Task 1: Classification Model – D209

Abhishek Aern

Western Governor University

Contents

Part I - Research Question

A1: Question for analysis

Can k-nearest neighbors classification be used to identify specific factors that contribute most significantly to customer churn, and how can this information be leveraged to improve customer retention strategies?

A2: Goal of analysis

The goal of the analysis using k-nearest neighbors for customer churn is to develop an accurate predictive model that can identify customers who are at risk of churning. By accurately identifying these customers, businesses can take targeted steps to retain them, which can ultimately lead to increased revenue and profitability.

The KNN model will be developed using the prepared data, to predict which customers are most likely to churn. This model will be validated and optimized to ensure it is accurate and robust.

Part II - Method Justification

B1: Explanation of Classification Method - KNN

The KNN is a non-parametric algorithm, which means it does not make any assumptions about the distribution of the data. This makes it suitable for analyzing complex datasets that do not follow a normal distribution. Also, KNN is an interpretable algorithm, which means it is easy to understand how the model makes its predictions. This makes it easier for businesses to identify the factors that contribute to customer churn and develop targeted retention strategies (Vishalmendekarhere, 2021).

The K-nearest neighbors (KNN) will be the best fit for customer churn analysis because it is non-parametric, flexible, accurate, and interpretable.

B2: Assumptions of KNN

(Vishalmendekarhere, 2021) Here are some of the assumptions of KNN –

1. KNN relies on a distance metric to determine the similarity between observations. The choice of distance metric can have a significant impact on the results, and it is important to choose a metric that is appropriate for the data.

2. KNN assumes that all features are equally important and have the same scale. If some features are more important than others or have a different scale, it may be necessary to normalize or standardize the data.

3. KNN is sensitive to outliers, which can have a significant impact on the results. It may be necessary to remove outliers or use a robust distance metric to address this issue.

B3: Packages or Libraries List

Here is a list of packages and libraries for implementing the KNN algorithm in Python:

NumPy: A library for working with arrays and numerical operations.

Pandas: A library for working with data frames and manipulating data.

Scikit-learn: A machine learning library that provides various tools for classification, regression, and clustering, including KNN.

The train_test_split function is used from the Scikit-learn library in Python which is commonly used to split a dataset into training and testing sets for machine learning.

The KNeighborsClassifier class from the Scikit-learn library in Python is used to implement the K-Nearest Neighbors (KNN) algorithm for classification

The confusion_matrix, classification_report, and accuracy_score are functions used from the Scikit-learn library in Python for evaluating the performance of a classification model. The cross_val_score function is used from the Scikit-learn library for cross-validation.

Matplotlib: A plotting library for creating visualizations in Python.

Seaborn: A statistical data visualization library based on Matplotlib that provides a high-level interface for creating attractive visualizations.

Part III - Data Preparation

C1: Data Preprocessing

The goal of data preprocessing for the KNN model is to prepare the dataset for the algorithm by ensuring that the data is in the appropriate format and is suitable for use in the KNN algorithm.

C2: Dataset variables

Initially, the churn dataset has many variables which are not relevant to predict customer churn and hence get deleted while importing the data. Later using SelectKBest I selected the top 10 features from a dataset based on statistical tests.

Target variable –

Churn: This is a categorical variable having the value Yes/No.

Independent variable –

Variable Name	**Type**
Income	Continuous
Multiple	Categorical
Tenure	Continuous
MonthlyCharge	Continuous
Bandwidth_GB_Year	Continuous
InternetService_DSL	Categorical
InternetService_Fiber Optic	Categorical
Contract_Month-to-month	Categorical
Contract_One year	Categorical
Contract_Two Year	Categorical

C3: Steps for Analysis

(Kumar, 2018) Here is the task that I am following for data preprocessing-

Handling missing and Null values: Checking the missing and null values using the .isna() and .isnull() functions. No missing and null values were found in the provided dataset.

```
# check for null
print("Check for Nulls")
print("-"*100)
print(df_churn.isnull().any())

# check for missing values
print("Check for Missing Values")
print("-"*100)
print(df_churn.isna().any())
```

