Assignment – 2 (Chapter3-4)

Part A – Theory Questions

Q1(a) Explain the Holdout Method.

Ans: In the Holdout method, the available dataset is partitioned into two mutually exclusive subsets: a training set and a testing set. Typically, 70–80% of the data is used for training and the remaining 20–30% is used for testing. The classifier is trained on the training set and evaluated on the testing set, ensuring that the model's performance is estimated on unseen data. This method is simple but has high variance because the evaluation depends heavily on how the data is split.

Q1(b) Apply holdout to a classification dataset.

Ans: To apply holdout, the data is first randomized. Then an 80/20 split is performed. The model is trained on the 80% training portion and predictions are made on the 20% test portion. A confusion matrix is generated to analyze TP, FP, FN, and TN, after which metrics like accuracy, precision, and recall are computed.

Q2(a) Why is cross-validation more reliable than holdout?

Ans: Cross-validation, especially k-fold CV, reduces variability by training and testing the model on multiple different partitions of the dataset. All samples are used for both training and validation across folds, which results in more stable and generalizable performance estimates compared to the
single-split nature of holdout.

Q2(b) 5-fold CV on 500 samples.

Ans: The dataset is divided into five equal folds of 100 samples each. In each iteration, one fold is used as the validation set while the remaining four folds (400 samples) form the training set. This process repeats five times and the average of the five accuracy values provides the final performance estimate.

Q3(a) Explain bias–variance trade-off.

Ans: Bias refers to the error due to overly simplistic assumptions in the learning algorithm, leading to underfitting. Variance refers to error caused by sensitivity to fluctuations in the training data, leading to overfitting. An optimal model balances both by maintaining moderate complexity.

Q3(b) Influence on generalization.

Ans: A high-bias model performs poorly on both training and testing data, whereas a high-variance model performs well on training data but poorly on unseen data. The trade-off determines how well the model generalizes to new inputs.

Q4(a) Role of confusion matrix

Ans: The confusion matrix organizes classification predictions into TP, FP, FN, and TN. These values help compute several important evaluation metrics such as accuracy, precision, recall, specificity, and F1-score, providing a detailed understanding of model behavior.

Q4(b) Compute precision, recall, F1.

Ans: Precision = 50/60 = 0.83.

Recall = 50/55 = 0.91.

F1-score = 2*(0.83*0.91)/(0.83+0.91) 0.87.

Q5(a) 95% train and 75% test accuracy.

Ans: This indicates overfitting—the model has learned specific patterns and noise from the training data but fails to generalize. Solutions include regularization (L1/L2), pruning (for trees), reducing features, or using cross-validation.

Q5(b) Bias–variance in model selection.

Ans: When selecting a model, high-bias models like linear regression are suitable for simple relationships, while low-bias high-variance models like decision trees handle complex data but risk overfitting. Validation performance guides selection.

Q6(a) Good train performance, poor test.

Ans: The model is overfitted. It captures noise and specific patterns from training data. Techniques such as simplifying the architecture, adding regularization, or increasing training data help reduce overfitting.

Q6(b) Role of R-squared.

Ans: R^2 measures the proportion of variance in the dependent variable explained by the regression model. Values closer to 1 indicate that the model effectively captures variability, while low values imply poor fit.

Q7(a) Importance of feature engineering.

Ans: Feature engineering improves model performance by transforming raw data into meaningful features, reducing noise, enhancing signal, improving interpretability, and preventing overfitting through dimensionality reduction.

Q8(a) Steps in feature selection.

Ans: The process includes generating candidate feature subsets, evaluating them using an evaluation function, applying a stopping criterion (no further improvement), and validating the chosen subset.

Q8(b) Reducing 100 features to 10.

Ans: Compute correlation between each feature and the target, rank features by correlation magnitude, remove highly correlated redundant features, and select the top 10 that provide maximum predictive power.

Q9(a) Effects of high-dimensional data.

Ans: High-dimensional data increases computational cost, introduces noise, leads to sparse data, and causes the curse of dimensionality, making models prone to overfitting and difficult to interpret.

Q9(b) Correlation-based filter method.

Ans: Calculate correlation scores, rank features, eliminate redundant features with high inter-correlation, and choose top-ranking meaningful predictors.

