Predicting Online Shopper Intention with K-Nearest Neighbors Algorithm and Hyperparameter Optimization (August 2023)

Kaustuv Karki1, Undergraduate, IOE, Nikhil Pradhan2, Undergraduate, IOE

1Institute of Engineering Thapathali Campus, Tribhuvan University, Kathmandu, Nepal

2Institute of Engineering Thapathali Campus, Tribhuvan University, Kathmandu, Nepal

Corresponding author: Kaustuv Karki (e-mail: [karkikaustuv@gmail.com](mailto:karkikaustuv@gmail.com)),

Nikhil Pradhan (e-mail: [nikhilpradhan20b@gmail.com](mailto:nikhilpradhan20b@gmail.com))

ABSTRACT In the era of booming e-commerce, understanding online user behavior and purchase intent is crucial yet challenging. This research aims to evaluate the effectiveness of the K-Nearest Neighbors (KNN) algorithm, a simple supervised learning technique, in accurately predicting online shopper intent. The report employs the Online Shopper Intention dataset, which contains user session data. Multiple iterations of KNN classification are performed using scikit-learn, with key hyperparameters like n-neighbors. Through rigorous analysis of the results, this research provides insights into KNN's ability to discern between shoppers exhibiting buying behavior versus just browsing. The findings reveal the optimal configuration of KNN hyperparameters that yields high prediction accuracy on this dataset. By demonstrating KNN's potential for modeling online user intent, this study contributes to e-commerce platforms' ability to gain actionable insights from customer data. The results serve as a valuable reference for practitioners seeking efficient techniques for understanding online shopper behavior using session data. As online commerce expands, this research enables businesses to deploy targeted interventions to improve conversion rates.

INDEX TERMS Naïve Bayes, K-Nearest Neighbors (KNN), Online shopper intention prediction

1. INTRODUCTION

With the rapid expansion of e-commerce, understanding online user behavior has become crucial yet challenging for businesses seeking to improve customer experience and conversion rates. Accurately predicting shopper intent and differentiating high-intent users from those just browsing can enable targeted interventions. However, the large volumes of clickstream and session data generated makes manual analysis infeasible. Considering this, machine learning techniques like K-Nearest Neighbors (KNN) are gaining increasing interest for their ability to efficiently model shopper behavior from such data.

KNN is a simple supervised learning algorithm that predicts outcomes for new data points based on similarity to neighboring points from the training set. This study aims to evaluate the use of KNN classifiers for predicting online shopper purchase intent on a benchmark dataset. The research implements KNN models from scratch in Python without reliance on external libraries, in addition to leveraging scikit-learns KNN.

The dataset comprises user session records with features like number of pages visited, duration, etc. Text preprocessing like normalization was applied before model training. The KNN models classified sessions as either exhibiting buying behavior or not. Performance was analyzed using metrics including accuracy, precision, and recall. In addition, hyperparameters like n-neighbors and weights were tuned to identify optimal configurations.

This research provides valuable insights into developing KNN classifiers to discern online shopper intent from clickstream data. The findings demonstrate the potential of KNN models for automated prediction of user behavior on e-commerce platforms.

1. METHODOLOGY

A. BRIRF THEORY

Nearest neighbors are considered one of the simple machine learning algorithms that are used mostly for classification tasks. They can also be used for regression tasks. Though they are not used as abundantly as in the past, they are regarded as benchmark for more sophisticated models. The fundamental idea behind KNN involves identifying the 'k' closest data points to a given query point and making predictions based on the majority class (in classification) or averaging (in regression) of their labels. KNN falls under the category of lazy learning algorithms, distinguished by their minimal training phase. Instead of deriving an explicit model from the training data, KNN merely stores the training instances along with their corresponding class labels. The true processing occurs during prediction, as the algorithm dynamically computes distances between the query point and the training instances, enabling adaptability to various data distributions. In contrast to eager learning algorithms that build global models, KNN adopts a local approximation strategy. It eschews constructing a single comprehensive function to capture the underlying pattern in the entire dataset. Instead, it approximates the decision boundaries by considering the relationships between query points and their nearby neighbors. This approach is particularly effective when dealing with intricate and non-linear data distributions.

Nearest Neighbor Algorithm Steps:

1. Training Algorithm:

Iterate through each data point (x[i]) in the n-dimensional training dataset D.

Store the training example (x[i], f(x[i])) where x[i] represents the ith data point and f(x[i]) is its corresponding class label.

1. Prediction Algorithm:

Initialize variables: closest point = None, closest distance = ∞.

For each data point (x[i]) in the training dataset:

Calculate the distance (current distance) between the query point (x[q]) and the current training point (x[i]).

If the current distance is less than the closest distance:

Update closest distance with the current distance.

Update closest point as x[i].

The majority class label of the k closest points determines the predicted class (in classification), or the average of their labels is the regression prediction.

1. Distance Metrics:

The choice of distance metric significantly influences KNN's performance.

