class without creating duplicates.

The advantage of SMOTE is that it produces synthetic data points that are slightly different from the original ones, providing a more sophisticated oversampling method. This reduces the risk of overfitting and improves model performance. Due to these benefits, I utilized SMOTE to balance the dataset, ensuring a more robust and unbiased model training process.

10. Deep Learning Model Development and Architecture

1. Define the ANN Model & Architecture

Classification

- **Load Dataset**
- **Create Keras Classifier Neural Network**
- **Wrap Keras Model into Scikeras KerasClassifier**
- **Train Model**
- **Make Predictions**
- **Evaluate Model Performance**

Scikit-learn is the most renowned machine learning library within the Python community, widely used by developers globally. It offers implementations of most machine learning algorithms through a straightforward API. This simplicity is a significant reason for its popularity, allowing users to train models, make predictions, and evaluate datasets with just a single function call each. This ease of use has made scikit-learn a widely accepted library, as it eliminates the need for a steep learning curve.

However, modern machine learning challenges like object detection, image classification, and speech recognition are often too complex for the basic algorithms provided by scikit-learn. These tasks require advanced neural networks such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs). Keras is a well-known library for building such complex neural networks. Like scikit-learn, Keras is widely adopted due to its user-friendly API, which simplifies the creation of deep learning models despite requiring some learning to master.

We have used Scikeras, a library that combines the strengths of both scikit-learn and Keras. Sci Keras allows developers to wrap Keras models within classes that mimic the scikit-learn API. Consequently, these wrapped instances can be used just like scikit-learn model instances, enabling methods such as fit(), predict(), and score() to be called on them. In essence, Scikeras lets users employ Keras models as if they were scikit-learn models. We will demonstrate the Sci Keras API through simple examples using toy datasets from scikit-learn.

Create Keras Classifier Neural Network

I have created a neural network for classification using Keras. This network will be used for solving our classification task. The network comprises multiple layers, each designed to handle the complexity of the task.

We start by defining a dropout rate of 0.5 to help prevent overfitting. The neural network model is constructed with several Dense layers, BatchNormalization, and Dropout layers to enhance performance and generalization.

The input layer of the network has 3296 units, followed by three hidden layers with 128, 64, and 32 units, respectively, each using the ReLU activation function and regularized with L2. BatchNormalization and Dropout layers are added after each Dense layer to stabilize and regularize the network. The final layer has units equal to the number of classes (num_classes) and uses the softmax activation function for multi-class classification.

2. Wrap Keras Model into Scikeras KerasClassifier

We wrap the Keras neural network we created in the previous step into a Scikeras KerasClassifier. The KerasClassifier provides an easy-to-use API for classification tasks, similar to other estimators in scikit-learn.

The KerasClassifier class constructor takes several parameters:

- model: The Keras model instance.

- optimizer: The optimizer to use, specified as a string (e.g., "adam") or an instance from the keras.optimizers module. The default is "rmsprop".
- loss: The loss function to use, specified as a string (e.g., "categorical_crossentropy") or an instance from the keras.losses module.
- metrics: A list of strings specifying metrics to evaluate on each epoch.
- epochs: The number of passes through the training data.
- batch_size: The batch size to use during training.
- callbacks: A list of callables to be executed at various stages of training (e.g., start/end of an epoch, start/end of a batch).
- validation_split: A float in the range 0-1 to divide the training data into training and validation sets based on the specified proportion.
- warm_start: A boolean indicating whether to perform a warm start. If False (default), the neural network weights are reinitialized each time fit() is called. If True, training starts with the weights from the last call to fit().
- random_state: An integer to ensure reproducibility.

To wrap our Keras classifier into an instance of KerasClassifier. We use the Adam optimizer and categorical cross-entropy loss function. We set epochs to 30 for 30 passes through the data, and validation_split to 0.1 to use 10% of the training data for validation. Additionally, we set warm_start to True to continue training without reinitializing weights and specify a random_state for reproducibility.

3. Initialize StratifiedKFold Cross Validation (no. of folds=3)

To ensure robust evaluation of our classification model, we use StratifiedKFold cross-validation. This technique is particularly useful when dealing with imbalanced datasets as it maintains the class distribution across the different folds. By doing so, it ensures that each fold is representative of the overall class distribution, leading to more reliable and unbiased performance metrics.

In our setup, we define the number of folds (n_folds) to be 3. This means the dataset will be split into three parts, or folds. Each fold will be used once as a validation set while the remaining two folds will be used for training. This process is repeated three times (once for each fold as the validation set), and the performance metrics are averaged to give a more stable estimate of the model's performance.

We initialize the StratifiedKFold cross-validator with the specified number of splits (n_splits=3). The shuffle=True parameter ensures that the data is shuffled before splitting into folds, adding an additional layer of randomness to help prevent any biases due to the order of the data. The random_state=42 parameter is set to ensure reproducibility of the results.

