CSE 5819 Assignment #5

by: Aayushi Verma (uef24001)

This assignment is organized as follows:

1 Part 1: ChatGPT Self-Learning (20pts)

You can start from the following prompts, but you need to create subsequent questions that attempt to understand the basic concepts of machine learning.

1.1 What is Bayes' theorem?

Bayes' theorem is a fundamental result in probability theory that describes how to update the probability of a hypothesis based on new evidence. It is given by the formula:

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)}$$

where:

- $P(H|E)$ is the posterior probability, the probability of the hypothesis H given the evidence E.
- $P(E|H)$ is the likelihood, the probability of the evidence given that the hypothesis is true.
- $P(H)$ is the prior probability, the initial probability of the hypothesis before seeing the evidence.
- $P(E)$ is the marginal likelihood or evidence, the total probability of observing the evidence.

1.2 What is naive Bayes?

Naive Bayes is a classification algorithm based on Bayes' theorem, with the assumption that the features are conditionally independent given the class label. The formula for the probability of a class C_k given a feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is:

$$P(C_k|\mathbf{x}) = \frac{P(C_k) \prod_{i=1}^{n} P(x_i|C_k)}{P(\mathbf{x})}$$

Where:

- $P(C_k|\mathbf{x})$ is the posterior probability of class C_k given the features.
- $P(C_k)$ is the prior probability of class C_k.
- $P(x_i|C_k)$ is the likelihood of feature x_i given class C_k.
- $P(\mathbf{x})$ is the marginal likelihood or evidence, the probability of the features.

The "naive" assumption is that the features x_i are independent of each other, which simplifies the computation of the joint probability.

1.3 Can you provide an example of naïve Bayes?

An example of a Naive Bayes classifier can be illustrated with a simple text classification task, such as spam detection. Suppose we want to classify emails as either "spam" or "not spam" based on the presence of certain words.

Let's assume we have the following training data:

- **Spam Emails:**
  - "Free money now"
  - "Exclusive offer just for you"
- **Not Spam Emails:**
  - "Meeting at 10 AM"
  - "Project deadline next week"

From this training data, we can calculate the prior probabilities:

$$P(\text{Spam}) = \frac{2}{4} = 0.5 \quad \text{and} \quad P(\text{Not Spam}) = \frac{2}{4} = 0.5$$

Next, we identify the features (words) and calculate their likelihoods. For example:

- $P(\text{"money"}|\text{Spam}) = \frac{1}{2}$
- $P(\text{"money"}|\text{Not Spam}) = 0$
- $P(\text{"meeting"}|\text{Spam}) = 0$
- $P(\text{"meeting"}|\text{Not Spam}) = \frac{1}{2}$

Now, suppose we receive a new email: "Get your free money now." We want to classify it as either spam or not spam.

Using Bayes' theorem, we compute:

$P(\text{Spam}|\text{"Get your free money now"}) \propto P(\text{Spam}) \cdot P(\text{"Get"}|\text{Spam}) \cdot P(\text{"your"}|\text{Spam}) \cdot P(\text{"free"}|\text{Spam}) \cdot P(\text{"money"}|\text{Spam}) \cdot P(\text{"no}$

Assuming that the other words in the email have a probability of 1 when the email is spam, we can simplify to:

$$P(\text{Spam}|\text{"Get your free money now"}) \propto 0.5 \cdot 1 \cdot 1 \cdot 1 \cdot \frac{1}{2} \cdot 1 = 0.25$$

Similarly, we compute:

$P(\text{Not Spam}|\text{"Get your free money now"}) \propto P(\text{Not Spam}) \cdot P(\text{"Get"}|\text{Not Spam}) \cdot P(\text{"your"}|\text{Not Spam}) \cdot P(\text{"free"}|\text{Not Spam}) \cdot P($

$$P(\text{Not Spam}|\text{"Get your free money now"}) \propto 0.5 \cdot 1 \cdot 1 \cdot 1 \cdot 0 \cdot 1 = 0$$

Since $P(\text{Spam}|\text{"Get your free money now"}) > P(\text{Not Spam}|\text{"Get your free money now"})$, we classify the email as spam.

1.4 How to run k-nearest neighbor classifier?

To run a k-nearest neighbor (KNN) classifier, follow these steps:

1. **Import Required Libraries:** You will need libraries for data manipulation, model training, and evaluation. Common libraries include NumPy, Pandas, and scikit-learn.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
```

2. **Load the Dataset:** Load your dataset into a Pandas DataFrame. For example, using the Iris dataset:

```
data = pd.read_csv('iris.csv')  % Load the dataset
```

3. **Preprocess the Data:** Split the dataset into features and target variables, and then divide it into training and testing sets.