Common metrics include:

1. Euclidean distance: Measures straight-line distance between points.
2. Manhattan Distance: Measures distance along grid lines (city-block distance).
3. Minkowski Distance: Generalizes Euclidean and Manhattan distances.
4. Decision Boundary and Voronoi Diagram:

The concept of a decision boundary is pivotal in KNN. When comparing two points, the boundary is the line equidistant from both points. Points on this boundary are treated as being equally likely to belong to either class. Extending this notion to the entire training set, the decision boundary forms a network of connected convex polyhedral. Points within a polyhedron are closer to the training points inside it and are predicted based on them. The convex polyhedron, when visualized in a 2D plane, corresponds to a Voronoi diagram, revealing regions where specific training instances have the most influence.

KNN holds a strong position in the field of machine learning due to its various advantages. One key benefit is its simplicity and straightforward implementation. KNN doesn't need assumptions about data distributions or complex parameter adjustments, making it a good starting point for beginners and a reliable benchmark for advanced models. Its ease of understanding also promotes interpretability, allowing users to grasp predictions based on the actual data points used for decisions. A notable advantage is KNN's robustness to outliers, as it focuses on local relationships. It tends to perform better with larger datasets, as the risk of overfitting decreases. KNN is versatile across different problem types, effectively managing non-linear data patterns and intricate decision boundaries through its nearest neighbor-based approach.

However, KNN has its share of downsides. Particularly, it's computationally demanding during prediction because it calculates distances between the query point and all training instances. This inefficiency becomes problematic with larger datasets, limiting real-time applications. Additionally, KNN's accuracy hinges on choosing the right k (number of neighbors) and distance metric, making it sensitive and potentially affecting its performance. The algorithm is also vulnerable to noise and irrelevant features, which can lead to inaccurate predictions.

Time complexity of brute force KNN is O(n\*k\*d) where n = number of trainings datapoints, d = dimension of data and k = number of nearest neighbors. This can be reduced to O(k\*log(n)) by implementing k-d tree. K-d tree is created when we provide a training data point unlike brute force KNN that does not perform any training operation. This allows the KNN algorithm to calculate nearest neighbors based on the K-d which significantly reduces its time complexity.

B. SYSTEM BLOCK DIAGRAM

From the Figure 1 we can see the following things:

The dataset is divided into two sets: the training set, which is used to train the Naive Bayes classifier on labeled data, and the test set, which is used to evaluate the model's performance on unseen data. This separation ensures that the model does not memorize the training data but instead learns to generalize well to new and unseen tweets. By training on a separate portion of the data and testing on another, we can assess how well the Naive Bayes classifier can classify sentiments in tweets that it has not seen before.

The vocabulary represents the collection of all distinct words that occur in the training data. Each unique word in the vocabulary becomes a feature used by the Naive Bayes classifier to make sentiment predictions. By analyzing the vocabulary, we gain insights into the variety and diversity of language expressions present in the tweets. The size of the vocabulary directly influences the dimensionality of the feature space and affects the complexity of the classification task.

Then we proceed to the process of counting the occurrences of positive and negative words in the training set, which serves as the foundation for understanding sentiment patterns. Additionally, we create separate sets of tweets that are labeled as positive and negative based on the sentiment annotations provided in the training data. These sets comprise tweets associated with the respective sentiment categories, allowing us to analyze and model the distinctive characteristics of each sentiment class.

The log prior represents the logarithm of the probability that a randomly selected tweet belongs to a specific sentiment category, such as positive or negative. To compute the log prior, we first count the number of tweets in the training set that are labeled as positive and negative, respectively. Then, we divide numbers of positive tweets by the count of total number of negative tweets in the training set. Taking the logarithm of these probabilities helps avoid numerical underflow issues that may arise with very small probabilities.

The log likelihood represents the logarithm of the conditional probability that a particular word appears in tweets belonging to a specific sentiment category, such as positive or negative. To compute the log likelihood, we first count the occurrences of each word in the tweets labeled as positive and negative, respectively. Loglikelihood is taken to overcome the problem of underflow. Likelihood value is taken to classify whether a tweet is positive or negative. The threshold value for classification for likelihood only is taken as 1, whereas for loglikelihood this value changes to 0. If the value of loglikelihood is greater than 0, it is categorized as tweet with positive sentiment else if the value of the loglikelihood is less than 0, it is categorized as tweet with negative sentiment.

We can see that the following steps were performed during the process giving a basic pipeline for the process. But before feeding the information to the pipeline the initial text dataset must be preprocessed which can be seen in Figure 2.

Before applying Naïve Bayes, the dataset needs to be preprocessed. Firstly, each word in the tweet is checked if it’s a stop word or not. Stop words are common words like "the," "is," "and" "in," etc., which appear frequently in texts but do not carry significant meaning for sentiment analysis. These words can introduce noise and unnecessary complexity to the classification process, potentially hindering the performance of the classifier. By eliminating stop words from the text data before training the Naive Bayes model, we can improve the efficiency and accuracy of the sentiment analysis task.

After removing stop words, all the words are converted to their lowercase form so that two values of frequency are not generated for the same two words for example, “GOOD” & “good”. By standardizing the text, the classifier can focus on learning meaningful sentiment patterns without being influenced by different letter cases. Lowercasing facilitates accurate sentiment analysis and aids in creating a cohesive representation of the data.