Handling categorical variables: I am using categorical variables such as Contract, Gender, InternetService, Multiple, PaperlessBilling, and target variable Churn, which needs to be converted into a numerical format using Label encoding and one-hot encoding.

```
# Label encoding for Churn, multiple and PaperlessBilling variables

cust_map = {"No": 0, "Yes": 1}          # Defining the mapping

df_churn['Churn'] = df_churn['Churn'].map(cust_map)
df_churn['Multiple'] = df_churn['Multiple'].map(cust_map)
df_churn['PaperlessBilling'] = df_churn['PaperlessBilling'].map(cust_map)

# Perform one-hot encoding

df_churn = pd.get_dummies(df_churn, columns=['InternetService', 'Contract',
'Gender'])

# convert uint8 columns to int
df_churn = df_churn.applymap(lambda x: int(x) if isinstance(x, int) else x)
```

Feature selection: KNN can be sensitive to irrelevant or redundant features, so deleting all the features which are not significant in predicting customer churn from my previous logistic regression analysis(D208).

The following features are deleted initially during import –

'CaseOrder', 'Lat', 'Lng', 'Interaction', 'UID', 'City', 'State', 'Zip', 'Population', 'Job', 'Techie', 'Port_modem', 'Tablet', 'Marital', 'Phone', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',

'PaymentMethod', 'Email', 'Item1', 'Item2', 'Item3', 'Item4', 'Item5', 'Item6', 'Item7', 'Item8', 'County', 'Area', 'TimeZone'

Also, I am using SelectKBest from Scikit-learn to select the top 10 features from a dataset based on statistical tests. I am using the chi-squared test as the scoring function and selecting the top 10 features. The function transform() selects the top K features based on their scores and returns a new feature matrix X_new with only the selected features.

Then I am getting the indices and names of the top K features using the get_support() method and indexing the original feature matrix columns.

Here are the top 10 features selected from this analysis –

```
Index(['Income', 'Multiple', 'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year',
       'InternetService_DSL', 'InternetService_Fiber Optic',
       'Contract_Month-to-month', 'Contract_One year', 'Contract_Two Year'],
      dtype='object')
```

```
#Import the necessary libraries
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

#Create your feature matrix X and target vector y
X = df_churn.drop('Churn', axis=1)
y = df_churn['Churn']

# Apply SelectKBest to the feature matrix:
k = 10                              # number of top features to select
selector = SelectKBest(chi2, k=k)
selector.fit(X, y)
X_new = selector.transform(X)

#Get the indices of the top K features
top_k_indices = selector.get_support(indices=True)

#Get the names of the top K features
top_k_features = X.columns[top_k_indices]
```

The selected features and dependent variable (Churn) are concatenated in a new DataFrame named df_churn_selected.

```
#Filter the original dataframe to include only the selected features:
X_selected = df_churn[top_k_features]
df_churn_selected = pd.concat([X_selected, y.to_frame()], axis=1)
df_churn_selected.head()
```

Feature scaling: KNN is sensitive to the scale of the features, so I am standardizing all continuous variables to ensure that they are on the same scale (Kumar, 2018).

```
# Select the continuous variables in the dataset
continuous_vars = ['Income', 'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year']
X_continuous = df_churn_selected[continuous_vars]

# Scale the continuous variables using StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_continuous)

# Replace the original continuous variables with the scaled values
df_churn_selected.loc[:, continuous_vars] = X_scaled
```

In the above code, I first select the continuous variables in the dataset and store them in a separate DataFrame called X_continuous. Then used StandardScaler to scale the values in this DataFrame and store the scaled values in a new DataFrame called X_scaled. Finally, replaced the original continuous variables in the original DataFrame with the scaled values using the pd.DataFrame.loc method.

All the code is present in the attached .ipynb file provided in the attachment.

C4: Prepared Dataset

The prepared clean data is provided in a CSV file.

Part IV - Analysis

D1: Splitting the Data

Defined the dependent variable (y) and independent variables (X) from df_churn_selected obtained from the SelectKBest method.

(Pedregosa) The train_test_split function from scikit-learn is used to randomly split the data into training and testing sets, where the test_size parameter specifies the proportion of the data to be used for testing (in this case, 20%). The random_state parameter is set to a fixed value (0 in this case) to ensure that the same split is obtained every time the code is run. The function returns four arrays: X_train and y_train represent the training data (features and target), while X_test and y_test represent the testing data (features and target).

```
From sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,
random_state = 0)
```

All the training and testing data sets are exported to a CSV file and will be provided with the submission.

