Final report

Introduction:

This report documents the development and analysis of KNN and Naïve Bayes models for classification tasks as part of our data science course. The chosen dataset for this project is the Adult Income dataset sourced from Kaggle.

About Data set:

https://www.kaggle.com/datasets/wenruliu/adult-income-dataset

An individual's annual income results from various factors. Intuitively, it is influenced by the individual's education level, age, gender, occupation, native country, etc**.**

Fields:

- The dataset contains 16 columns (1 for index)
- Number of attributes: 14 - 6 continuous, 8 categorical**.**
- Target filed: Income
- The income is divided into two classes: <=50K and >50K
- Class Distribution**:**
- Probability for the label '>50K' : 23.93% / 24.78% (without unknowns)
- Probability for the label '<=50K' : 76.07% / 75.22% (without unknowns)
- Size: 48,842
- Size after removing rows with missing values: 45,225

Data Processing

Ensuring good data quality is crucial for the success of the model. To start, I removed rows with missing values. However, since the dataset is already quite large, this step didn't impact the performance of the models.

In the subsequent sections, I will delve into the specific data processing steps tailored for each model.

KNN Classifier

I developed a K-Nearest Neighbors (KNN) model with adjustable parameters, including 'k' for the number of neighbors and a choice of distance metric. The model encapsulates key functions: fit, predict, and evaluate.

- **fit(x_train, y_train, num_classes)**: This function establishes the model's understanding by accepting training data (x_train and y_train) and the number of classes in the dataset.
- **Knn.predict(vlues_to_predict):** By inputting data for classification, this function utilizes a user-defined distance metric (set during model instantiation) to calculate distances from each vector in the model's training data. The predicted class is determined via a voting mechanism among the 'k' nearest neighbors.
- **knn.evaluate(y_test, pred)**: This function, taking predicted and true labels as input, returns a tuple with accuracy, precision, recall, and f-measure metrics, using sklearn library offering a comprehensive assessment of model performance.

Data Processing

First try:

- **Label Encoding:**
  - Utilizes LabelEncoder from scikit-learn to convert categorical variables into numerical representations of integers.

- **Random Sampling:**
  - Due to dataset size, extracts a specified number of random samples (controlled by num_of_samples and random_state).
- **Data Splitting:**
  - Splits the dataset into feature matrix X and target variable y.
  - Splits X, y to: X_train, X_test, y_train, y_test Utilizes train_test_split() function of sklearn.
  - 0.7 for train, 0.3 for test.

Results:

I ran the model for different values of k, and the best results were:

random samples	**4000**
K	**11**
Accuracy	0.7900
Precision	0.1562
Recall	0.8333
F1 Score	0.2632

Second try:

I utilized One-Hot Encoding using Pandas, instead of Label Encoding (converting strings to integers).

Results:

I ran the model for different values of k, and the best results were:

random samples	**3000**
K	**21**
Accuracy	0.8500
Precision	0.5211
Recall	0.7708
F1 Score	0.6218

Third try:

- **Categorical One-Hot Encoding:**
  - Identifies categorical columns (String values) and performs one-hot encoding using Pandas.
- **Random Sampling:**
  - Due to dataset size, extracts a specified number of random samples (controlled by num_of_samples and random_state).
- **Normalization of non-binary columns using StandardScaler:**
  - Applies StandardScaler to normalize only the non-binary columns.

- **PCA Transformation:**
  - Utilizes PCA to transform the normalized data into principal components for reducing dimensionality.
- **Random Sampling:**
  - Due to dataset size, extracts a specified number of random samples (controlled by num_of_samples and random_state).
- **Stratified Sampling:**
  - Ensure that your training and testing datasets maintain the same distribution of classes as the original dataset, Utilizes train_test_split() function of sklearn.
- **Data Splitting:**
  - 0.7 - train, 0.3 - test.