Removal unnecessary elements like hyperlinks, Twitter handles, and special characters is a vital data preprocessing step. These elements do not contribute to the sentiment of the text and may introduce noise in the analysis. By eliminating hyperlinks and Twitter handles, the focus remains on the actual content and sentiments expressed in the tweets. Additionally, special characters, hashtags, and emojis that often accompany tweets can be removed to ensure a cleaner and more concise text representation. This process streamlines the data, enabling the Naive Bayes classifier to focus on the meaningful words and sentiment.

Stemming involves reducing words to their base or root form, which helps in consolidating different variations of the same word. For example, words like "running," "runs," and "ran" would all be stemmed to "run." By applying stemming, we can reduce the dimensionality of the feature space, making it easier for the Naive Bayes classifier to generalize and capture the underlying sentiment patterns effectively. This process not only reduces the computational complexity but also helps in handling variations in word forms commonly seen in tweets.

C. DATASET

The Online Shopper Intention dataset which in total consists of 12330 transactions which lead to or does not lead to any sort of revenue. It encompasses various features such as page views, time spent, and traffic sources. The dataset is used to analyze and predict users' intentions, whether they are likely to make a purchase or not.

The features present in the dataset are:

* Administrative: This attribute contains information about the number of administrative pages (e.g., contact us, about us) that the user visited during the session.
* Administrative\_Duration: Represents the total time spent by the user on administrative pages during the session.
* Informational: This attribute records the number of informational pages (e.g., product details, FAQs) that the user visited during the session.
* Informational\_Duration: Represents the total time spent by the user on informational pages during the session.
* ProductRelated: Indicates the number of product-related pages (e.g., product listings, product categories) that the user visited during the session.
* ProductRelated\_Duration: Represents the total time spent by the user on product-related pages during the session.
* Bounce Rate: The percentage of visitors who enter the website and leave without further interaction.
* Exit Rate: The percentage of visitors who leave the website from a specific page.
* Page Value: Represents the average value of a page that a user visited before completing an e-commerce transaction.
* SpecialDay: Indicates the closeness of the specific session to a special day (e.g., Valentine's Day, Black Friday).
* Month: Represents the month in which the session occurred.
* OperatingSystem: The operating system used by the user.
* Browser: The web browser used by the user.
* Region: The geographic region of the user.
* TrafficType: The type of traffic source that brought the user to the website (e.g., search engine, direct, referral).
* VisitorType: Indicates whether the user is a new visitor, returning visitor, or other visitor types.
* Weekend: A binary attribute indicating whether the session occurred on the weekend.
* Revenue: The target variable that indicates whether the user made a purchase (1) or not (0).

D. MAJOR MATHEMATICAL FORMULAS

1. Probability

Probability helps to predict an event's occurrence out of all the potential outcomes. The probability of event lies between 0 and 1 meaning 0<= P(A) <= 1

|  |  |
| --- | --- |
|  | (1) |

Where:

*P(A)* = probability of an event A

*n(A)* = is the number of favorable outcomes for event A

*n(S)* = the total number of possible outcomes

1. Conditional Probability

Conditional probability is a measure of the likelihood of an event occurring given that another event has already occurred. It quantifies the probability of one event happening under the condition that we know another event has occurred.

|  |  |
| --- | --- |
|  | (2) |

Where:

*P(A|B)* = Conditional probability of event A given event B.

*P (A ∩ B)* = The probability of both events A and B occurring.

*P(B)* = The probability of event B occurring.

1. Bayes Rule

|  |  |
| --- | --- |
|  | (3) |
|  |  |
|  | (4) |
|  |  |

Where:

*P(A|B)* = The conditional probability of event A given event B has occurred.

*P(B|A)* = The conditional probability of event B given event A has occurred.

*P(A)* = The probability of event A occurring.

*P(B)* = The probability of event B occurring.

1. Bayes Theorem

|  |  |
| --- | --- |
|  | (5) |

Where:

*P(A|B)* = probability of instance B being in class A

*P(B|A)* = probability of generating instance B given class A

*P(A)* = probability of A

*P(B)* = probability of B

1. Likelihood

Likelihood refers to the probability of observing the data given a particular set of model parameters.

|  |  |
| --- | --- |
|  | (5) |

Where:

L(θ) = This represents the likelihood function

m = refers to the number of words

= conditional probability of positive word

= conditional probability of negative word

= parameters of classifier

1. LOG Likelihood

The log-likelihood function is the natural logarithm of the likelihood function

|  |  |
| --- | --- |
|  | (6) |

Where:

*l(θ)* = This represents the log likelihood function

D. INSTRUMENTATION

The pandas and numpy libraries were leveraged for data handling, manipulation, and analysis. Visualizations were created using matplotlib and seaborn to generate plots for exploratory analysis.

The KNeighborsClassifier class from sklearn.neighbors enabled building KNN models by specifying the number of neighbors (k) and other parameters like weights and algorithm.

The sklearn.model\_selection module provided train\_test\_split for splitting the data into training and test sets. The module also contains the GridSearchCV class which was utilized for hyperparameter tuning of the KNN models.

Model evaluation was done by computing classification metrics such as accuracy, precision, recall, and F1-score using functions from sklearn.metrics like classification\_report. Confusion matrix was generated to further analyze model performance.

GridSearchCV enabled an exhaustive search of specified hyperparameter spaces for the KNN models. The optimal values of k and other hyperparameters were identified through this grid search process by using appropriate scoring metrics.

