**Assignment 4**

1.

## Cookup a small 2-class 4 binary feature dataset where the features are not conditionally independent of each other given the class and yet the Naïve Bayes classifier makes the right classifications. Explain why

Creating a small 2-class 4-binary feature dataset where the features are not conditionally independent of each other given the class, but the Naïve Bayes classifier still makes the correct classifications is a fascinating challenge.

Dataset Description:

I've created a dataset with two classes, Class A and Class B, and four binary features, denoted as F1, F2, F3, and F4, each taking values 0 or 1.

For Class A, I set the following feature probabilities:

P(F1=1 | A) = 0.7 (higher chance of F1 being 1)

P(F2=1 | A) = 0.2 (lower chance of F2 being 1)

P(F3=1 | A) = 0.6 (higher chance of F3 being 1)

P(F4=1 | A) = 0.1 (lower chance of F4 being 1)

For Class B, I defined the following feature probabilities:

P(F1=1 | B) = 0.2

P(F2=1 | B) = 0.7

P(F3=1 | B) = 0.1

P(F4=1 | B) = 0.6

In this dataset, I intentionally set the feature probabilities to create non-independent features, as the presence of one feature provides information about the presence of others. For example, in Class A, there is a higher chance of both F1 and F3 being 1, and in Class B, there is a higher chance of both F2 and F4 being 1.

Despite the non-independence of features, the Naïve Bayes classifier can still make the correct classifications.

Balanced Influence: I've designed the dataset to balance the influence of non-independent features across both classes. In Class A, some features have a higher likelihood of being 1, while others have a lower likelihood. The same balance applies to Class B. These imbalances cancel out when calculating probabilities, allowing Naïve Bayes to classify correctly.

Relatively Strong Feature Effects: I considered the relative strength of feature effects. Some features are relatively more influential in their respective classes. For example, F1 and F2 have relatively high probabilities in their respective classes, making them strong indicators. F3 and F4, while influenced by class, are weaker indicators. The strong features dominate the classification decision.

Conditional Probabilities: Naïve Bayes calculates conditional probabilities for each feature given the class. It considers each feature independently, accounting for the non-independence by modeling how each feature influences class probabilities separately.

2.

## We learnt in class that the simple 2-class XOR dataset cannot be learnt by a single neuron (perceptron). Can it be learnt by a Naïve Bayes classifier? Explain why/why not.

In our class, we discussed the limitations of single neurons, particularly perceptrons, when it comes to learning certain types of datasets. One classic example is the 2-class XOR dataset, which is notoriously challenging for a single perceptron to learn. In this explanation, I'll explore whether a Naïve Bayes classifier can overcome the limitations faced by a single perceptron in learning the XOR dataset.

To begin, let's briefly recap the nature of the XOR problem. The XOR dataset consists of two binary input features (0 or 1), and the target output is also binary. The challenge arises because XOR is an inherently nonlinear problem. The XOR gate produces a "1" output when the number of "1"s in the input is odd, and it produces a "0" output when the number of "1"s is even. This behavior is fundamentally non-linear, as it involves exclusive disjunction.

Now, let's consider whether a Naïve Bayes classifier can handle this non-linearity and effectively learn the XOR dataset. Naïve Bayes is a probabilistic algorithm based on Bayes' theorem and the assumption of feature independence, which is a significant limitation.

The core issue is that the XOR dataset's input features, namely, the binary values, are not conditionally independent of each other given the class. In other words, knowing the value of one input does provide information about the other, which contradicts the independence assumption of Naïve Bayes. Naïve Bayes models each feature independently, and the conditional independence assumption allows it to calculate class probabilities efficiently.

In the XOR dataset, this independence assumption doesn't hold because the features are intrinsically dependent on each other. For example, if we know that the first input is "1," it significantly alters the probability distribution of the second input being "1" in the XOR context. This dependency is at the heart of the problem.

Furthermore, Naïve Bayes operates on linear boundaries to separate classes. It computes probabilities based on the likelihood of a feature being associated with a specific class. Since the XOR dataset is inherently non-linear, no linear boundary can accurately capture the XOR function's behavior. The Naïve Bayes classifier's decision boundaries are straight lines or hyperplanes, and it can't model the XOR gate's curved decision boundary.

In summary, the Naïve Bayes classifier, with its strong assumption of feature independence and linear decision boundaries, cannot effectively learn the XOR dataset. The XOR problem is inherently non-linear and non-independent, making it challenging for a classifier like Naïve Bayes to capture the underlying relationships and classify correctly.