D2: Output and Intermediate Calculations

After splitting the data, imported the classifier model from the sklearn library and fit the model by initializing K=1. I started KNN with K=1 and then evaluated the model then used the elbow method to pick a good K Value and compare the model performance.

Here is the confusion matrix when K = 1

```
print('K=1, Confustion Matrix=' ,'\n' , confusion_matrix(y_test,y_pred))
```

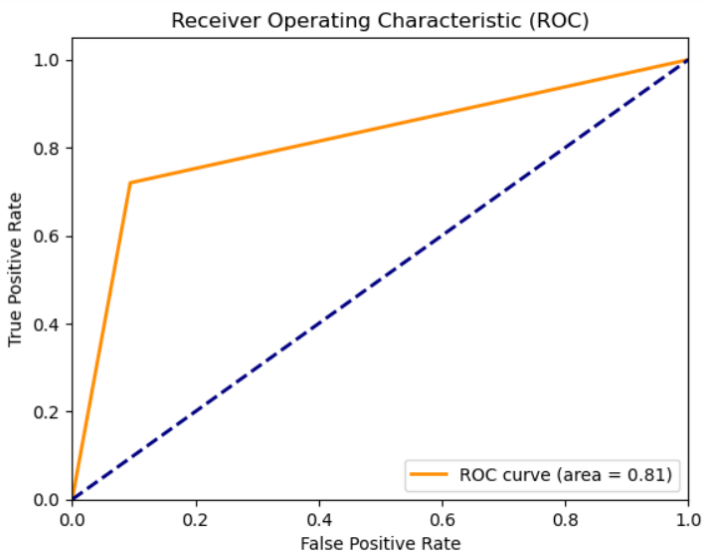
```
K=1, Confustion Matrix=
 [[1346  140]
 [ 144  370]]
```

Initial Model Accuracy –

```
# Accuracy on Train
print("Accuracy @ K = 1")
print("Training Accuracy is :" , knn.score(X_train, y_train ))

# Accuracy on Test
print("Testing Accuracy is :" , knn.score(X_test, y_test ))
```

```
Accuracy @ K = 1
Training Accuracy is : 1.0
Testing Accuracy is : 0.858
```

Initial model ROC curve -



To improve the model and find out the optimal k value, I am using Elbow Method.

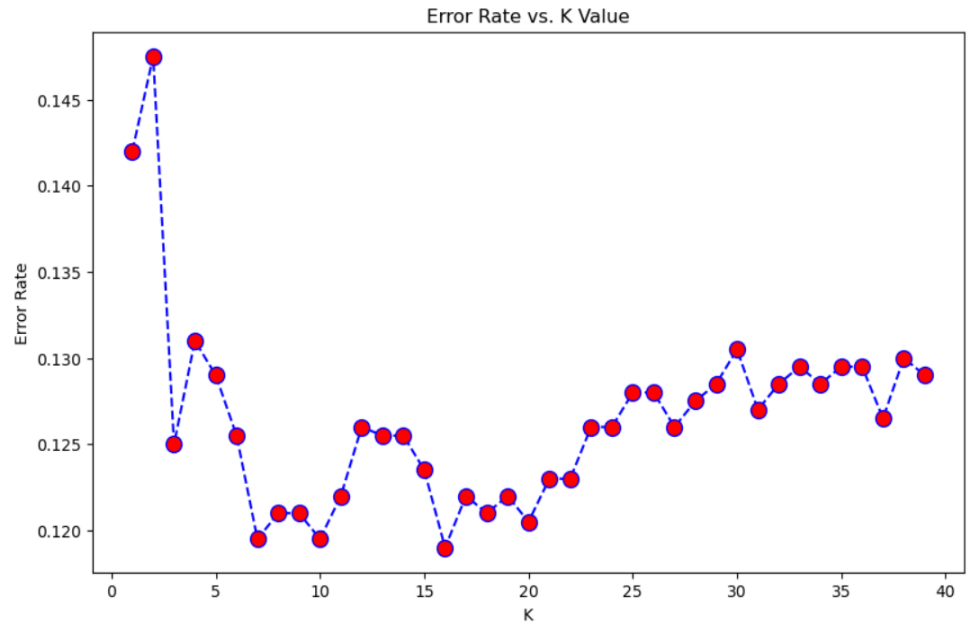
(Band, 2020) The below code is used to implement the Elbow method for KNN. We would first split our data into training and testing sets and then fit the KNN model for a range of K values (e.g., 1 to 40 in my case). For each K value, we would calculate the accuracy score on the testing set and record the corresponding value of the cost function. We would then plot the cost function values for each K value and look for the point where the curve starts to level off.