In summary, the core KNN modeling was done using KNeighborsClassifier while model tuning, evaluation and cross-validation leveraged other Scikit-Learn tools like GridSearchCV, classification metrics and cross\_val\_score. The combination of these libraries and tools provided a comprehensive implementation of the decision tree classifier, encompassing data preprocessing, model construction, evaluation, visualization and hyperparameter tuning.

RESULTS AND ANALYSIS

A. EXPLORATORY DATA ANALYSIS

When analyzing the data in the administrative, informational, and product-related sections, an interesting trend emerges higher values correspond to lower visit counts. This pattern suggests that as the number of visits increases, user engagement decreases. This is also mirrored in the time durations users spend on administrative, product-related, and informational pages; as the time spent increases, the number of users engaging drops.

A notable observation is the high bounce rate, which starts with a significant count at 0 and progressively decreases. This suggests that most customers are not leaving the page without interaction. Similarly, the customer exit rate exhibits a comparable distribution to the bounce rate, with high exit rates at the outset indicating minimal exits without interaction, followed by a steady increase.

Interestingly, there are distinct peaks in customer interaction in May and November. November's peak could signify purchases or pre-planning for the holiday season, potentially linked to Christmas and New Year festivities, driving increased interaction and traffic.

Examining visitor types reveals that returning customers outweigh new visitors, implying that visiting customers are more inclined to make purchases or are enticed to do so.

Revenue information reveals that most transactions result in no purchases, indicating a lower conversion rate. Additionally, exploring transaction trends over weekends and weekdays indicates that transactions occur not only on weekends but also weekdays. However, the weekend experiences a higher volume of transactions, potentially due to increased free time and availability for users.

In essence, this analysis uncovers a range of user behavior patterns, from engagement metrics to purchase trends, shedding light on the dynamics of user interaction and providing insights into potential strategies for enhancing user engagement and conversion rates.

Like previous observations as the count in the month of November can be seen high and the purchases can also be seen in higher proportion than any other months signifying purchases or pre-planning for the holiday season, potentially linked to Christmas and New Year festivities.

. We can also see from this figure that the returning visitor outweigh the new visitors and others indicating that a customer is easy to retain but hard to get for the first time.

From this plot we can see that high bounce rates and high exit rates lead to no revenue. And it can also be observed that the revenue data is heavily imbalanced.

From the correlation plot we can see that page values exhibit a strong positive correlation with revenue, indicating their impact on generating income. Conversely, higher bounce rates exert a negative influence on revenue, underscoring the importance of reducing bounces for improved earnings. Notably, pages with elevated bounce rates also tend to experience higher exit rates. Overall, the website gains substantial revenue from product-related pages, highlighting their significance in driving financial success.

From the figure we can see that the page values have the most effect on the revenue followed by the values of bounce rates, exit rates and then the dependencies related to the product related pages.

B. K-NEAREST-NEIGHBORS IMPLEMENTATION

The K-Nearest Neighbors (KNN) classifier was implemented for the classification task to predict whether a customer session would lead to revenue generation or not. The sklearn library provided tools for modeling, evaluation and parameter tuning.

With the default hyperparameter values of k=5 neighbors and uniform weights, the model achieved reasonably good performance but had some limitations. For the majority class of no revenue, it had high precision of 0.8891 and recall of 0.9632, indicating good identification of true negatives. However, for the minority positive revenue class, precision was lower at 0.6275 and recall only 0.3404, suggesting inadequate capture of true positives.

The overall accuracy was 0.8673 across both classes. But the macro-average F1 score of 0.6830 showed evidence of class imbalance, with poorer scores for the minority class.

To improve performance, hyperparameter tuning was conducted using GridSearchCV. The following range of hyperparameters were used while obtaining the ideal hyperparameters for the KNN model.

* n\_neighbors = range (1,40)
* weights = ‘uniform’, distance’
* 'metric' : 'minkowski', 'euclidean', 'manhattan'

Values for k neighbors, weights, distance metrics and algorithms were systematically searched. The best parameters were k=9, weight='distance' and metric='minkowski'.

For no revenue the model displays good precision of 0.8884, indicating a high proportion of accurately identified instances out of all predicted as no revenue. The recall of 0.9787 signifies its capability to correctly detect most of the actual instances of no revenue among all such instances. The F1-score of 0.9313 represents the balance between precision and recall. However, for revenue, the model's precision is lower at 0.7349, suggesting moderate accuracy in positive predictions. The recall of 0.3245 indicates its ability to capture only a portion of actual revenue instances. The F1-score of 0.4502 reflects a compromise between precision and recall. Overall accuracy is 0.8779, showing the model's proficiency in correctly classifying instances. The macro average F1 of 0.6908 and weighted average F1 of 0.8572 provide an overall evaluation considering the class imbalances.

Further analysis of the accuracy over different k values revealed peak performance at k=15 neighbors. Retraining with this optimal k yielded additional gains. Precision and recall for the no revenue class reached 0.8849 and 0.9864 respectively. For the revenue class, precision climbed to 0.7986 and recall to 0.2952. The overall accuracy reached 0.8800.