Results:

I ran the model for different values of k, and the best results were:

random samples	**3000**

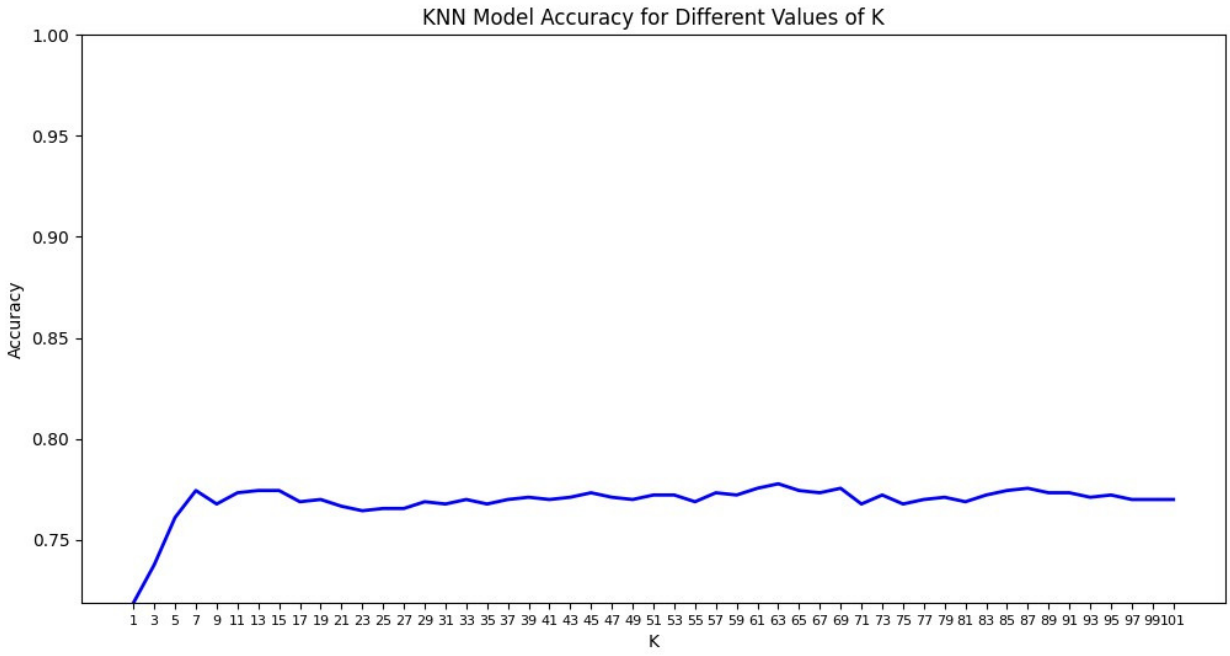
K	21
PCA components	**6**
Accuracy	0.8850
Precision	0.6383
Recall	0.8333
F1 Score	0.7229

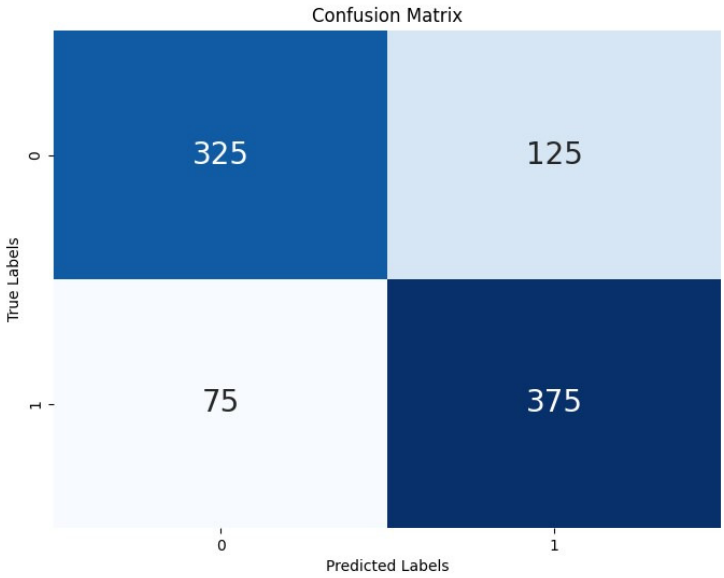
Upon achieving satisfactory data processing for the model, I observed that the probability for the label '>50K' stands at 24.78%, while the probability for the label '<=50K' is 75.22%. This indicates that my model outperforms a hypothetical model that simply classifies all inputs into the '<=50K' class by approximately 13%. Recognizing the unbalanced distribution of the dataset, I opted for a more realistic evaluation by taking a balanced random sample. This approach ensures that 50% of the data is from the class '<=50K' and the remaining 50% is from the class '>50K'.