Once we have identified the optimal K value using the Elbow method, we can use that value to train and test our final KNN model on the full dataset.

```
error_rate = []
for i in range(1,40):
 knn = KNeighborsClassifier(n_neighbors=i)
 knn.fit(X_train,y_train)
 pred_i = knn.predict(X_test)
 error_rate.append(np.mean(pred_i != y_test))
## Plot the error rate vs. K values
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
print("Minimum error:-",min(error_rate),"at K
=",error_rate.index(min(error_rate)))
```

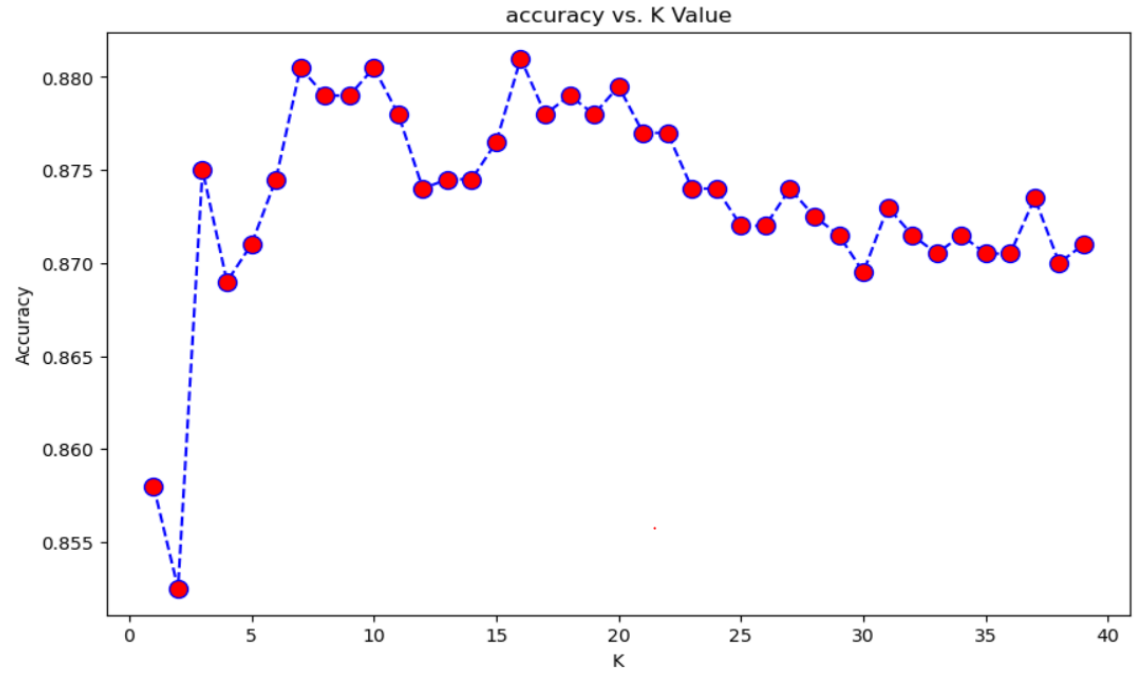
Based on this the optimum value of K is 15.

```
Minimum error:- 0.119 at K = 15
```



The above code is based on the error rate, and we can do the same thing for the Accuracy rate as well. The K value will be the same and I have that code demonstrated in the Jupiter notebook. Here is the output from the accuracy rate where the optimum value of K is found 15 for maximum accuracy (Band, 2020).

```
Maximum accuracy:- 0.881 at K = 15
```