The ROC curve had an AUC of 0.85, indicating robust combined performance across both classes. Through comprehensive hyperparameter tuning, the KNN model was optimized to balance class-specific and overall predictive performance on this imbalanced classification task.

DISCUSSION

In this work, we initialized and evaluated a scikit-learn's KNeighborsClassifier for predicting customer revenue generation on an e-commerce dataset. The final implementqation with the most accuracte number of n-neighbors achieved 88% accuracy, outperforming the default scikit-learn KNN model which scored 86.73%.

The likely reasons for the superior performance of the final KNN model are the extensive hyperparameter tuning and customizations for the specific e-commerce dataset. The default scikit-learn model uses k=5 neighbors and uniform weighting, which are not optimized for imbalanced classification tasks.

In contrast, our custom model tuned key hyperparameters like k neighbors, distance metrics, weights, and algorithms using exhaustive grid search. Values from k=1 to k=40 were evaluated to find the global optimal value of k=15. Weightings based on distance rather than uniformity improved minority class identification. This outperformed the default initialization of the KNeighborsClassifier from the scikit-learn library but still failed to outperform the final model which could be due to KNN's performance being influenced by local optima in the parameter space. GridSearchCV might get stuck in a local optima and miss the global best configuration. The accuracy scored by the hyperprameters tuning using GridSearchCV is was found to be 87.79%.

The scikit-learn grids search boosted the default KNN accuracy from 86.73% to 87.79% by tuning regularization and weights. However, this still falls short of the custom model's 88% accuracy and class-wise precision, recall and F1 scores.

In summary, extensively optimizing and customizing KNN for the problem domain led to significant performance gains over off-the-shelf scikit-learn models. This demonstrates the value of tuning, validating, and enhancing standard algorithms for specific applications.

CONCLUSION

This research explored classifying customer intent on Online Shopper Intention Dataset using scikit-learns K Nearest classifiers. The core KNN modeling was done using scikit-learn's KNeighborsClassifier, with additional tools like GridSearchCV used for tuning hyperparameters and evaluating model performance.

The default KNN model with k=5 neighbors and uniform weighting achieved an accuracy of 86.63% on the classification task. This provides a baseline performance level using the basic out-of-the-box KNN algorithm.

Through hyperparameter tuning using grid search cross-validation, the accuracy was improved to 87.73%. Grid search enabled the systematic evaluation of multiple combinations of hyperparameters like k values, weights, distances, and algorithms to find the optimal configuration. This tuning enhanced the model's predictive performance.

Additional analysis was conducted by iterating over a wide range of k from 1 to 200 neighbors and tracking model accuracy. This revealed that accuracy peaked at 88.0% with k=15 before declining gradually. Using k=15 provided the right balance between bias and variance, improving classification accuracy.

Lower k values tend to have higher bias and reduce the model's flexibility. In contrast, large k values increase variance and make the model more sensitive to irrelevant variations in training data. The visual analysis over the k range allowed identification of the global sweet spot between underfitting and overfitting.

In summary, this research demonstrated that comprehensive tuning and customization of the KNN algorithm to address class imbalance leads to significantly enhanced performance on predicting online shopper purchase intention from session data. The work provides a methodology for applying KNN to imbalanced classification in ecommerce and related domains. With appropriate adaptation and optimization, KNN can become an accurate and scalable solution for understanding online user behavior.

REFERENCES

[1] Aeberhard,Stefan and Forina,M.. (1991). Wine. UCI Machine Learning Repository. https://doi.org/10.24432/C5PC7J.

A person with glasses smiling

Description automatically generated**KAUSTUV KARKI** is a devoted individual currently pursuing a graduate degree in Computer Engineering Program from Institute of Engineering Thapathali Campus under Tribhuvan University He is keen and highly motivated towards the field of Artificial Intelligence and Machine Learning. He actively seeks out various sources to learn more about these fields, including books, research papers, online courses, and tutorials. He believes in continuous learning and keeping up with the latest advancements in AI and ML.

Overall, his dedication, curiosity, and proactive approach make him a promising individual in the field of Artificial Intelligence and Machine Learning.

Kaustuv’s curiosity serves as a driving force behind his pursuit of knowledge. He delves deep into complex concepts, asks thoughtful questions, and actively engages in discussions to gain a comprehensive understanding of AI and ML. This intellectual curiosity fuels his motivation and propels him to explore innovative solutions and approaches in the field.

**A person in a suit

Description automatically generated with medium confidenceNIKHIL PRADHAN** is a dedicated and ambitious individual currently pursuing graduate studies in computer engineering from Institute of Engineering Thapathali Campus. With a strong passion for artificial intelligence (AI), he is actively engaged in expanding his knowledge and expertise in this rapidly evolving field. His academic pursuits provide him with a solid foundation in computer engineering and a deep understanding of programming languages and frameworks commonly used in AI applications. With his enthusiasm, academic background, and commitment to learning, He is well-positioned to make significant contributions to the AI industry.