```
X = data.drop('species', axis=1)  % Features
y = data['species']                % Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

4. **Create the KNN Classifier:** Initialize the KNN classifier with a specified value for k, which represents the number of nearest neighbors to consider.

```
k = 3
knn = KNeighborsClassifier(n_neighbors=k)
```

5. **Train the Model:** Fit the KNN classifier to the training data.

```
knn.fit(X_train, y_train)
```

6. **Make Predictions:** Use the trained model to make predictions on the test set.

```
y_pred = knn.predict(X_test)
```

7. **Evaluate the Model:** Assess the performance of the classifier using metrics like accuracy and confusion matrix.

```
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)

print('Accuracy:', accuracy)
print('Confusion Matrix:\n', confusion)
```

1.5 How to run k-nearest neighbor regression?

To run a k-nearest neighbor (KNN) regression, follow these steps:

1. **Import Required Libraries:** Import necessary libraries for data handling, model training, and evaluation. Commonly used libraries include NumPy, Pandas, and scikit-learn.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

2. **Load the Dataset:** Load your dataset into a Pandas DataFrame. For example, you could use a dataset like the Boston housing dataset.

```
data = pd.read_csv('boston_housing.csv')  % Load the dataset
```

3. **Preprocess the Data:** Split the dataset into features and target variables, and then divide it into training and testing sets.

```
X = data.drop('MEDV', axis=1)  % Features
y = data['MEDV']                % Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

4. **Create the KNN Regressor:** Initialize the KNN regressor with a specified value for k, which represents the number of nearest neighbors to consider.

```
k = 3
knn_regressor = KNeighborsRegressor(n_neighbors=k)
```

5. **Train the Model:** Fit the KNN regressor to the training data.

```
knn_regressor.fit(X_train, y_train)
```

6. **Make Predictions:** Use the trained model to make predictions on the test set.

```
y_pred = knn_regressor.predict(X_test)
```

7. **Evaluate the Model:** Assess the performance of the regressor using metrics like mean squared error (MSE) and R^2 score.

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print('Mean Squared Error:', mse)
print('R^2 Score:', r2)
```

1.6 How to choose appropriate value for k in k-nearest neighbor classifier?

Choosing the appropriate value for k in a k-nearest neighbor (KNN) classifier is crucial for model performance. Here are some common methods to determine the best k:

1. **Domain Knowledge:** If you have prior knowledge about the dataset, it can provide insights into an appropriate k value based on the problem context.
2. **Cross-Validation:** Use cross-validation to evaluate model performance for different values of k. The process involves:
   - Splitting the dataset into k subsets (folds).
   - Training the model on $k-1$ folds and testing it on the remaining fold.
   - Repeating this process for each fold and averaging the performance metrics (e.g., accuracy).
3. **Odd vs. Even Values:** For binary classification problems, choose an odd value of k to avoid ties in voting. For multiclass problems, either odd or even values can be considered.
4. **Error Rate Analysis:** Plot the error rate for different k values.
   - Start with $k = 1$ and incrementally increase k to observe the trend.
   - Look for a point where increasing k no longer significantly decreases the error rate. This can indicate a good balance between bias and variance.
5. **Grid Search:** Utilize a grid search approach to systematically evaluate a range of k values along with other hyperparameters. This method is often coupled with cross-validation for better reliability.
6. **Considerations for Large Datasets:** For larger datasets, smaller values of k might be preferred as they allow the model to capture local patterns. However, for noisy datasets, larger k values can help to smooth out the noise.

In summary, the optimal k value is typically found through experimentation and evaluation, balancing the trade-off between model complexity and generalization.

1.7 How to evaluate a supervised machine learning model?

Evaluating a supervised machine learning model is crucial to understand its performance and generalization ability. Here are common steps and metrics used for evaluation:

1. **Split the Dataset:** Divide the dataset into training and testing sets, typically using an 80/20 or 70/30 split. Optionally, create a validation set for hyperparameter tuning.
2. **Cross-Validation:** Implement k-fold cross-validation to ensure that the model's performance is stable across different subsets of the data. In k-fold cross-validation:
   - The dataset is divided into k folds.
   - The model is trained k times, each time using $k-1$ folds for training and 1 fold for validation.
   - The average performance across all folds is calculated.
3. **Performance Metrics:** Depending on the type of supervised learning problem (classification or regression), use appropriate metrics:
   - **For Classification:**
     - **Accuracy:**
     $$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Predictions}}$$
     - **Precision:**
     $$\text{Precision} = \frac{TP}{TP + FP}$$
     - **Recall (Sensitivity):**
     $$\text{Recall} = \frac{TP}{TP + FN}$$
     - **F1 Score:**
     $$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$
     - **Confusion Matrix:** A matrix that summarizes true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).
   - **For Regression:**
     - **Mean Absolute Error (MAE):**
     $$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$
     - **Mean Squared Error (MSE):**
     $$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$
     - **Root Mean Squared Error (RMSE):**
     $$\text{RMSE} = \sqrt{\text{MSE}}$$
     - **R-squared (R^2):** A measure of how well the independent variables explain the variability of the dependent variable.
     $$R^2 = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}}$$
     where SS_{res} is the sum of squares of residuals and SS_{tot} is the total sum of squares.
4. **Visualizations:** Utilize visualizations such as:
   - ROC curves for binary classification, plotting true positive rate against false positive rate.
   - Precision-Recall curves to evaluate the trade-off between precision and recall.
   - Residual plots to analyze the difference between actual and predicted values in regression tasks.
5. **Interpretability:** Assess the model's interpretability using techniques such as feature importance scores, SHAP values, or LIME, to understand which features most influence predictions.
6. **Model Comparison:** Compare the performance of different models using the metrics calculated above to determine which model performs best for the given task.
7. **Hyperparameter Tuning:** Use techniques such as grid search or random search with cross-validation to optimize hyperparameters and improve model performance.

1.8 What are common metrics used to evaluate a regression model?

Common metrics used to evaluate a regression model include the following:

1. **Mean Absolute Error (MAE):** This metric measures the average absolute difference between the predicted values and the actual values. It is expressed as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

   where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of observations.

2. **Mean Squared Error (MSE):** This metric measures the average of the squared differences between the predicted and actual values. It penalizes larger errors more than smaller ones and is expressed as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

3. **Root Mean Squared Error (RMSE):** This is the square root of the MSE and provides error metrics in the same units as the target variable, making it easier to interpret:

$$\text{RMSE} = \sqrt{\text{MSE}}$$

4. **R-squared (R^2):** This metric indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. It is calculated as:

$$R^2 = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}}$$

   where SS_{res} is the sum of squares of residuals, and SS_{tot} is the total sum of squares.

5. **Adjusted R-squared:** This metric adjusts the R^2 value for the number of predictors in the model, providing a more accurate measure when comparing models with different numbers of predictors.

6. **Mean Absolute Percentage Error (MAPE):** This metric expresses the error as a percentage of the actual values, making it useful for understanding error in relative terms:

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

7. **Explained Variance Score:** This metric measures the proportion of variance explained by the model. A higher score indicates better model performance.

8. **Quantile Loss:** This metric is particularly useful for quantile regression, measuring how well the model predicts specific quantiles of the target variable.

1.9 What are common metrics used to evaluate a classifier?

Common metrics used to evaluate a classifier include the following:

1. **Accuracy:** This metric measures the proportion of correctly classified instances out of the total instances. It is calculated as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

   where TP is true positives, TN is true negatives, FP is false positives, and FN is false negatives.

2. **Precision:** Precision measures the proportion of true positive predictions among all positive predictions made. It is calculated as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

3. **Recall (Sensitivity or True Positive Rate):** Recall measures the proportion of true positives identified out of all actual positive instances. It is calculated as:

$$\text{Recall} = \frac{TP}{TP + FN}$$

4. **F1 Score:** The F1 Score is the harmonic mean of precision and recall, providing a single score that balances both metrics. It is calculated as:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

5. **Receiver Operating Characteristic (ROC) Curve:** The ROC curve is a graphical representation of the true positive rate against the false positive rate at various threshold settings. It helps visualize the trade-off between sensitivity and specificity.

6. **Area Under the ROC Curve (AUC-ROC):** AUC measures the area under the ROC curve. A value of 1 indicates perfect classification, while a value of 0.5 suggests no discriminative ability.

7. **Confusion Matrix:** The confusion matrix provides a detailed breakdown of the classifier's performance, showing true positives, true negatives, false positives, and false negatives.

8. **Specificity (True Negative Rate):** Specificity measures the proportion of true negatives identified out of all actual negative instances. It is calculated as:

$$\text{Specificity} = \frac{TN}{TN + FP}$$

9. **Logarithmic Loss (Log Loss):** Log loss measures the performance of a classifier by penalizing incorrect classifications based on the predicted probability. It is defined as:

$$\text{Log Loss} = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

   where y_i is the true label and p_i is the predicted probability.

