Department of Information Engineering Technology

The Superior University, Lahore

Artificial Intelligence Lab

Experiment No.5

Supervised Learning: Classification (Cross-Validation using multiple Classifiers)

Prepared for

_____

By:

Name: _____

ID: _____

Section: _____

Semester: _____

Total Marks: _____

Obtained Marks: _____

Signature: _____

Date: _____

Experiment No. 5

Supervised Learning: Classification (Cross-Validation using multiple Classifiers)

(Rubrics)

Name: _____ **Roll No. : _____**

A. PSYCHOMOTOR

Sr. No.	Criteria	Allocated Marks	Unacceptable 0%	Poor 25%	Fair 50%	Good 75%	Excellent 100%	Total Obtained
1	Follow Procedures	1	0	0.25	0.5	0.75	1	
2	Software and Simulations	2	0	0.5	1	1.5	2	
3	Accuracy in Output Results	3	0	0.75	1.5	2.25	3	
Sub Total		**6**	**Sub Total marks Obtained in Psychomotor(P)**					

B. AFFECTIVE

Sr. No.	Criteria	Allocated Marks	Unacceptable 0%	Poor 25%	Fair 50%	Good 75%	Excellent 100%	Total Obtained
1	Respond to Questions	1	0	0.25	0.5	0.75	1	
2	Lab Report	1	0	0.25	0.5	0.75	1	
3	Assigned Task	2	0	0.5	1	1.5	2	
Sub Total		**4**	**Sub Total marks Obtained in Affective (A)**					

Instructor Name:_____ **Total Marks (P+A): _____**

Instructor Signature:_____ **Obtained Marks (P+A): _____**

Introduction to Decision Tree

A decision tree is a popular machine learning algorithm used for both classification and regression tasks in artificial intelligence. It is a simple and intuitive model that is often used to make decisions based on input features.

Here's how a decision tree works:

- **Tree Structure**: A decision tree consists of nodes, where each node represents a decision or a test on a feature. The tree starts with a root node and branches out into internal nodes and leaf nodes. Internal nodes contain decision rules, while leaf nodes represent the final class or value predictions.
- **Decision Rules:** At each internal node, a decision rule is applied to one of the input features. This rule is typically a binary decision (e.g., Is feature X greater than a certain value?). The decision rule guides the tree's traversal.
- **Leaf Nodes:** The tree traversal continues from the root node through internal nodes until it reaches a leaf node. The leaf node contains the predicted class label (in classification) or the predicted value (in regression).
- **Training:** To build a decision tree, you need a dataset with labeled examples. The algorithm selects the best features and decision rules to create a tree that best fits the training data. Common algorithms for constructing decision trees include CART (Classification and Regression Trees) and ID3 (Iterative Dichotomiser 3).
- **Pruning:** Decision trees can be prone to overfitting, where they fit the training data too closely and do not generalize well to unseen data. Pruning is a technique used to reduce the complexity of the tree by removing branches that do not significantly improve predictive accuracy.
- **Predictions:** Once a decision tree is trained, you can use it to make predictions on new, unseen data. Starting at the root node, you follow the decision rules down the tree until you reach a leaf node, which provides the prediction.

Decision trees have several advantages, including their interpretability, ease of use, and ability to handle both categorical and numerical data. However, they may not always be the most accurate model for complex problems and can be sensitive to small variations in the training data.

Ensemble methods like Random Forest and Gradient Boosting are often used to improve the performance of decision trees by combining the predictions from multiple trees. These ensembles can reduce overfitting and enhance predictive accuracy.

Decision trees are widely used in various applications, including medical diagnosis, fraud detection, recommendation systems, and more, due to their versatility and ease of interpretation.

Decision Tree with Python

In Python, you can implement decision trees and work with them using various libraries, with scikit-learn being one of the most popular for machine learning tasks. Here's a basic example of how to create and use a decision tree classifier in Python using scikit-learn:

1. **Importing Libraries:** Start by importing the necessary libraries.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn import tree
from sklearn.linear_model import LinearRegression
```

2. **Load Data:** Load your dataset using pandas or any other method suitable for your data source.

```
df = pd.read_csv("salaries.csv")
df.head()
```

3. **Prepare Data:** Organize your data into the dependent variable (target) and independent variables (features).

```
inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']
```

4. **Label Encoder:** Label encoding is a technique used in machine learning and data preprocessing to convert categorical data (data that represents categories or labels) into numerical values. It's commonly used when working with algorithms or models that require numerical input data. The label encoder assigns a unique integer to each category in a categorical feature.