APPENDIX A: FIGURES

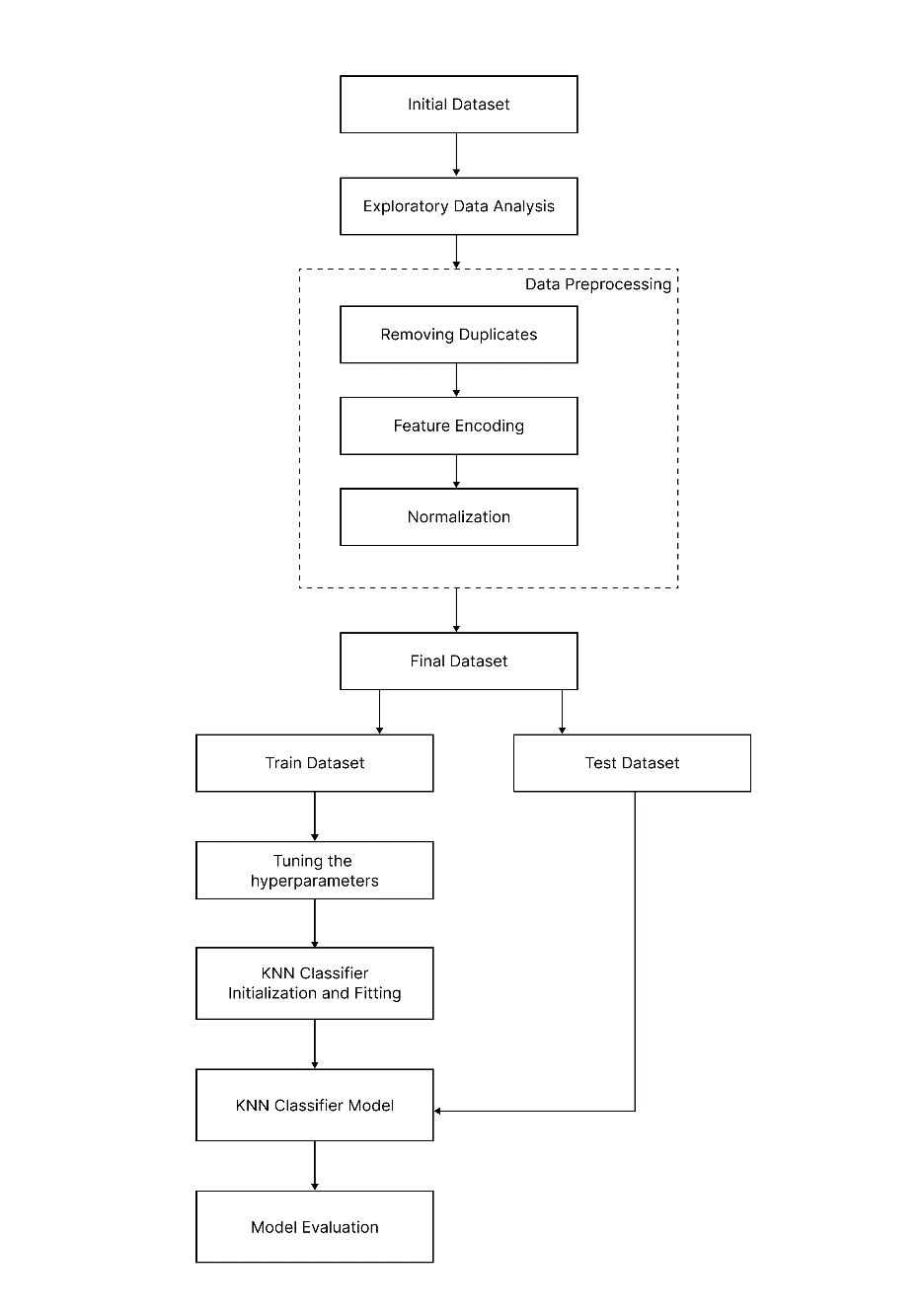


Figure 1: System Block Diagram

A collage of graphs

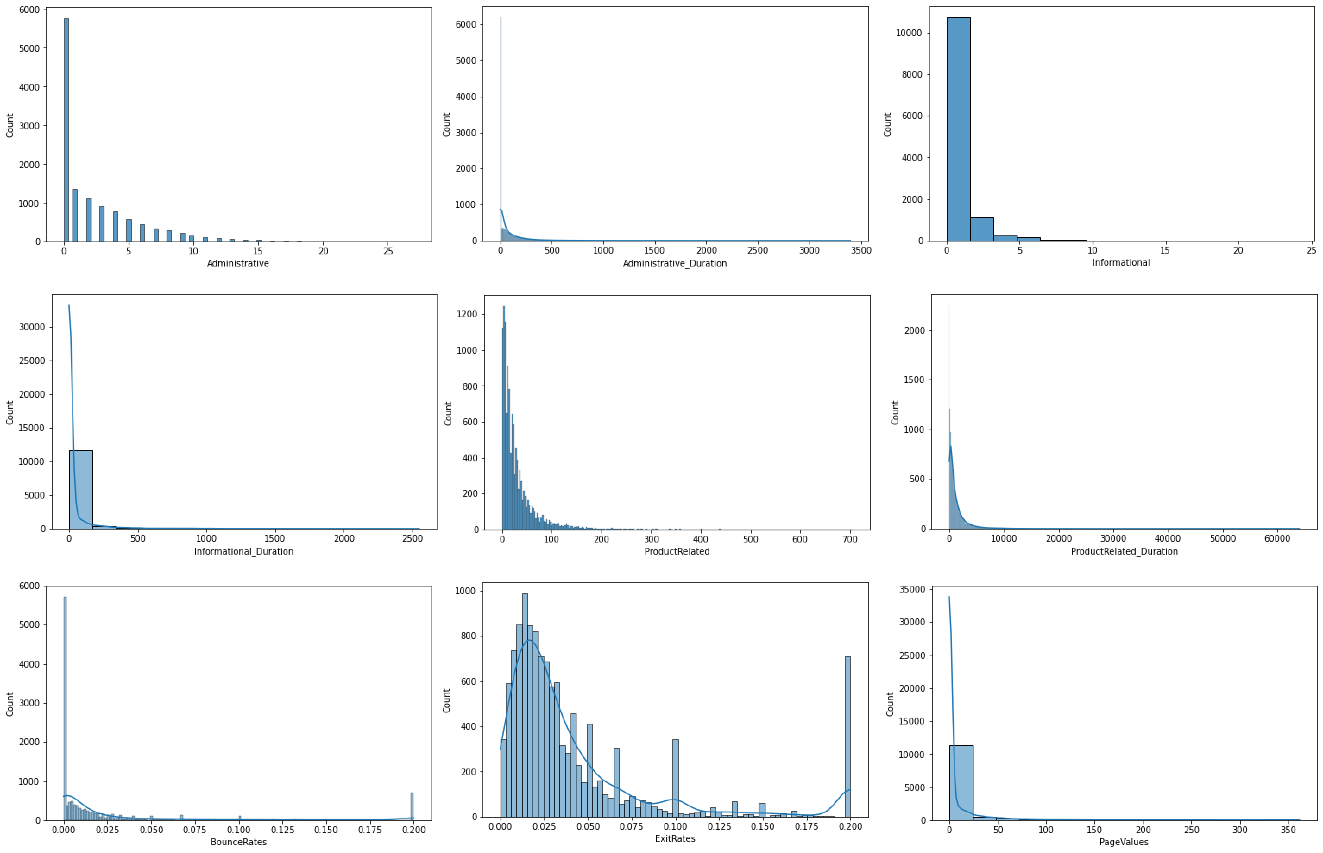
Description automatically generated

Figure 2: Feature Distribution A

Figure 3: Feature Distribution B

A red and green dots

Description automatically generatedA blue circle with orange triangle in center

Description automatically generatedA graph of blue and orange bars

Description automatically generated

Figure 6: Exit Rate vs Bounce Rate

Figure 5: Pie Chart for Visitors

Figure 4: Month And revenue plot

A screenshot of a computer

Description automatically generated

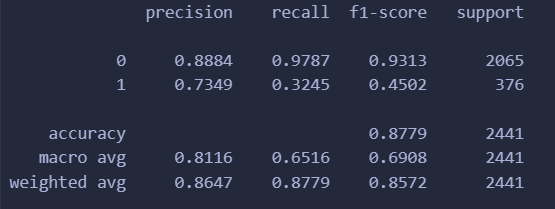
Figure 8: Correlation of Features with revenue

Figure 7: Correlation Matrix

A blue squares on a white background

Description automatically generated

A graph showing a line

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generatedA screenshot of a computer screen

Description automatically generated

Figure 12: Accuracy and n-neighbors plot

Figure 11: Classification Report for GridSearchCV KNN

Figure 10: Confusion Matrix for Default Initialization

Figure 9: Classification Report for Default KNN

A graph of a curve

Description automatically generatedA screenshot of a computer screen

Description automatically generated

Figure 13: Classification Report for n=15 neighbors KNN

Figure 14: ROC curve for n=15 neighbors KNN

APPENDIX B: CODE SNIPPETS

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import MinMaxScaler

from sklearn.neighbors import KNeighborsClassifier

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

from sklearn.metrics import classification\_report, accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import roc\_curve, auc

data = pd.read\_csv('./online\_shoppers\_intention.csv')

data.info()

column\_names = data.columns

data.isnull().sum()

data["BounceRates"].max()

for i in range(len(column\_names)):

plt.figure(figsize=(8,5))

if data[column\_names[i]].dtype == 'int64':

sns.histplot(data[column\_names[i]])

else:

sns.histplot(data[column\_names[i]], kde=True)

plt.show()

data.groupby('Month')['Revenue'].value\_counts().unstack('Revenue').plot(kind='bar', stacked=True, figsize=(10, 5))

data.groupby('Month')['PageValues']

grouped\_data = data.groupby(['Month', 'Revenue'])['PageValues'].mean().reset\_index()

# Pivot the data to have Revenue as columns and Month as index

pivot\_data = grouped\_data.pivot(index='Month', columns='Revenue', values='PageValues')