10. **Matthews Correlation Coefficient (MCC):** The MCC is a balanced measure that takes into account all four confusion matrix categories (TP, TN, FP, FN). It is calculated as:

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

2 Problems

2.1 Question One

The following shows a history of customers with their incomes, age groups, and an attribute called Have_iPhone indicating whether they have an iPhone. We also indicate whether they will buy an iPad or not in the last column. We want to use Naive Bayes to predict whether a new customer will buy an iPad or not. Consider a new young customer whose income is medium and who has an iPhone. Please predict whether this new customer will buy an iPad.

No.	Income	Age	Have_iPhone	Buy_iPad
1	high	young	yes	yes
2	high	old	yes	yes
3	medium	young	no	yes
4	high	old	no	yes
5	medium	young	no	no
6	medium	young	no	no
7	medium	old	no	no
8	medium	old	no	no

For each attribute: medium income ($I = m$), young age ($A = y$), have iPhone ($iPhone = y$) for our new customer, let's calculate the probability for each value of buying an iPad (yes, no). We do this by counting the number of matching attribute values for each possible value of 'Buy_iPad' (yes, no).

$$\begin{aligned} P(I = m|iPad = y) &= \frac{1}{4} \\ P(I = m|iPad = n) &= \frac{4}{4} = 1 \end{aligned} \tag{1}$$

$$\begin{aligned} P(A = y|iPad = y) &= \frac{2}{4} = \frac{1}{2} \\ P(A = y|iPad = n) &= \frac{2}{4} = \frac{1}{2} \end{aligned} \tag{2}$$

$$\begin{aligned} P(iPhone = y|iPad = y) &= \frac{2}{4} = \frac{1}{2} \\ P(iPhone = y|iPad = n) &= \frac{0}{4} = 0 \end{aligned} \tag{3}$$

We also calculate the total probability of 'Buy_iPad':

$$\begin{aligned} P(iPad = y) &= \frac{4}{8} = \frac{1}{2} \\ P(iPad = n) &= \frac{4}{8} = \frac{1}{2} \end{aligned} \tag{4}$$

Finally we use Bayes' rule and for each value of 'Buy_iPad', calculate the class probability, given the customer's attributes:

$$\begin{aligned} P(iPad = y|\text{attributes}) &= P(iPad = y) \cdot P(I = m|iPad = y) \cdot P(A = y|iPad = y) \cdot P(iPhone = y|iPad = y) \\ &= \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot \frac{1}{2} \\ &= 0.03125 \end{aligned} \tag{5}$$

$$\begin{aligned} P(iPad = n|\text{attributes}) &= P(iPad = n) \cdot P(I = m|iPad = n) \cdot P(A = y|iPad = n) \cdot P(iPhone = y|iPad = n) \\ &= \frac{1}{2} \cdot 1 \cdot \frac{1}{2} \cdot 0 \\ &= 0 \end{aligned} \tag{6}$$

Finally, we compare the calculated class probabilities (Eq. 5 vs 6) given the attributes:

$$[P(iPad = n|\text{attributes}) = 0.03125] > [P(iPad = n|\text{attributes}) = 0] \tag{7}$$

$\therefore$ This new customer with the given attributes: medium income ($I = m$), young age ($A = y$), have iPhone ($iPhone = y$), **is likely** to buy an iPad with 0.03125 probability.

2.2 Question Two

You have been asked to develop a classification model for diagnosing whether a patient is infected with a certain disease. To help you construct the models, your collaborator has provided you with a small training set (N=10 individuals) with equal number of positive and negative examples. You tried several approaches and found two most promising models, C1 and C2. The outputs of the models in terms of predicting whether each of the training examples belong to the "positive(+)" class are summarized in the table below. The first row shows the probability a training example belongs to the positive class according to the classifier C1, while the second row shows the same information for classifier C2. The last row indicates the true class labels of the 10 training examples.

$P(y=+ \mid C1)$	0.1	0.15	0.2	0.3	0.31	0.4	0.62	0.77	0.81	0.95
$P(y=+ \mid C2)$	0.25	0.49	0.05	0.35	0.66	0.6	0.7	0.65	0.55	0.99
y	−	+	−	−	+	−	+	+	−	+

For each model, we will evaluate different thresholds within the range of [0,1], and a sample with probability P(y=+ — C1) (or P(y=+ — C2)) that is lower than this threshold will be estimated as -, or + if greater than this threshold. By varying the thresholds (referred to the lecture slide on ROC), you can study the model performance and draw the ROC. (you can either use code or hand-calculating, but if you use code, you need to show the calculation process.)

a. Draw the corresponding ROC curves for each classifier on the same plot.

b. Which classifier can be considered better? Why?