```
le_company = LabelEncoder()
le_job = LabelEncoder()
le_degree = LabelEncoder()
inputs['company_n'] = le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_job.fit_transform(inputs['job'])
inputs['degree_n'] = le_degree.fit_transform(inputs['degree'])
inputs_n = inputs.drop(['company','job','degree'],axis='columns')
```

5. **Create and Train the Model:** Create a linear regression model and fit it to your training data.

```
model = tree.DecisionTreeClassifier()
model.fit(inputs_n, target)
```

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

```
model.predict([[2,1,0]])
```

7. **Check Accuracy:** *.score(X_test, y_test)* is typically used to calculate the accuracy of a classification model, not for linear regression. In scikit-learn, the score method is used to compute the accuracy of classifiers, and it expects the features (X_test) and the true labels (y_test) as input.

```
model.score(inputs_n,target)
```

Introduction to Support Vector Machine (SVM)

Support Vector Machine (SVM) is a powerful and versatile machine learning algorithm used for both classification and regression tasks. SVM is particularly effective in cases where data is not linearly separable or when you want to find a good balance between bias and variance in your model. SVM is a supervised learning algorithm, meaning it requires labeled training data to make predictions. The main idea behind SVM is to find a hyperplane that best separates the data into different classes while maximizing the margin between the classes. The "support vectors" are the data points that are closest to the decision boundary (the hyperplane). These are the most important points in SVM, as they define the margin.

Here's a brief introduction to SVM:

- **Linear vs. Non-Linear Separation:** SVM is initially designed for linearly separable data, where a straight line (in 2D), a hyperplane (in higher dimensions) can perfectly separate two classes. In cases where data is not linearly separable, SVM can still be used with a "kernel trick" to map the data into a higher-dimensional space where separation becomes possible.
- **Margin:** The margin is the distance between the decision boundary and the support vectors. SVM aims to find the hyperplane that maximizes this margin. A larger margin indicates a more confident separation.
- **C and Regularization:** The parameter C in SVM allows you to control the trade-off between maximizing the margin and minimizing the classification error. A smaller C value will result in a larger margin but may allow some misclassifications. A larger C value will minimize misclassifications but may lead to a smaller margin.
- **Kernel Functions:** SVM can use different kernel functions, such as linear, radial basis function (RBF), polynomial, and more. These functions determine the shape of the decision boundary in the transformed space.

Advantages:

- SVM is effective in high-dimensional spaces.
- It's memory-efficient because it uses a subset of data points (support vectors).
- It can handle non-linear decision boundaries through kernel functions.
- It is robust against overfitting, thanks to the margin concept.

Disadvantages:

- SVMs can be sensitive to the choice of the kernel and its parameters.
- Training an SVM can be time-consuming, especially on large datasets.
- Interpretability can be a challenge, especially in high-dimensional spaces.

SVM is a versatile and powerful algorithm, but it may require some tuning to achieve the best results. The choice of kernel, regularization parameter C, and other hyperparameters depends on the specific problem you are trying to solve.

SVM with Python

Support Vector Machine (SVM) is a popular supervised machine learning algorithm used for both classification and regression tasks. In scikit-learn, which is a widely used machine learning library in Python, the SVM algorithm is implemented through the SVC (Support Vector Classification) class for classification tasks and SVR (Support Vector Regression) class for regression tasks.

We'll be using SVC class for classification tasks.

1. **Importing Libraries:** Start by importing the necessary libraries.

```
import pandas as pd
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
```

2. **Load Data:** Load your dataset using pandas or any other method suitable for your data source.

```
iris = load_iris()
df = pd.DataFrame(iris.data,columns=iris.feature_names)
df['target'] = iris.target
```

3. **Prepare Data:** Organize your data into the dependent variable (target) and independent variables (features).

```
X = df.drop(['target','flower_name'], axis='columns')
y = df.target
```

4. **Scatter Plot:** You can visualize by plotting the actual vs. predicted values or any other relevant visualizations.

```
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(df0['sepal length (cm)'], df0['sepal width
(cm)'],color="green",marker='+')
plt.scatter(df1['sepal length (cm)'], df1['sepal width
(cm)'],color="blue",marker='.')
```

5. **Split the dataset into Training and Testing sets:** Divide your dataset into two parts: a training set used to train the model and a testing set used to evaluate its performance. The *train_test_split* function helps you do this.

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

6. **Create and Train the Model:** Create a logistic regression model and fit it to your training data.