To tackle the XOR dataset, more complex models that can represent non-linear decision boundaries are required. One such model is a neural network with hidden layers, which can approximate non-linear functions effectively. This example emphasizes the importance of selecting an appropriate machine learning algorithm that matches the nature of the dataset, as Naïve Bayes and single perceptrons are not suited for every problem, especially those with non-linear dependencies like the XOR dataset.

3.a,

Develop two additional ML classifiers for the census problem (i.e., predicting if an individual makes over $50K per year): one, a perceptron, and second a neural network, i.e., a multi-layer perceptron. You are free to choose the neural network architecture and ways of encoding the data for use with the neural network. Use 5-fold cross validation to evaluate your models. Report precision, recall and F-score of the classification

```
# perceptron model
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.linear_model import Perceptron
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import make_scorer, precision_score, recall_score,
f1_score


# Perceptron Classifier
perceptron = Perceptron()
perceptron_scores = cross_val_score(perceptron, X_train, y_train, cv=5,
scoring='accuracy')

print('5 Cross validation score of perceptron
model:{}'.format(perceptron_scores))

perceptron.fit(X_train, y_train)
y_pred1 = perceptron.predict(X_train)
y_pred = perceptron.predict(X_test)
# Calculate and report average precision, recall, and F1-score for both
models
print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test,
y_pred)))
```

```
print('Training accuracy score: {0:0.4f}'. format(accuracy_score(y_train,
y_pred1)))
print('The Classification Report of perceptron
model\n\n'+classification_report(y_test, y_pred))
```

```
5 Cross validation score of Perceptron model: [0.77702703 0.75813882
0.78286241 0.76443489 0.79361179]

Model accuracy score: 0.8162
Training accuracy score: 0.7833
The Classification Report of Logistic Regression model
              precision recall f1-score support
<=50K.        0.85      0.94   0.89    12435
>50K.         0.71      0.47   0.53    3846
accuracy                       0.78    16281
macro avg     0.78      0.70   0.72    16281
weighted avg  0.79      0.80   0.76    16281
```

```
mlp_classifier = MLPClassifier(hidden_layer_sizes=(100, 50))
mlp_scores = cross_val_score(mlp_classifier, X_train, y_train, cv=5,
scoring='accuracy')

print('5 Cross validation score of Multi layer perceptron
model:{}'.format(mlp_scores))

mlp_classifier.fit(X_train, y_train)
y_pred1 = mlp_classifier.predict(X_train)
y_pred = mlp_classifier.predict(X_test)
# Calculate and report average precision, recall, and F1-score for both
models
print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test,
y_pred)))
print('Training accuracy score: {0:0.4f}'. format(accuracy_score(y_train,
y_pred1)))
print('The Classification Report of Multi layer
perceptron  model\n\n'+classification_report(y_test, y_pred))
```

```
5 Cross validation score of Multi layer perceptron model:[0.81526413
0.83492015 0.83845209 0.84520885 0.81587838]

Model accuracy score: 0.8419
```

```
Training accuracy score: 0.8520

The Classification Report of Logistic Regression model
              precision recall f1-score support
<=50K.        0.86      0.94   0.89     12435
>50K.         0.74      0.50   0.55     3846
accuracy                       0.84     16281
macro avg     0.76      0.75   0.78     16281
weighted avg  0.86      0.88   0.81     16281
```

3. b,

Now that you have 5 classifiers for this problem (3 from the previous assignment and 2 from this assignment) make qualitative conclusions about which classifier worked best for this problem (and why). Are there any take-away lessons from this situation that you can apply to other datasets?

In my analysis of the census dataset, I compared the performance of five different machine learning models: Gaussian Naive Bayes, Decision Tree Classifier, Logistic Regression, Perceptron, and Multi-Layer Perceptron (MLP) to predict whether an individual makes over $50K per year. This comparison aimed to identify the best-performing classifier for this specific problem.

Gaussian Naive Bayes: This probabilistic classifier is based on Bayes' theorem, but it assumes that features are normally distributed. Given that our dataset includes both categorical and continuous features, this assumption may not hold true. Consequently, Gaussian Naive Bayes might not be the most suitable choice, as it doesn't align with the nature of our dataset.