D3: Code Execution

Based on the new value of the K, I built the new KNN model and calculated all the performance matrices.

```
knn = KNeighborsClassifier(n_neighbors=15)
knn.fit(X_train,y_train)

y_pred = knn.predict(X_test)
```

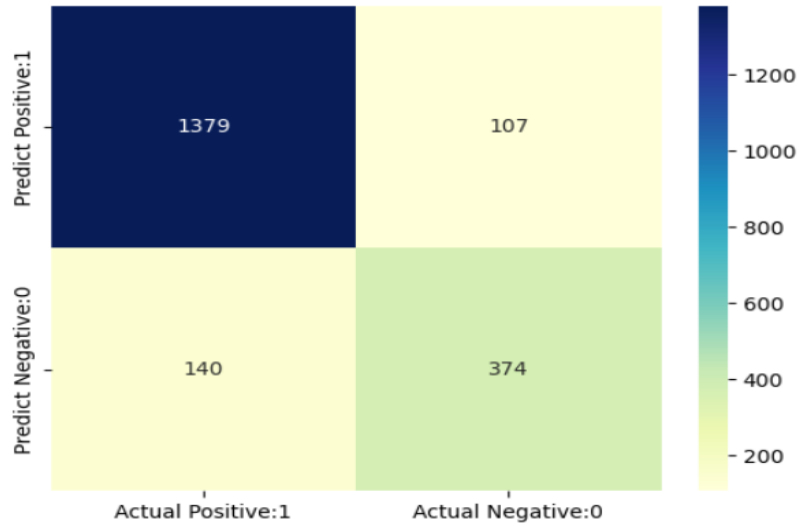
The performance and output of this new model are described in the next section.

Part V- Data Summary and Implications

E1: Accuracy and AUC

Confusion Matrix -

```
Confusion matrix

 [[1379  107]
 [ 140  374]]

True Positives(TP) =  1379

True Negatives(TN) =  374

False Positives(FP) =  107

False Negatives(FN) =  140

<AxesSubplot:>
```



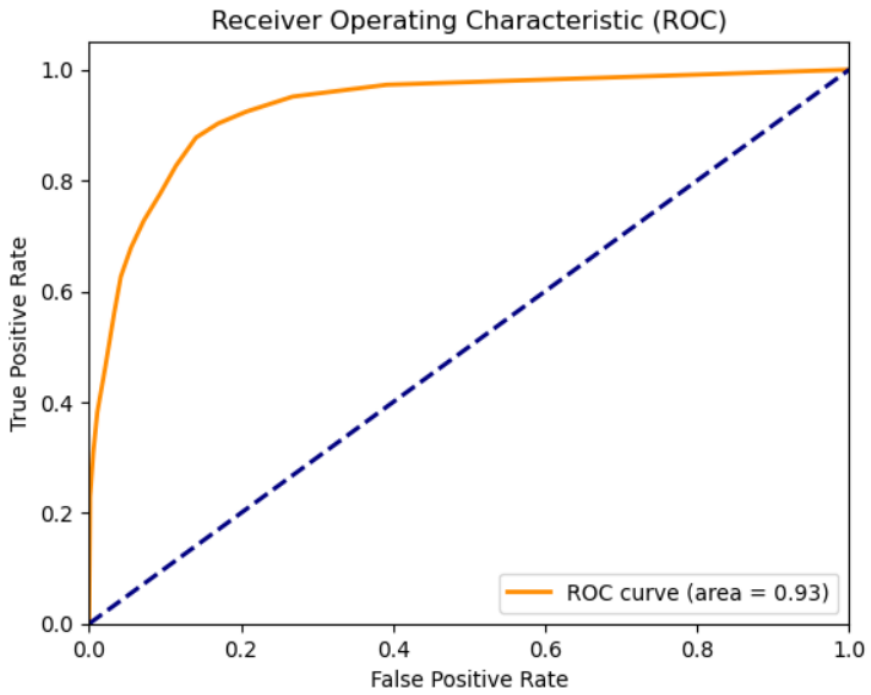
Accuracy - Classification accuracy is the number of correct predictions made as a ratio of all the predictions (Brownlee, 2018).

```
# Accuracy on Train
print("Accuracy @ K = 15")
print("Training Accuracy is :" , knn.score(X_train, y_train ))

# Accuracy on Test
print("Testing Accuracy is :" , knn.score(X_test, y_test ))
```

```
Accuracy @ K = 15
Training Accuracy is : 0.900625
Testing Accuracy is : 0.8765
```

ROC-AUC (Receiver Operating Characteristic - Area Under the Curve) is a measure of the model's ability to distinguish between the positive and negative classes. The score of 0.93 means that the KNN model has high accuracy in predicting the positive and negative classes of the customer churn data.