Results:

- **Data Splitting:**
  - 0.7 - train, 0.3 - test.

```
K:  63
accuarcy:  0.7777777777777778
precision:  0.8333333333333334
recall:  0.75
f1:  0.7894736842105262
```

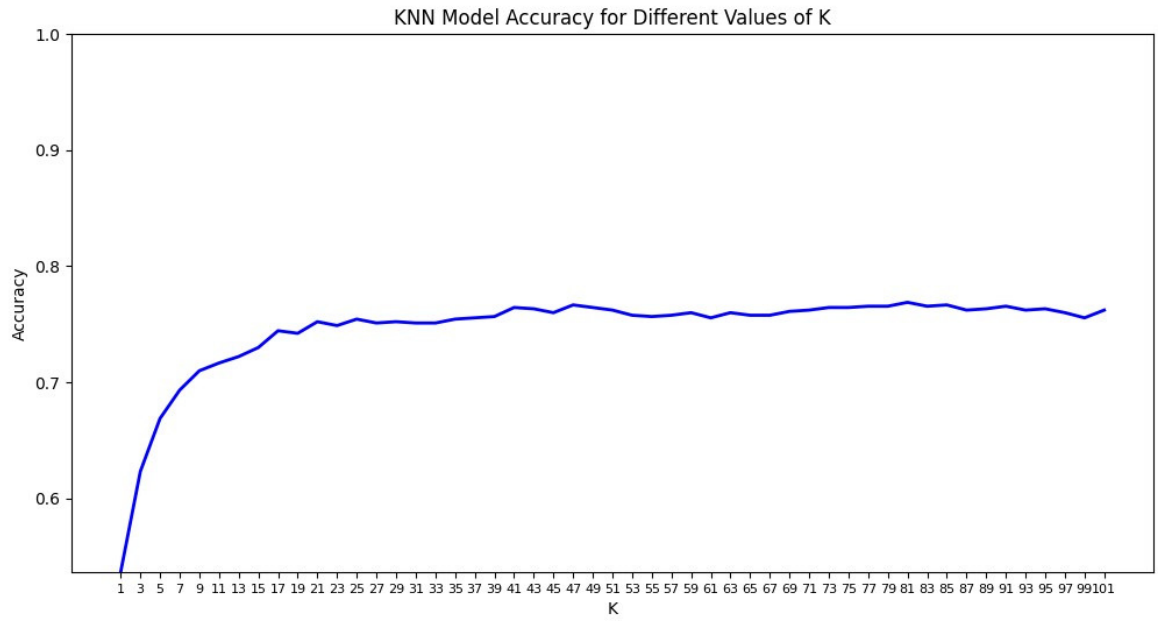




In an additional effort to reduce the data size, I employed clustering on the training data, creating 1500 centroids per class and splitting the data for 0.7 – train, 0.3 - test.

Results:

```
K:  81
accuarcy:  0.7688888888888888
precision:  0.7911111111111111
recall:  0.7574468085106383
f1:  0.7739130434782608
```



Naïve Bayes Classifier:

I developed a Naive Bayes model which is initialized with a specified class column (target variable)**.** The model encapsulates key functions: fit, predict, and evaluate.

fit(df_train: pd.DataFrame):

- Calculates feature vectors and sizes for each class in the training data.
- Computes word probabilities for each word in each class using Laplace smoothing.
- Calculates class probabilities.

predict(X_test: np.ndarray):

- Accepts data to be predicted (X_test) and returns predicted classes.
- Evaluates the probability of a data point belonging to each class.
- Considers Laplace smoothing for missing values.
- The predicted value is the class with max prob.

evaluate(y_test, pred):

- This function, taking predicted and true labels as input, returns a tuple with accuracy, precision, recall, and f-measure metrics, using sklearn library offering a comprehensive assessment of model performance.