```
model = SVC()
model.fit(X_train, y_train)
```

7. **Check Accuracy:** *.score(X_test, y_test*) is typically used to calculate the accuracy of a classification model. In scikit-learn, the score method is used to compute the accuracy of classifiers, and it expects the features (X_test) and the true labels (y_test) as input.

```
model.score(X_test, y_test)
```

Regularization (C):

Regularization in SVM is controlled by the hyperparameter C, often referred to as the "cost parameter."

C determines the trade-off between maximizing the margin and minimizing the classification error.

A smaller C value results in a larger margin but allows some misclassifications in the training data (softer margin). This can make the model more robust and less prone to overfitting.

A larger C value reduces the margin, attempting to classify all training points correctly (hard margin). This may lead to overfitting, where the model fits the training data very closely but may not generalize well to unseen data.

The choice of C depends on the problem and the balance you want to strike between maximizing the margin and minimizing classification errors.

```
model_C = SVC(C=1)
model_C.fit(X_train, y_train)
model_C.score(X_test, y_test)
model_C = SVC(C=10)
model_C.fit(X_train, y_train)
model_C.score(X_test, y_test)
```

Gamma (γ):

Gamma is a hyperparameter used in the context of the radial basis function (RBF) kernel, which is one of the most commonly used kernel functions in SVM.

The gamma parameter controls the shape and flexibility of the decision boundary. A small gamma results in a more flexible (and potentially overfit) boundary, while a large gamma makes the boundary smoother and less flexible.

When gamma is small, each data point has a far-reaching influence on the decision boundary, which can lead to overfitting. When gamma is large, the influence is limited to nearby data points, leading to a smoother and less complex boundary.

The choice of gamma depends on the specific dataset and the level of complexity you need in your decision boundary.

```
model_g = SVC(gamma=10)
model_g.fit(X_train, y_train)
model_g.score(X_test, y_test)
```

Kernel:

The kernel in SVM specifies the type of function used to map data into a higher-dimensional space, where it might become linearly separable.

The choice of kernel depends on the nature of the data and the problem you are trying to solve. Common kernel functions include:

- **Linear Kernel:** It is used for linearly separable data and results in a linear decision boundary.
- **RBF (Radial Basis Function) Kernel:** This is a popular choice for handling non-linear data. It allows the SVM to create complex, non-linear decision boundaries.
- **Polynomial Kernel:** It introduces non-linearity through polynomial functions. You can specify the degree of the polynomial as a hyperparameter.
- **Sigmoid Kernel:** It maps data into a hyperbolic tangent space, suitable for data with a sigmoid shape.

```
model_linear_kernal = SVC(kernel='linear')
model_linear_kernal.fit(X_train, y_train)
model_linear_kernal.score(X_test, y_test)
```

Introduction to Random Forest

Random Forest is a popular ensemble learning method in machine learning, widely used for both classification and regression tasks. It's based on the idea of combining multiple decision trees to create a more robust and accurate model. Random Forest was introduced by Leo Breiman and Adele Cutler and is known for its ability to handle high-dimensional data and provide robust predictions. Here's an overview of Random Forest:

- **Ensemble Learning:** Ensemble learning involves combining the predictions of multiple machine learning models to improve overall predictive accuracy and reduce the risk of overfitting. Random Forest is an ensemble method that uses a collection of decision trees to make predictions.
- **Decision Trees:** Decision trees are the basic building blocks of Random Forest. Each tree is constructed using a subset of the training data and a random subset of features. Decision trees are known for their simplicity and interpretability. However, they can be prone to overfitting.
- **Randomness:** The "random" in Random Forest comes from two sources of randomness:
    - **Bootstrap Sampling:** Each tree is built using a random sample (with replacement) from the original dataset. This process is called bootstrapping.
    - **Feature Randomness:** At each node of a tree, a random subset of features is considered for splitting the node.
- **Voting or Averaging:** For classification tasks, Random Forest combines the predictions of individual trees using majority voting.
- For regression tasks, the predictions are averaged to obtain the final prediction.

Advantages of Random Forest:

- **High Accuracy:** Random Forest often yields high predictive accuracy and generalizes well to unseen data.
- **Robustness:** It is robust against overfitting due to the combination of multiple decision trees.
- **Feature Importance:** Random Forest can provide insights into feature importance, helping to identify the most relevant features in the data.
- **Non-linearity:** It can handle non-linear relationships between features and the target variable.

Applications:

Random Forest is used in various applications, including:

- Image classification
- Text classification
- Anomaly detection
- Bioinformatics
- Remote sensing
- Financial modeling

Random Forest is a versatile and powerful machine learning technique that is particularly well-suited for complex and high-dimensional data. It is a popular choice in both academia and industry due to its robustness and ease of use.

Random Forest with Python

Implementing a Random Forest classifier or regressor in Python using the scikit-learn library. Scikit-learn is a powerful Python library that provides a wide range of machine learning algorithms, including Random Forests. Here's an example of how to use Random Forest in Python:

1. **Importing Libraries:** Start by importing the necessary libraries.