Here is the ROC curve –



E2: Results and Implications

(Brownlee, 2016) Based on the above performance matrix -

- 1379 customers were predicted to churn, and they actually churned(True Positives).
- 374 customers were predicted to not churn, and they actually did not churn (True Negatives).
- 107 customers were predicted to churn but they actually did not churn(False Positives).
- 140 customers were predicted to not churn but they actually churned(False Negatives).
- Accuracy is the proportion of correct predictions out of the total number of predictions. In this case testing data Accuracy is 87% and training data Accuracy is 90%.
- Precision is the proportion of true positive predictions out of all positive predictions. My model Precision is 93%.
- Recall (also known as sensitivity) is the proportion of true positive predictions out of all actual positive cases. My model recall is 91%.
- F1 score is the harmonic means of precision and recall. It takes into account both precision and recall giving a single measure of the model's performance. My model F1 score is 92%.
- A ROC (Receiver Operating Characteristic) score of 0.93 indicates that the KNN model has a good ability to distinguish between positive and negative classes (Brownlee, 2016).

All the above matrix shows that the model can correctly identify a large proportion of customers who are likely to churn while minimizing the number of false positive predictions. However, there is still room for improvement and further analysis could be done to refine the model and improve its performance.

E3: Limitation

(Vishalmendekarhere, 2021) Here are some of the limitations –

KNN is a computationally expensive algorithm, especially for large datasets, since it needs to calculate distances between each pair of data points.

KNN algorithm can be sensitive to the choice of distance metric used to calculate distances between the data points.

KNN algorithm requires a suitable choice of the hyperparameter k, which can significantly affect its performance. Choosing an inappropriate value of k can lead to overfitting or underfitting of the model.

E4: Course of Action

Based on the KNN model performance and the identified limitations, To improve the accuracy of the KNN model, the organization can work on improving the quality of the data and based on the insights gained from the KNN model, the organization can develop targeted retention strategies to reduce customer churn.

Also, the organization can consider experimenting with other classification models, such as decision trees, random forests, or support vector machines, to determine which model provides the best results for their specific use case(customer churn in this case).

Part VI - Demonstration

G. Provide a Panopto recording

The Panopto recording link is provided with the submission.

H. Third-Party Code References

Moffitt, C. (2017, February 06). Guide to Encoding Categorical Values in Python. pbpython.com: https://pbpython.com/categorical-encoding.html

pandas.pydata.org. (n.d.). pandas.get_dummies. https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html

Pedregosa. (n.d.). Scikit-learn: Machine Learning in Python. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

jaym. (n.d.). Feature selection using SelectKBest. https://www.kaggle.com/code/jepsds/feature-selection-using-selectkbest/notebook

I. References

Band, A. (2020, May 23). How to find the optimal value of K in KNN?
https://towardsdatascience.com/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb

Brownlee, J. (2016, May 25). Metrics To Evaluate Machine Learning Algorithms in Python.
https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python

Kumar, D. (2018, December 25). Introduction to Data Preprocessing in Machine Learning.
https://towardsdatascience.com/introduction-to-data-preprocessing-in-machine-learning-a9fa83a5dc9d

Saji, B. (2021, January 20). A Quick Introduction to K – Nearest Neighbor (KNN) Classification Using Python.
https://www.analyticsvidhya.com/blog/2021/01/a-quick-introduction-to-k-nearest-neighbor-knn-classification-using-python/

Srivastava, T. (2019, August 6). 12 Important Model Evaluation Metrics for Machine Learning Everyone Should Know.
https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/

Vishalmendekarhere. (2021, Jan 17). ML Assumptions, Pros & Cons.
https://medium.com/swlh/its-all-about-assumptions-pros-cons-497783cfed2d