Q10(a) Importance of feature engineering.

Ans: Proper feature engineering ensures cleaner inputs, better representation of relationships, and improved training efficiency. It reduces model complexity and enhances generalization.

Q10(b) Why fewer features can outperform many.

Ans: Extra features introduce noise and redundancy, leading to overfitting. Reducing to the most relevant features streamlines learning and improves accuracy on unseen data.

Part B – Lab Questions

Q1. An automobile company wants to predict a car's mpg value from its physical attributes.
Tasks:

(a) Load the dataset auto_mpg.csv and remove missing values.
(b) Identify predictor and target variables.
(c) Perform data splitting (80% train, 20% test).
(d) Fit a Linear Regression model and predict test outcomes.
(e) Evaluate the model using Mean Squared Error and R^2 score.
(f) Discuss: If the R^2 score = 0.85, what does it imply about model performance?

```python
# (a)
import pandas as pd

df = pd.read_csv("auto_mpg.csv")
df = df.dropna()
df = df.drop(columns=["car_name"])

# (b)
X = df.drop("mpg", axis=1)
y = df["mpg"]

# (c)
from sklearn.model_selection import train_test_split

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

# (d)
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# (e)
from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("MSE =", mse)
print("R²  =", r2)
```

✓ 0.0s

```
MSE = 10.710864418838373
R²  = 0.790150038676035
```

Q2. Exploring random sampling methods to estimate model uncertainty.
Tasks:

(a) From the btissue.csv data, extract only the feature columns (excluding labels).
(b) Using the resample() method, create a bootstrap sample of 100 observations.
(c) Show the first 10 rows of the sample and identify if any rows are repeated.

```python
# (a)
import pandas as pd
from sklearn.utils import resample

df = pd.read_csv("btissue.csv")
X = df.drop(columns=["class"])

# (b)
boot = resample(X, n_samples=100, replace=True, random_state=42)

# (c)
print(boot.head(10))
print(boot.duplicated().any())
```

Q3. Instead of relying on a single train–test split, you want to check how consistent your model is.
Tasks:

(a) Using the btissue.csv dataset, implement 5-fold cross-validation.
(b) For each fold, print the train/test indices and record how many samples are used for training vs testing.
(c) Visualize or summarize how different folds cover the entire dataset without overlap.

```python
import pandas as pd
from sklearn.model_selection import KFold
import numpy as np

df = pd.read_csv("btissue.csv")

kf = KFold(n_splits=5, shuffle=True, random_state=42)

all_test = []

for i, (train_idx, test_idx) in enumerate(kf.split(df)):
    print("Fold", i+1)
    print("Train indices:", train_idx)
    print("Test indices:", test_idx)
    print("Train size:", len(train_idx), "Test size:", len(test_idx))
    print()
    all_test.append(test_idx)

all_test = np.concatenate(all_test)
print("Total samples:", len(df))
print("Total test indices collected across folds:", len(all_test))
print("Unique test indices:", len(np.unique(all_test)))
```

Q4. Testing two validation techniques to measure model generalization.
Tasks:

(a) Use the btissue.csv dataset and a Decision Tree Classifier.
(b) Evaluate model performance using:
    i) Holdout (80/20 split)
    ii) 5-Fold Cross-Validation
(c) Compare the accuracy results from both methods.

```python
# (a)
import pandas as pd
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

df = pd.read_csv("btissue.csv")
X = df.drop("class", axis=1)
y = df["class"]

# (b-i) Holdout
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
holdout_acc = accuracy_score(y_test, dt.predict(X_test))

# (b-ii) 5-Fold CV
kf = KFold(n_splits=5, shuffle=True, random_state=42)
cv_acc = cross_val_score(dt, X, y, cv=kf).mean()

# (c)
print("Holdout Accuracy:", holdout_acc)
print("5-Fold CV Accuracy:", cv_acc)
```