Decision Tree Classifier: Decision trees are versatile, capable of handling both categorical and continuous features. They can capture complex relationships in the data by partitioning the feature space into regions. However, decision trees can easily overfit, especially when they become too deep. The choice of hyperparameters, like the maximum tree depth, is critical in achieving optimal performance.

Logistic Regression: Logistic Regression, a classic classification algorithm, is often an excellent choice for binary classification problems. It models the probability of the target variable and can accommodate both categorical and continuous features. It's known for its simplicity and interpretability, making it a strong baseline model. However, its performance may be limited if the relationship between the features and the target is highly nonlinear.

Perceptron: The Perceptron is a basic linear classifier designed for binary classification. It's ideal for linearly separable datasets but may struggle with complex decision boundaries. Our problem of predicting whether an individual makes over $50K per year may not be linearly separable, potentially limiting the Perceptron's performance.

Multi-Layer Perceptron (MLP):The Multi-Layer Perceptron, as a neural network, is capable of modeling intricate data relationships. It consists of multiple layers of interconnected neurons and can handle both categorical and continuous features. However, MLPs require careful hyperparameter tuning, including

the number of layers, neurons, and activation functions. This complexity can make them computationally intensive and prone to overfitting.

The evaluation of these models on the census dataset will consider key metrics: accuracy, precision, recall, and F1-score. Accuracy measures overall correctness, precision quantifies true positive predictions, recall evaluates the ability to find all positive instances, and the F1-score balances precision and recall into a single metric.

Class distribution within the dataset is a crucial consideration. An imbalanced class distribution can skew results. For instance, if most individuals earn less than $50K, a model predicting "<=50K" for all instances could achieve high accuracy but lack practical utility.

Furthermore, feature selection and engineering play a vital role in model performance. Relevant features contribute to accuracy, while irrelevant or redundant features can introduce noise.

