CS 484 - Introduction to Machine Learning
Final Exam

Problem 1:-

Given Data,

Data ID	X_1	X_2	Y	$P(false \mid X_1, X_2)$	$P(true \mid X_1, X_2)$
d_1	-4	-2	true	0.08	0.92
d_2	-2	-1	false	0.18	0.82
d_3	0	0	false	0.38	0.62
d_4	3	2	true	0.62	0.38
d_5	1	-1	false	0.82	0.18

Here, we know that

$$P(true \mid X_1, X_2) = 1 - P(false \mid X_1, X_2)$$

$$\frac{\partial CLL}{\partial w_i} = ((Y - P(true \mid X_1, X_2)) \cdot \frac{\partial z}{\partial w_i}$$

Here, $w = w_0 + w_1 X_1 + w_2 X_2$

$$\frac{\partial z}{\partial w_0} = 1 \quad \frac{\partial z}{\partial w_1} = X_1 \quad \frac{\partial z}{\partial w_2} = X_2$$

d_1:-

Given,

$X_1 = -4$, $X_2 = -2$, $P(true \mid X_1, X_2) = 0.92$

$Y - P(true \mid X_1, X_2)$, Here $Y = True$

So, $Y = 1$

$1 - 0.92 = 0.08$

$$\frac{\partial CLL}{\partial w_0} = (Y - P(\text{true} \mid X_1, X_2)) \cdot \frac{\partial z}{\partial w_0}$$

$$= 0.08 \times 1 = 0.08$$

$$\frac{\partial CLL}{\partial w_1} = (Y - P(\text{true} \mid X_1, X_2)) \frac{\partial z}{\partial w_1}$$

$$= 0.08 \times -4 = -0.32$$

$$\frac{\partial CLL}{\partial w_2} = (Y - P(\text{true} \mid X_1, X_2)) \cdot \frac{\partial z}{\partial w_2}$$

$$= 0.08 \times -2 = -0.16$$

So, Here we will be applying the above formulas for all data points.

d_2 :-

Given,

$X_1 = -2$, $X_2 = -1$, $Y = \text{False}$

so, $Y = 0$

$$P(\text{True} \mid X_1, X_2) = 0.82$$

$$Y - P(\text{true} \mid X_1, X_2) = 0 - 0.82 = -0.82$$

$$\frac{\partial CLL}{\partial w_0} = -0.82 \times 1 = -0.82$$

$$\frac{\partial CLL}{\partial w_1} = -0.82 \times (-2) = 1.64$$

$$\frac{\partial CLL}{\partial w_2} = -0.82 \times (-1) = 0.82$$

$\underline{d_3}$:-

Given,

$X_1 = 0$, $X_2 = 0$, $Y = $ False

so, $Y = 0$

$$P(\text{true} \mid X_1, X_2) = 0.62$$

$$Y - P(\text{True} \mid X_1, X_2) = 0 - 0.62 = -0.62$$

$$\frac{\partial CLL}{\partial w_0} = -0.62 \times 1 = -0.62$$

$$\frac{\partial CLL}{\partial w_1} = -0.62 \times 0 = 0$$

$$\frac{\partial CLL}{\partial w_2} = -0.62 \times 0 = 0$$

$\underline{d_4}$:-

Given,

$X_1 = 3$, $X_2 = 2$, $Y = $ True, so $Y = 1$

$$P(\text{True} \mid X_1, X_2) = 0.38$$

$$Y - P(\text{True} \mid X_1, X_2) = 1 - 0.38 = 0.62$$

$$\frac{\partial CLL}{\partial w_0} = 0.62 \times 1 = 0.62$$

$$\frac{\partial CLL}{\partial w_1} = 0.62 \times 3 = 1.86$$

$$\frac{\partial CLL}{\partial w_2} = 0.62 \times 2 = 1.24$$

d_5 :-

Given,

$X_1 = 1,\ X_2 = -1,\ Y = \text{False}$

so, $Y = 0$

$P(\text{True} \mid X_1, X_2) = 0.18$

$Y - P(\text{True} \mid X_1, X_2) = 0 - 0.18 = -0.18$

$$\frac{\partial CLL}{w_0} = -0.18 \times 1 = -0.18$$

$$\frac{\partial CLL}{\partial w_1} = -0.18 \times 1 = -0.18$$

$$\frac{\partial CLL}{\partial w_2} = -0.18 \times (-1) = 0.18$$

so, now let us fill the table given with the values we found

Data Id	X_1	X_2	Y	$P(\text{False} \mid X_1, X_2)$	$P(\text{True} \mid X_1, X_2)$	$\frac{\partial CLL}{\partial w_0}$	$\frac{\partial CLL}{\partial w_1}$	$\frac{\partial CLL}{\partial w_2}$
d_1	-4	-2	True	0.08	0.92	0.08	-0.32	-0.16
d_2	-2	-1	False	0.18	0.82	-0.82	1.64	0.82
d_3	0	0	False	0.38	0.62	-0.62	0	0
d_4	3	2	True	0.62	0.38	0.62	1.86	1.24
d_5	1	-1	False	0.82	0.18	-0.18	-0.18	0.18