4. Performing 3-fold cross validation and training the ANN deep learning model

To ensure a comprehensive evaluation of our artificial neural network (ANN) model, we performed 3-fold cross-validation. This approach splits the dataset into three parts, using each part once as a validation set while the remaining parts are used for training. The results from each fold are averaged to provide a robust estimate of the model's performance.

We initialized lists to store the training and testing accuracies for each fold, as well as the training history for later visualization. The following steps were performed in each fold:

1. Data Splitting: The training data was split into training and validation sets using StratifiedKFold. This maintains the class distribution across the folds.
2. Model Training: The model was trained using the training fold. We used two callbacks:
    - EarlyStopping: Monitors the validation loss and stops training if the model's performance does not improve for 5 consecutive epochs, restoring the best weights observed during training.
    - LearningRateScheduler: Adjusts the learning rate according to a predefined schedule to improve training efficiency.
3. Model Evaluation: After training, the model's performance was evaluated on both the training and validation sets. The score method provided the accuracy, which was stored in the respective lists.
4. Metric Calculation: The predictions on the training and validation sets were converted to class labels, and the accuracy scores were calculated using accuracy_score from scikit-learn. Additionally, a detailed classification report was generated using classification_report.

11. Hyperparameter Tuning

To optimize our artificial neural network (ANN) model's performance, we employed Grid Search, a systematic approach to tuning hyperparameters. This technique exhaustively searches through a specified parameter grid and evaluates each combination using cross-validation to identify the optimal hyperparameters.

In the provided code snippet, we utilized the GridSearchCV class from scikit-learn. Here's how it works:

1. **Define Parameter Grid:** We defined a parameter grid params containing the hyperparameters we want to tune. In this case, we focused on two hyperparameters:
    - batch_size: The number of samples used in each batch during training.
    - optimizer__learning_rate: The learning rate of the optimizer used for training the neural network.
2. **Instantiate Grid Search:** We created an instance of GridSearchCV by passing the classifier (scikeras_classifier), parameter grid (params), and the scoring metric ('accuracy'). The scoring metric determines how the performance of each parameter combination is evaluated.
3. **Fit Grid Search:** We called the fit method on the GridSearchCV object, passing the training data (X_train, y_train). Grid Search then exhaustively searches through all possible combinations of hyperparameters specified in the parameter grid.
4. **Evaluation:** During the fitting process, Grid Search performs cross-validation internally to evaluate each parameter combination's performance. It splits the training data into multiple folds, trains the model on each fold, and evaluates its performance on the validation set. The scoring metric specified ('accuracy') is used to assess model performance.
5. **Best Parameters:** After fitting, we can access the best combination of hyperparameters found by Grid Search using the best_params_ attribute.
6. **Best Model:** Additionally, the best model obtained during the search can be accessed using the best_estimator_ attribute. This model is trained on the entire training dataset with the optimal hyperparameters.

By leveraging Grid Search, we systematically explored different hyperparameter combinations, allowing us to identify the configuration that maximizes the model's accuracy on unseen data. This process enhances the model's performance and ensures robustness across various datasets and tasks.

12. Evaluation Metrics and Model Validation

To assess the performance of our trained model and ensure its effectiveness in making accurate predictions, we employ various evaluation metrics. We utilize the accuracy score, a commonly used metric for classification tasks, to evaluate our model's performance on both the training and test datasets.

1. Mean Train Accuracy: The mean train accuracy represents the average accuracy score across multiple iterations of the cross-validation process. In our evaluation, the mean train accuracy is approximately 0.8602, indicating that, on average, the model correctly classifies around 86.02% of instances in the training data.
2. Mean Test Accuracy: Similarly, the mean test accuracy denotes the average accuracy score across multiple iterations of the cross-validation process on the test data. With a mean test accuracy of approximately 0.8383, our model achieves an average accuracy rate of 83.83% on unseen test instances.

By comparing the train and test accuracies and considering the mean values obtained through cross-validation, we can assess whether the model suffers from overfitting (high train accuracy but low test accuracy) or underfitting (low train and test accuracy). Ideally, we aim for a model with high train and test accuracies, indicating a good balance between learning from the training data and generalizing to unseen data.

These evaluation metrics play a crucial role in validating the model's performance and guiding further optimizations to enhance its effectiveness in real-world applications.

13. Insight Generation from model prediction

The performance of our deep learning model in predicting CSAT scores, which range from 1 to 5. Here are some key observations:

1. Overall Accuracy: The model achieves a satisfactory overall accuracy of approximately 85% on both the training and test datasets. This indicates that the model effectively learns patterns from the data and generalizes well to unseen instances.