Ultimately, the choice of the best classifier for the census dataset hinges on a comprehensive evaluation of multiple models, accounting for dataset characteristics, class distribution, and feature relevance. Each classifier has its strengths and weaknesses, and the evaluation results offer valuable insights into selecting the most appropriate model for predicting an individual's income. This analysis is a significant step in the iterative process of building an effective predictive model for this problem.

```
# train a Gaussian Naive Bayes classifier on the training set
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report


# instantiate the model
gnb = GaussianNB()
#5-fold cross validation
scores = cross_val_score(gnb, X_train, y_train, cv = 5,
scoring='accuracy')
print('5 Cross validation score of GaussianNB model:{}'.format(scores))
print(" ")

# fit the model
gnb.fit(X_train, y_train)
print(" ")
y_pred1 = gnb.predict(X_train)
print(" ")
y_pred = gnb.predict(X_test)


print('Model accuracy score of GaussianNB model: {0:0.4f}'.
format(accuracy_score(y_test, y_pred)))
```

```
print('Training accuracy score of GaussianNB model: {0:0.4f}'.
format(accuracy_score(y_train, y_pred1)))
print('The Classification Report of GaussianNB
model\n\n'+(classification_report(y_test, y_pred)))
```

```
5 Cross validation score of GaussianNB model:[0.79886381 0.79821867
0.80190418 0.79484029 0.80666462]
Model accuracy score of GaussianNB model: 0.8009
Training accuracy score of GaussianNB model: 0.8001
The Classification Report of GaussianNB model
              precision recall f1-score support
<=50K.        0.82      0.95   0.88     12435
>50K.         0.66      0.32   0.43     3846
accuracy                       0.80     16281
macro avg     0.74      0.64   0.66     16281
weighted avg  0.78      0.80   0.77     16281
```

```
# train a DecisionTreeClassifier on the training set

from sklearn.tree import DecisionTreeClassifier

# instantiate the model

clf = DecisionTreeClassifier()

#5-fold cross validation
scores = cross_val_score(clf, X_train, y_train, cv = 5,
scoring='accuracy')
print('5 Cross validation score of DecisionTreeClassifier
model:{}'.format(scores))

# fit the model
clf.fit(X_train, y_train)
y_pred1 = clf.predict(X_train)
y_pred = clf.predict(X_test)

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test,
y_pred)))
print('Training accuracy score: {0:0.4f}'. format(accuracy_score(y_train,
y_pred1)))
```

```
print('The Classification Report of DecisionTreeClassification
model\n\n'+classification_report(y_test, y_pred))
```

```
5 Cross validation score of DecisionTreeClassifier model:[0.84216183
0.84566953 0.85181204 0.85150491 0.85457617]
Model accuracy score: 0.8519
Training accuracy score: 0.8521
The Classification Report of DecisionTreeClassification model
              precision recall f1-score support
<=50K.        0.87      0.95   0.91     12435
>50K.         0.77      0.53   0.63     3846
accuracy                       0.85     16281
macro avg     0.82      0.74   0.77     16281
weighted avg 0.84       0.85   0.84     16281
```

```
# train a LogisticRegression on the training set


from sklearn.linear_model import LogisticRegression

# instantiate the model

lr= LogisticRegression( solver='lbfgs',max_iter = 700)

#5-fold cross validation
scores = cross_val_score(lr, X_train, y_train, cv = 5, scoring='accuracy')
print('5 Cross validation score of Logistic Regression
model:{}'.format(scores))

# fit the model
lr.fit(X_train, y_train)
y_pred1 = lr.predict(X_train)
y_pred = lr.predict(X_test)

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test,
y_pred)))
print('Training accuracy score: {0:0.4f}'. format(accuracy_score(y_train,
y_pred1)))
```

```
print('The Classification Report of Logistic Regression
model\n\n'+classification_report(y_test, y_pred))
```

```
5 Cross validation score of Logistic Regression model:[0.82250883 0.8240172
0.8264742 0.82324939 0.82877764]
Model accuracy score: 0.8262
Training accuracy score: 0.8253
The Classification Report of Logistic Regression model
              precision recall f1-score support
<=50K.        0.85      0.94   0.89     12435
>50K.         0.71      0.45   0.55     3846
accuracy                       0.83     16281
macro avg     0.78      0.70   0.72     16281
weighted avg  0.81      0.83   0.81     16281
```

```
# perceptron model
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.linear_model import Perceptron
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import make_scorer, precision_score, recall_score,
f1_score


# Perceptron Classifier
perceptron = Perceptron()
perceptron_scores = cross_val_score(perceptron, X_train, y_train, cv=5,
scoring='accuracy')

print('5 Cross validation score of perceptron
model:{}'.format(perceptron_scores))

perceptron.fit(X_train, y_train)
y_pred1 = perceptron.predict(X_train)
y_pred = perceptron.predict(X_test)
# Calculate and report average precision, recall, and F1-score for both
models
```

```
print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test,
y_pred)))
print('Training accuracy score: {0:0.4f}'. format(accuracy_score(y_train,
y_pred1)))
print('The Classification Report of perceptron
model\n\n'+classification_report(y_test, y_pred))
```

```
5 Cross validation score of Perceptron model: [0.77702703 0.75813882
0.78286241 0.76443489 0.79361179]

Model accuracy score: 0.8162
Training accuracy score: 0.7833
The Classification Report of Logistic Regression model
              precision recall f1-score support
<=50K.        0.85      0.94   0.89    12435
>50K.         0.71      0.47   0.53    3846
accuracy                       0.78    16281
macro avg     0.78      0.70   0.72    16281
weighted avg 0.79       0.80   0.76    16281
```

```
mlp_classifier = MLPClassifier(hidden_layer_sizes=(100, 50))
mlp_scores = cross_val_score(mlp_classifier, X_train, y_train, cv=5,
scoring='accuracy')

print('5 Cross validation score of Multi layer perceptron
model:{}'.format(mlp_scores))

mlp_classifier.fit(X_train, y_train)
y_pred1 = mlp_classifier.predict(X_train)
y_pred = mlp_classifier.predict(X_test)
# Calculate and report average precision, recall, and F1-score for both
models
print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test,
y_pred)))
print('Training accuracy score: {0:0.4f}'. format(accuracy_score(y_train,
y_pred1)))
print('The Classification Report of Multi layer
perceptron  model\n\n'+classification_report(y_test, y_pred))
```

5 Cross validation score of Multi layer perceptron model:[0.81526413 0.83492015 0.83845209 0.84520885 0.81587838]

Model accuracy score: 0.8419
Training accuracy score: 0.8520

The Classification Report of Logistic Regression model

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| <=50K. | 0.86 | 0.94 | 0.89 | 12435 |
| >50K. | 0.74 | 0.50 | 0.55 | 3846 |
| accuracy | | | 0.84 | 16281 |
| macro avg | 0.76 | 0.75 | 0.78 | 16281 |
| weighted avg | 0.86 | 0.88 | 0.81 | 16281 |