1. Introduction to Classification

1.1 What is Classification?

1.1.1 Definition and Purpose

Classification is a supervised machine learning technique used to assign labels or categories to data points based on their features. The purpose of classification is to build a model that can predict the categorical label of new, unseen data points. Common examples of classification tasks include determining whether an email is spam or not, identifying the species of an iris flower, and diagnosing a medical condition based on patient data.

1.1.2 Difference Between Classification and Regression

The primary difference between classification and regression lies in the nature of the output variable:

- **Classification**: The output variable is categorical, meaning it belongs to a finite set of classes or labels. Examples include binary classification (e.g., spam or not spam) and multi-class classification (e.g., classifying different species of flowers).
- **Regression**: The output variable is continuous, meaning it can take any value within a range. Examples include predicting house prices and forecasting stock prices.

1.2 Use Cases of Classification

1.2.1 Applications in Various Domains

Classification has a wide range of applications across various domains. Some common use cases include:

- **Spam Detection**: Classifying emails as spam or not spam based on their content and metadata.
- **Medical Diagnosis**: Predicting the presence or type of a disease based on patient data such as symptoms, medical history, and test results.
- **Customer Segmentation**: Grouping customers into different segments based on their purchasing behaviour and demographic information for targeted marketing.
- **Sentiment Analysis**: Determining the sentiment (positive, negative, neutral) of a piece of text, such as a product review or social media post.
- **Image Recognition**: Classifying images into categories, such as identifying objects in photos or recognizing handwritten digits.
- **Credit Scoring**: Assessing the creditworthiness of individuals by classifying them into different risk categories based on their financial history.
- **Fraud Detection**: Identifying fraudulent transactions by classifying them based on patterns and anomalies in the data.
- **Speech Recognition**: Converting spoken language into text by classifying segments of audio into phonemes or words.

Classification is a fundamental task in machine learning and plays a crucial role in many practical applications, enabling automated decision-making and predictive analytics.

2. Logistic Regression

3. k-Nearest Neighbours (k-NN)

3.1 Understanding k-NN

3.1.1 Basic Concept and Intuition

The k-Nearest Neighbours (k-NN) algorithm is a simple, non-parametric, and instance-based learning method used for classification and regression. The basic concept is that given a new data point, the algorithm finds the k closest data points (neighbours) in the training dataset and makes predictions based on the majority class (for classification) or the average value (for regression) of these neighbours.

3.1.2 Choosing the Value of k

The value of k determines the number of neighbours to consider for making predictions. A small k value (e.g., 1) may lead to a noisy model that overfits the training data, while a large k value may result in a model that underfits and fails to capture the structure of the data. The optimal value of k is typically chosen using cross-validation.

3.1.3 Distance Metrics

Distance metrics are used to measure the similarity between data points. Common distance metrics include:

- **Euclidean Distance**: The straight-line distance between two points in Euclidean space.

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- **Manhattan Distance**: The sum of the absolute differences between the coordinates of two points.

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

4. Support Vector Machines (SVM)

4.1 Understanding SVM

4.1.1 Basic Concept and Intuition

Support Vector Machines (SVM) are a powerful supervised learning algorithm used for classification and regression tasks. The basic concept of SVM is to find the optimal hyperplane that separates the data points of different classes with the maximum margin. The points that lie closest to the decision boundary are called support vectors, and they play a crucial role in defining the position and orientation of the hyperplane.

4.1.2 Hyperplanes and Margins

- **Hyperplane**: In an n-dimensional space, a hyperplane is a flat affine subspace of dimension n-1 that separates the data points into different classes.
- **Margins**: The margin is the distance between the hyperplane and the nearest data points from either class. SVM aims to maximize this margin, providing better generalization to new data.

4.1.3 Kernel Trick

The kernel trick allows SVM to perform well in non-linear classification tasks by transforming the input space into a higher-dimensional space where a linear hyperplane can be used to separate the classes. Common kernels include:

- **Linear Kernel**: No transformation, directly used in linearly separable data.
- **Polynomial Kernel**: Applies a polynomial transformation to the input data.
- **Radial Basis Function (RBF) Kernel**: Maps the input space into a higher-dimensional space using Gaussian functions.

4.2 Mathematical Formulation

4.2.1 Objective Function

The objective function of SVM aims to find the hyperplane that maximizes the margin while minimizing classification errors

$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$
subject to the constraints:
$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i$
$\xi_i \geq 0$
where:

- $\mathbf{w}$ is the weight vector.
- b is the bias.
- C is the regularization parameter.
- ξ_i are the slack variables for handling misclassification.

4.2.2 Lagrange Multipliers

The dual form of the SVM optimization problem uses Lagrange multipliers to handle the constraints, resulting in the following optimization problem

$\max_\alpha \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$

subject to the constraints:

$0 \leq \alpha_i \leq C$

$\sum_{i=1}^{n} \alpha_i y_i = 0$

where α_i are the Lagrange multipliers.