Q5. Feature Creation from Structured Data

(a) Using a dataset containing columns like Age, Income, and Spending Score, construct new derived features such as Age Group, Income-to-Spending Ratio, and Normalized Spending.
(b) Plot and analyze how the new features correlate with the target variable.

```python
# (a)
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("data.csv")

df["Age_Group"] = pd.cut(df["Age"], bins=[0,25,45,65,120],
                         labels=["Young","Adult","Middle","Senior"])

df["Income_Spend_Ratio"] = df["Annual Income (k$)"] / df["Spending Score (1-100)"]

df["Norm_Spending"] = df["Spending Score (1-100)"] / df["Spending Score (1-100)"].max()

# (b)
print(df[["Income_Spend_Ratio","Norm_Spending","Spending Score (1-100)"]].corr())

plt.scatter(df["Norm_Spending"], df["Spending Score (1-100)"])
plt.show()
```

Q6. Load the Iris dataset and select a subset of features manually using the .iloc function. Train a simple Decision Tree Classifier using only the selected subset of features and compare its performance with the model trained using all features.

Tasks:

(a) Load the Iris dataset from sklearn.datasets.
(b) Create a DataFrame and display the first few rows.
(c) Train a Decision Tree Classifier using all features and record the accuracy.
(d) Select a subset of columns (for example, the first two features: sepal length and sepal width) using .iloc.
(e) Train another model using only the selected features and evaluate its accuracy.
(f) Compare and discuss the results of both models.

```python
# (a)
from sklearn.datasets import load_iris
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris = load_iris()

# (b)
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df["target"] = iris.target
print(df.head())

# (c)
X = df.iloc[:, :-1]
y = df["target"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
dt_all = DecisionTreeClassifier()
dt_all.fit(X_train, y_train)
acc_all = accuracy_score(y_test, dt_all.predict(X_test))

# (d)
X_sub = df.iloc[:, :2]   # first two features

# (e)
X_train2, X_test2, y_train2, y_test2 = train_test_split(X_sub, y, test_size=0.2, random_state=42)
dt_sub = DecisionTreeClassifier()
dt_sub.fit(X_train2, y_train2)
acc_sub = accuracy_score(y_test2, dt_sub.predict(X_test2))

# (f)
print("Accuracy with all features:", acc_all)
print("Accuracy with subset features:", acc_sub)
```

Q7. Load the Iris dataset and apply Principal Component Analysis (PCA) to reduce its four numerical features (sepal length, sepal width, petal length, petal width) into two principal components. Visualize the transformed data in a 2D scatter plot to observe how the classes (Setosa, Versicolor, Virginica) are separated in the reduced feature space. Additionally, display the explained variance ratio for each component.

Tasks:

(a) Load the Iris dataset using sklearn.datasets.

(b) Perform PCA to reduce the dataset to two components.

(c) Create a new DataFrame containing the two principal components and target labels.

(d) Plot the two components using a scatter plot with different colors for each class.

```python
# (a)
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
import pandas as pd
import matplotlib.pyplot as plt

iris = load_iris()
X = iris.data
y = iris.target

# (b)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# (c)
df = pd.DataFrame(X_pca, columns=["PC1", "PC2"])
df["target"] = y

print(pca.explained_variance_ratio_)

# (d)
plt.scatter(df["PC1"], df["PC2"], c=df["target"])
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

Q8. Create a dataset containing employee information, including Department, Job Role, and Marital Status. Convert all categorical columns into numeric form so that the dataset can be used effectively for training machine learning models. Use appropriate encoding techniques such as Label Encoding and One-Hot Encoding.

Tasks:

(a) Create a DataFrame with the following columns and sample data:
Department (e.g., HR, IT, Finance)
Job_Role (e.g., Manager, Analyst, Clerk)
Marital_Status (e.g., Single, Married, Divorced)

(b) Display the original dataset.

(c) Encode categorical columns using:
Label Encoding for ordered or binary categories.
One-Hot Encoding for nominal categories.

```python
# (a)
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.DataFrame({
    "Department": ["HR", "IT", "Finance", "IT"],
    "Job_Role": ["Manager", "Analyst", "Clerk", "Manager"],
    "Marital_Status": ["Single", "Married", "Divorced", "Single"]
})

# (b)
print(df)

# (c) Label Encoding (example: Marital_Status)
le = LabelEncoder()
df["Marital_Status_LE"] = le.fit_transform(df["Marital_Status"])

# One-Hot Encoding (for Department, Job_Role)
df_encoded = pd.get_dummies(df, columns=["Department", "Job_Role"])

print(df_encoded)
```