```
import pandas as pd
from sklearn.datasets import load_digits
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sn
```

2. **Load Data:** Load your dataset using pandas or any other method suitable for your data source.

```
digits = load_digits()
```

3. **Prepare Data:** Organize your data into the dependent variable (target) and independent variables (features).

```
df = pd.DataFrame(digits.data)
df['target'] = digits.target
X = df.drop('target',axis='columns')
y = df.target
```

4. **Split the dataset into Training and Testing sets:** Divide your dataset into two parts: a training set used to train the model and a testing set used to evaluate its performance. The *train_test_split* function helps you do this.

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

5. **Create and Train the Model:** Create a logistic regression model and fit it to your training data.

```
model = RandomForestClassifier(n_estimators=20)
model.fit(X_train, y_train)
```

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

```
y_predicted = model.predict(X_test)
```

7. **Check Accuracy:** *.score(X_test, y_test)* is typically used to calculate the accuracy of a classification model. In scikit-learn, the score method is used to compute the accuracy of classifiers, and it expects the features (X_test) and the true labels (y_test) as input.

```
model.score(X_test, y_test)
```

Confusion Matrix:

A confusion matrix is a common tool for evaluating the performance of a classification model. It provides a detailed summary of how many true positives, true negatives, false positives, and false negatives your model produced. In Python, you can create a confusion matrix using libraries like scikit-learn. Here's how to do it:

```
cm = confusion_matrix(y_test, y_predicted)
cm
plt.figure(figsize=(10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

. Introduction to K-Fold Cross Validation

K-Fold Cross-Validation is a widely used technique in machine learning and model evaluation. It's used to assess how well a machine learning model will generalize to an independent dataset (i.e., how it performs on new, unseen data). The basic idea is to divide your dataset into K equally sized "folds" (or subsets), train the model on K-1 of these folds, and then test it on the remaining fold. This process is repeated K times, each time using a different fold as the validation set. The results from these K iterations are averaged to evaluate the model's performance.

K-Fold Cross-Validation involves the following steps:

- **Data Splitting:** The dataset is divided into K equally sized "folds" or subsets. For example, if K is 5, the dataset is divided into 5 subsets.
- **Model Training and Testing:** The learning algorithm is trained and evaluated K times, with each of the K subsets serving as the test set exactly once, while the remaining K-1 subsets are used for training.
- **Performance Evaluation:** After K iterations, you have K performance measures (e.g., accuracy, mean squared error) from each fold. The performance measures are typically averaged to provide an overall evaluation of the model's performance.

Key Advantages:

- **Robust Performance Estimation:** It provides a more robust estimate of a model's performance compared to a single train-test split because it averages the results over multiple random splits of the data.
- **Bias-Variance Trade-off:** It helps you understand how your model's performance varies with different subsets of the data, revealing potential issues with overfitting (high variance) or underfitting (high bias).
- **Utilizes Data Effectively**: It makes efficient use of the available data by ensuring that each data point is used for both training and testing.
- **Model Comparison:** K-Fold Cross-Validation enables fair model comparisons. You can evaluate multiple models using the same cross-validation procedure and compare their performance.
- **Hyperparameter Tuning:** It is useful for tuning hyperparameters. You can perform cross-validation for different hyperparameter settings and choose the best combination.

Common Variations:

- **Stratified K-Fold:** In this variation, each fold maintains the same class distribution as the original dataset, ensuring that each fold has a representative mix of target classes. It's particularly useful for imbalanced datasets.
- **Leave-One-Out Cross-Validation (LOOCV**): LOOCV is a special case of K-Fold Cross-Validation where K is set to the number of samples in the dataset. It provides a comprehensive assessment but can be computationally expensive for large datasets.

Use Cases:

- K-Fold Cross-Validation is widely used in various machine learning tasks, including:
- Model evaluation and comparison.
- Hyperparameter tuning.
- Assessing model generalization and robustness.
- Dealing with limited data by maximizing its utility.