2. Class-wise Performance: The precision, recall, and F1-score metrics for each CSAT score class provide insights into the model's performance for individual classes. Notably:
    - Class 0 (CSAT score 1): The model demonstrates good precision (92%) but lower recall (64%), suggesting that while it correctly identifies instances with CSAT score 1, it may miss some relevant instances.
    - Class 1 (CSAT score 2): The model achieves high precision (99%) and recall (98%), indicating excellent performance in identifying instances with CSAT score 2.
    - Class 2 (CSAT score 3): Similar to class 1, the model shows high precision (97%) and recall (94%), reflecting its effectiveness in predicting instances with CSAT score 3.
    - Class 3 (CSAT score 4): The model exhibits decent precision (82%) and recall (70%) for class 3, indicating moderate performance in identifying instances with CSAT score 4.
    - Class 4 (CSAT score 5): The model demonstrates relatively lower precision (65%) but higher recall (98%) for class 4, implying that while it correctly identifies most instances with CSAT score 5, it may also misclassify some instances from other classes as class 4.

3. Macro and Weighted Averages: The macro-average F1-score (0.85) and weighted-average F1-score (0.85) suggest balanced performance across all classes, considering both class distribution and predictive accuracy.

Overall, the classification report provides valuable insights into the model's strengths and weaknesses in predicting CSAT scores. The model performs exceptionally well in identifying instances with CSAT scores 2 and 3, while its performance is relatively moderate for scores 1, 4, and 5. These insights can guide further improvements in model training and optimization to enhance its accuracy and robustness in predicting CSAT scores.

14. Conclusion

- **Data Overview:** The dataset comprises records from the e-commerce industry, focusing on customer service interactions and CSAT scores. It contains 85907 rows and 20 columns, with missing values in several columns such as Customer_city, Product_category, and item_price.
- **CSAT Importance:** CSAT is a crucial KPI for e-commerce businesses, reflecting customer satisfaction with products, services, and overall experience. Understanding CSAT is vital for driving business success.
- **Variable Insights:** The dataset captures detailed information about customer service interactions, including customer feedback, order details, agent information, and timestamps. Understanding these variables provides valuable insights into customer satisfaction drivers.
- **Exploratory Data Analysis (EDA):** EDA aims to gain insights into customer satisfaction patterns. Factors like response time, product category, channel effectiveness, agent tenure, shift timings, and customer feedback are analyzed to uncover potential reasons for CSAT scores.
- **Response Time Impact:** Longer response times correlate with lower CSAT scores, indicating the need for quicker response mechanisms to improve customer satisfaction.
- **Agent Experience:** Agents with longer tenures tend to receive higher CSAT scores, highlighting the importance of experience in delivering satisfactory customer service.
- **Shift Timings Influence:** CSAT scores vary based on agent shift timings, indicating potential workload or resource issues during specific shifts that need attention.
- **Customer Satisfaction Analysis:** The analysis of CSAT scores reveals that a significant portion of customers (69.4%) rated the service with a score of 5, indicating high satisfaction. However, there is also a notable proportion (15%) of customers who experienced poor service, warranting further investigation into the factors contributing to dissatisfaction.
- **CSAT Score vs. Item Price:** A negative correlation between item price and CSAT score suggests that higher-priced items are associated with lower customer satisfaction. This finding underscores the importance of pricing strategies in maintaining high CSAT scores.

- **Response Time and CSAT Score:** Statistical analysis indicates that a mean response time of less than 2 hours is significantly correlated with higher CSAT scores. This underscores the importance of prompt response times in enhancing customer satisfaction.
- **Price Impact on CSAT Score:** Hypothesis testing suggests that items priced above a certain threshold do not significantly affect CSAT scores to go below 3. This finding provides insights into pricing strategies and their impact on customer satisfaction.
- **Data Preprocessing Techniques:** Various techniques such as handling missing values, outlier detection, and categorical encoding were employed to ensure data quality and prepare it for analysis.
- **Feature Engineering:** Feature manipulation, selection, and transformation techniques were utilized to create informative features and enhance the predictive power of the model.
- **Handling Imbalance in Target Variable:** The Synthetic Minority Over-sampling Technique (SMOTE) was applied to address the imbalanced class distribution, ensuring robust model training.
- **Deep Learning Model Development:** The development of a deep learning model using a neural network architecture, wrapped into a KerasClassifier, demonstrated promising performance in predicting CSAT scores, with an overall accuracy of approximately 85%.

15. References

1. https://coderzcolumn.com/tutorials/artificial-intelligence/scikeras-give-scikit-learn-like-api-to-your-keras-networks#5

2. https://github.com/keras-team/tf-keras/issues/42

3. https://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learning-library/

4. https://towardsdatascience.com/build-your-first-deep-learning-classifier-using-tensorflow-dog-breed-example-964ed0689430

5. https://machinelearningmastery.com/use-keras-deep-learning-models-scikit-learn-python/