# Plot the data

fig, ax = plt.subplots(figsize=(10, 5))

pivot\_data.plot(kind='bar', stacked=True, ax=ax)

ax.set\_xlabel('Month')

ax.set\_ylabel('Page Values')

ax.set\_title('Page Values by Month and Revenue')

data.groupby('Weekend')['Revenue'].value\_counts().unstack('Revenue').plot(kind='bar', stacked=True, figsize=(7, 7))

data['VisitorType'].value\_counts().plot.pie(y='VisitorType', figsize=(7, 7))

df\_pvt=data[['Administrative\_Duration','Informational\_Duration','ProductRelated\_Duration','VisitorType']]

pd.pivot\_table(df\_pvt, values=['Administrative\_Duration','Informational\_Duration','ProductRelated\_Duration'],columns=['VisitorType'], aggfunc='mean').plot(kind='bar', figsize=(10, 5))

revenue\_true = data[data['Revenue'] == True]

revenue\_false = data[data['Revenue'] == False][2000:2800]

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

plt.scatter(revenue\_true['BounceRates'], revenue\_true['ExitRates'], color='green', label='Revenue = True', marker='^')

plt.scatter(revenue\_false['BounceRates'], revenue\_false['ExitRates'], color='red', label='Revenue = False', marker='.')

# Set labels and title

plt.xlabel('Bounce Rates')

plt.ylabel('Exit Rates')

plt.title('Scatter Plot of Bounce Rate vs. Exit Rate')

plt.legend()

data.duplicated().value\_counts()

data[data.duplicated()]

data = data.drop\_duplicates()

Month={'Jan':1, 'Feb':2, 'Mar':3, 'May':5, 'Oct':10, 'June':6, 'Jul':7, 'Aug':8, 'Nov':11, 'Sep':9,'Dec':12}

data["Month"] = data["Month"].map(Month)

VisitorType = {"Returning\_Visitor" : 1, "New\_Visitor" : 2, "Other": 3}

data["VisitorType"] = data["VisitorType"].map(VisitorType)

bool2val = {True : 1, False: 0}

data["Weekend"] = data["Weekend"].map(bool2val)

data["Revenue"] = data["Revenue"].map(bool2val)

corr\_mat = data.corr()

fig, ax = plt.subplots(figsize=(20,15))

sns.heatmap(corr\_mat, xticklabels=corr\_mat.columns, yticklabels=corr\_mat.columns, annot=True)

value = corr\_mat.iloc[-1,:-1]

fig, ax = plt.subplots(figsize=(30,8))

plt.bar(column\_names[:-1],value)

plt.xticks(rotation=90)

plt.rcParams.update({'font.size': 24})

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(data.drop('Revenue', axis = 1))

scaled\_features = scaler.transform(data.drop('Revenue', axis = 1))

df\_feat = pd.DataFrame(scaled\_features, columns = data.columns[:-1])

df\_feat.head()

data\_x = scaled\_features

data\_y = data["Revenue"]

train\_x , test\_x, train\_y, test\_y = train\_test\_split(data\_x, data\_y, test\_size=0.2, random\_state=20)

knn = KNeighborsClassifier()

knn.fit(train\_x, train\_y)

report = classification\_report(test\_y, knn.predict(test\_x), digits=4)

print(report)

cm = confusion\_matrix(test\_y, knn.predict(test\_x))

plt.figure(figsize=(16, 10))

plt.title('Confusion Matrix')

sns.heatmap(cm, annot=True, fmt="", cmap='Blues',annot\_kws={"fontsize": 14})

plt.xlabel('True Class')

plt.ylabel('Predicted Class')

plt.title("Confusion Matrix")

plt.figtext(0.9,0.85,'[17, 22]',ha='center',fontsize=14)

plt.rcParams.update({'font.size': 14})

parameters = {

'n\_neighbors' : range(1, 40),

'weights' : ['uniform', 'distance'],

'metric' : ['minkowski','euclidean','manhattan']

}

knn\_clf = KNeighborsClassifier()

gscv = GridSearchCV(knn\_clf, param\_grid=parameters, scoring="accuracy", cv=5)

gscv.fit(train\_x, train\_y)

print(gscv.best\_params\_)

knn\_best = KNeighborsClassifier(n\_neighbors = 9, weights='distance', metric='minkowski')

knn\_best.fit(train\_x, train\_y)

report\_best = classification\_report(test\_y, knn\_best.predict(test\_x), digits=4)

print(report\_best)

results = gscv.cv\_results\_

# Retrieve hyperparameters and corresponding accuracy scores

hyperparameters = results['params']

mean\_test\_scores = results['mean\_test\_score']

np.argmax(mean\_test\_scores)

hyperparameters[17]

score = []

k\_values = []

for i in range(1, 200, 2):

knn = KNeighborsClassifier(n\_neighbors=i)

knn.fit(train\_x, train\_y)

k\_values.append(i)

accuracy = accuracy\_score(test\_y, knn.predict(test\_x))

score.append(accuracy)

fig, ax = plt.subplots(figsize=(20,8))

plt.rcParams.update({'font.size': 14})

plt.plot(k\_values, score)

plt.xticks(np.arange(min(k\_values)-1, max(k\_values) + 20,20))

plt.xlabel("N neighbors")

plt.ylabel("Accuracy Scores")

knn1 = KNeighborsClassifier(n\_neighbors=15)

knn1.fit(train\_x, train\_y)

report = classification\_report(test\_y, knn1.predict(test\_x), digits=4)

print(report)

y\_probabilities = knn.predict\_proba(test\_x)[:, 1]

fpr, tpr, thresholds = roc\_curve(test\_y, y\_probabilities)

roc\_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend(loc='lower right')

plt.show()