K-Fold Cross Validation with Python

K-Fold Cross-Validation is a valuable technique for estimating a model's performance on unseen data and assessing its generalization capabilities. It helps you make more informed decisions about hyperparameter tuning and model selection. Here's how you can perform K-Fold Cross-Validation in Python using scikit-learn:

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
#Load Dataset
digits = load_digits()
#Train_Test_Split
X_train, X_test, y_train, y_test =
train_test_split(digits.data,digits.target,test_size=0.3)
#Logistic Regression
lr = LogisticRegression(solver='liblinear',multi_class='ovr')
lr.fit(X_train, y_train)
lr.score(X_test, y_test)
#SVM
svm = SVC(gamma='auto')
svm.fit(X_train, y_train)
svm.score(X_test, y_test)
#Random Forest
rf = RandomForestClassifier(n_estimators=40)
rf.fit(X_train, y_train)
rf.score(X_test, y_test)
#KFold cross validation Basic example
from sklearn.model_selection import KFold
kf = KFold(n_splits=3)
kf
for train_index, test_index in kf.split([1,2,3,4,5,6,7,8,9]):
    print(train_index, test_index)
#Use KFold for our digits example
def get_score(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    return model.score(X_test, y_test)
get_score(SVC() ,X_train, X_test, y_train, y_test )
from sklearn.model_selection import StratifiedKFold
folds = StratifiedKFold(n_splits=3)
```

```
scores_logistic = []
scores_svm = []
scores_rf = []

for train_index, test_index in folds.split(digits.data,digits.target):
    X_train, X_test, y_train, y_test = digits.data[train_index],
digits.data[test_index],\
                                       digits.target[train_index],
digits.target[test_index]
    scores_logistic.append(get_score(LogisticRegression(solver='liblinea
r',multi_class='ovr'), X_train, X_test, y_train, y_test))
    scores_svm.append(get_score(SVC(gamma='auto'), X_train, X_test,
y_train, y_test))
    scores_rf.append(get_score(RandomForestClassifier(n_estimators=40),
X_train, X_test, y_train, y_test))
scores_logistic
scores_svm
scores_rf
```

Cross Validation Score:

Cross-validation is a crucial technique in machine learning to assess a model's performance and its ability to generalize to new, unseen data. The *cross_val_score* function in scikit-learn simplifies the process of cross-validation by providing a convenient way to obtain cross-validated performance metrics for your machine learning models.

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

digits = load_digits()
#Train_Test_Split
X_train, X_test, y_train, y_test =
train_test_split(digits.data,digits.target,test_size=0.3)
# Logistic regression model performance using cross_val_score
cross_val_score(LogisticRegression(solver='liblinear',multi_class='ovr')
, digits.data, digits.target,cv=3)
# svm model performance using cross_val_score
cross_val_score(SVC(gamma='auto'), digits.data, digits.target,cv=3)
```

```
# Random forestperformance using cross_val_score
cross_val_score(RandomForestClassifier(n_estimators=40),digits.data,
digits.target,cv=3)
#Parameter tunning using k fold cross validation
scores1 =
cross_val_score(RandomForestClassifier(n_estimators=70),digits.data,
digits.target, cv=10)
np.average(scores1)

scores2 =
cross_val_score(RandomForestClassifier(n_estimators=20),digits.data,
digits.target, cv=10)
np.average(scores2)
```

Lab task:

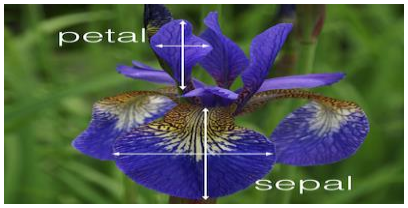
1. Exercise: Build decision tree model to predict survival based on certain parameters CSV file is provided to you. In this file using following columns build a model to predict if person would survive or not,

    - Pclass
    - Sex
    - Age
    - Fare
    - Calculate score of your model

2. Train SVM classifier using sklearn digits dataset (i.e. from sklearn.datasets import load_digits) and then,

    - Measure accuracy of your model using different kernels such as rbf and linear.
    - Tune your model further using regularization and gamma parameters and try to come up with highest accurancy score
    - Use 80% of samples as training data size

Home Task:

1. 

    - Use famous iris flower dataset from sklearn.datasets to predict flower species using random forest classifier.

- Measure prediction score using default n_estimators (10)
- Now fine tune your model by changing number of trees in your classifer and tell me what best score you can get using how many trees

2. Use iris flower dataset from sklearn library and use cross_val_score against following models to measure the performance of each. In the end figure out the model with best performance,

- Logistic Regression
- SVM
- Decision Tree
- Random Forest