5. Decision Trees

5.1 Understanding Decision Trees

5.1.1 Basic Concept and Intuition

Decision trees are a type of supervised learning algorithm used for classification and regression tasks. The algorithm works by recursively splitting the data into subsets based on the value of a feature, creating a tree-like model of decisions. Each internal node represents a decision based on a feature, each branch represents the outcome of the decision, and each leaf node represents a class label (for classification) or a continuous value (for regression).

5.1.2 Splitting Criteria

The quality of a split is determined by a splitting criterion, which measures how well a split separates the data into classes. Common splitting criteria include:

- **Gini Impurity**: Measures the impurity of a node. The goal is to minimize the Gini impurity.

$$Gini = 1 - \sum_{i=1}^{n} p_i^2$$

  where p is the probability of a randomly chosen element being classified correctly.

- **Entropy (Information Gain)**: Measures the amount of information needed to describe a node. The goal is to maximize information gain.

$$Entropy = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

5.1.3 Tree Pruning

Tree pruning is a technique used to prevent overfitting by removing branches that have little importance. Pruning can be done in two ways:

- **Pre-pruning (early stopping)**: Stop growing the tree when further splits do not significantly improve the model.
- **Post-pruning**: Grow the full tree and then remove branches that do not provide significant improvement based on a validation set.

6. Model Evaluation and Metrics

6.1 Evaluation Metrics for Classification

6.1.1 Accuracy

Accuracy is the ratio of correctly predicted instances to the total instances

Accuracy = Number of Correct Predictions / Total Number of Predictions

6.1.2 Precision, Recall, and F1-score

- **Precision**: Precision is the ratio of correctly predicted positive observations to the total predicted positives:

  Precision = True Positives / (True Positives + False Positives)

- **Recall (Sensitivity)**: Recall is the ratio of correctly predicted positive observations to all observations in the actual class:

  Recall = True Positives / (True Positives + False Negatives)

- **F1-score**: F1-score is the weighted average of Precision and Recall:

  F1-score = 2 × (Precision × Recall / Precision + Recall)

6.1.3 Confusion Matrix

A confusion matrix is a summary of prediction results on a classification problem. The matrix shows the ways in which your classification model is confused when it makes predictions. It gives insight not only into the errors being made by a classifier but also more importantly, the types of errors that are being made.

6.1.4 ROC and AUC

- **ROC Curve**: Receiver Operating Characteristic (ROC) curve is a graphical representation of a classifier's performance across all classification thresholds. It plots the True Positive Rate (Recall) against the False Positive Rate.
- **AUC (Area Under Curve)**: AUC measures the entire two-dimensional area underneath the entire ROC curve. An excellent model has an AUC close to 1, which means it has a good measure of separability. A poor model has an AUC close to 0, meaning it has the worst measure of separability.

6.2 Implementing Model Evaluation in PyTorch

6.2.1 Calculating Evaluation Metrics

Let's assume you have a trained model and you have obtained predictions for the test dataset. We will use these predictions to calculate various evaluation metrics.

```
import torch
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns

# Assume y_test and y_pred are the true labels and predicted labels respectively
# y_test_tensor and y_pred_tensor are torch tensors
y_test = y_test_tensor.numpy()
y_pred = y_pred_test

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Calculate precision, recall, and F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1-score: {f1:.2f}')

# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)
```

6.2.2 Visualizing Model Performance

Confusion Matrix

```
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

ROC Curve and AUC

For multiclass problems, the ROC curve and AUC can be calculated using the One-vs-Rest (OvR) approach.

```
from sklearn.preprocessing import label_binarize

# Binarize the output
y_test_binarized = label_binarize(y_test, classes=[0, 1, 2])
n_classes = y_test_binarized.shape[1]

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_pred_test[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve
plt.figure()
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Class {i} (area = {roc_auc[i]:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

Explanation with Examples

1. **Accuracy**: Measures how often the classifier is correct. For instance, if your model correctly predicts 90 out of 100 test samples, your accuracy is 90%.
2. **Precision, Recall, and F1-score**:
    - **Precision**: If your model predicts 10 samples as positive, out of which 8 are actually positive, precision is 0.8.
    - **Recall**: If there are 12 actual positive samples, and your model correctly predicts 8 of them, recall is 812=0.67\frac{8}{12} = 0.67128=0.67.
    - **F1-score**: The harmonic mean of precision and recall. If precision is 0.8 and recall is 0.67, F1-score is 2×0.8×0.670.8+0.67≈0.732 \times \frac{0.8 \times 0.67}{0.8 + 0.67} \approx 0.732×0.8+0.670.8×0.67≈0.73.

3. **Confusion Matrix**: Provides a summary of prediction results. For example, a binary classifier might have:

   [[50, 5],

   [10, 35]]

   This means 50 true negatives, 5 false positives, 10 false negatives, and 35 true positives.

4. **ROC and AUC**:
   - **ROC Curve**: Shows the trade-off between the true positive rate and false positive rate at various threshold settings. A curve closer to the top-left corner indicates a better model.
   - **AUC**: The area under the ROC curve. AUC of 1 indicates a perfect model, while 0.5 indicates a model with no discriminative power.