Data processing

First try:

- **Data Loading:**
  - Utilizes Pandas to create a data frame from the specified path.
- **Handling Missing Values:**
  - Drops rows with any remaining missing values.
- **Data Splitting:**
  - Splits all the dataset into feature matrix X and target variable y.
  - Splits X, y to: X_train, X_test, y_train, y_test, utilizes sklearn.

Results:

- extract 36,177 random training samples for analysis.
- extract 9045 random test samples for analysis.

Accuracy	0.7633
Precision	1.0000
Recall	0.0281
F1 Score	0.0547

Data processing

Second try:

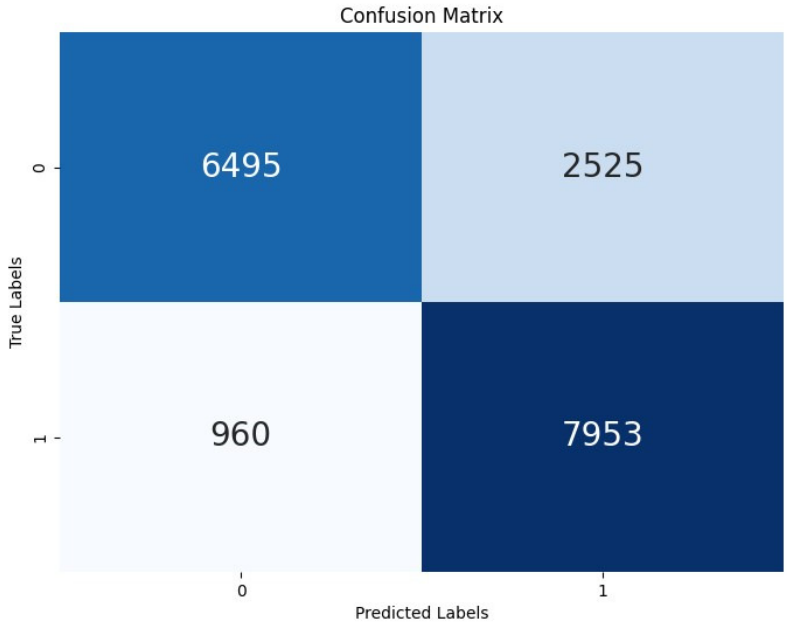
- Upon recognizing the presence of columns with continuous values in the dataset, which are incompatible with the fit function of my model and unbalanced distribution, I determined it necessary to undertake the following steps:
- **Discretization Of Continuous Values:**
  - Utilizes Pandas cut function for discretization, ensuring compatibility with the probabilistic assumptions of the Naive Bayes model.
- **Handling Imbalanced Data:**
  - Applies balance_sample to address more even distribution of classes.
- **Data Splitting:**
  - Splits all the dataset into feature matrix X and target variable y.
  - Splits X, y to: X_train, X_test, y_train, y_test with test size=0.2, Utilizes train_test_split() function of sklearn.

Results:

- extract 17,933 random training samples for analysis.
- extract 4483 random test samples for analysis.

Accuracy	0.8056
Precision	0.7590
Recall	0.8922
F1 Score	0.8202

```
accuracy:  0.8056655328165951
precision:  0.7590188967360183
recall:  0.8922921575227196
f1:  0.820277448300758
```



Comparing The Models:

Naïve Bayes

```
accuracy:  0.8056655328165951
precision:  0.7590188967360l8
recall:  0.8922921575227196
f1:  0.820277448300758
```

KNN

```
K:  63
accuarcy:  0.7777777777777778
precision:  0.8333333333333334
recall:  0.75
f1:  0.7894736842105262
```

Naïve Bayes generally performs better in accuracy, recall, and overall effectiveness, while KNN stands out for precision. Although there are differences, neither model shows a big advantage over the other in the evaluation scores. It seems the decision on which model is better depends on the specific task you need to accomplish. Naïve Bayes is good for a well-rounded performance, while KNN is especially accurate when precision is crucial.

Summary:

The analysis indicates that both Naïve Bayes and KNN may not be the most suitable models for the dataset at hand. Exploring enhanced data processing techniques, particularly addressing outlier handling, could potentially yield more favorable results. This insight underscores the importance of refining the data preprocessing pipeline to enhance model performance and achieve more optimal outcomes..