Problem 2 :-

Given data,

$E = -(1-t)\ln(1-y) - t\ln(y)$

$A = -0.7$

$B = 0.6$

$C = 0.23$

$y = 0.8$

$f = 0$

$\omega_A = 2$

$\omega_B = -3$

$\omega_C = 4$

a) Partial gradient of E with respect to $\omega_C \left(\frac{\partial E}{\partial \omega_c}\right)$:

$$\frac{\partial E}{\partial \omega_c} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial \omega_c}$$

$$\frac{\partial E}{\partial y} = \frac{y-t}{y(1-y)} = \frac{0.8-0}{0.8(1-0.8)} = \frac{0.8}{(0.8)(0.2)} = 5$$

$$\frac{\partial y}{\partial \omega_c} = \sigma'(c) \cdot \frac{\partial c}{\partial \omega_c}$$

$$\sigma'(c) = y(1-y) = 0.8(1-0.8) = 0.8 \times 0.2$$
$$= 0.16$$

$$\frac{\partial y}{\partial \omega_c} = 5 \times 0.16 \times 0.23$$

$$\frac{\partial y}{\partial w_c} = 0.184$$

b) Partial gradient of E with respect to w_B $\left(\frac{\partial E}{\partial w_B}\right)$:

$$\frac{\partial E}{\partial w_B} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial c} \cdot \frac{\partial c}{\partial w_B}$$

Here, we know that $\frac{\partial E}{\partial y} = 5$

$$\frac{\partial y}{\partial c} = y(1-y) \cdot w_c = 0.64$$

$\sigma(A) = \tanh(A)$

$\sigma'(A) = 1 - \tanh(A)^2$

$z'(A) = A$ (since $z(A) = w_A * A$)

$(y-t) = 0.8 - 0 = 0.8$

$\sigma(B) = \text{sigmoid}(B)$

$\sigma'(B) = \text{sigmoid}(B)(1 - \text{sigmoid}(B))$

$A = -0.7$

$z'(B) = B$ (since $z(B) = w_B * B$)

$(y-t) = 0.8 - 0 = 0.8$

$\sigma'(B) = \text{sigmoid}(0.6)(1 - \text{sigmoid}(0.6))$
$\simeq 0.2350$

$y'(A) = 1 - \tanh(-0.7)^2 \simeq 0.5806$

$z'(B) = 0.6$

$\frac{\partial E}{\partial w_B} = 5 \times 0.64 \times 0.06216 = 0.1989$

c) partial gradient of error with respect to w_A ($\partial E / \partial w_A$)

$\frac{\partial E}{\partial w_A} = (y - t) \times \sigma'(A) \times z'(A)$

Here, $y = 0.8$.

$t = 0$

$A = -0.7$

$\sigma(A) = \tanh(A)$

$\sigma'(A) = 1 - \tanh(A)^2$

$z'(A) = A$ (since $z(A) = w_A * A$)

$(y - t) = 0.8 - 0 = 0.8$

$\sigma'(A) = 1 - \tanh(-0.7)^2 = 0.5806$

$z'(A) = -0.7$

$\frac{\partial B}{\partial w_A} = 0.6(1 - 0.6) \times (-0.7) = -0.168$

$\frac{\partial E}{\partial w_A} = 5 \times 0.64 \times (-0.31) \times (-0.160)$

$= 0.167$

Problem 3:-

Given,

$$P(x|\mu) = \prod_{i=1}^{D} \mu_i^{x_i} (1-\mu_i)^{(1-x_i)}$$

$$P(x|M,\pi) = \sum_{K=1}^{K} \pi_K \, P(x|\mu_K)$$

$$P(x|\mu_K) = \prod_{K=1}^{D} \mu_K^{x_{Ki}} (1-\mu_{Ki})^{(1-x_{Ki})}$$

i) we know that,

$$E(x) = \int x (P(x|M,\pi)) \, dx$$

$$E(x) = \int x \cdot \sum_{K=1}^{K} \pi_K \, P(x|\mu_K) dx$$

By linearity,

$$E(x) = \sum_{K=1}^{K} \pi_K \int x \cdot P(x|\mu_K) \, dx$$

Here,

For each component $P(x|\mu_K)$, $E(x|\mu_K)$ is μ_K

since each x_i is with parameters μ_{Ki}

So, $E\{x \mid \mu_k\} = \mu_k$

$$E[x] = \sum_{k=1}^{K} \pi_k \cdot \mu_k$$

Now, $P(x \mid \mu_k)$ is given and for

$x = (x_1, x_2, \ldots, x_D)$

$$E\{x \mid \mu_k\} = \mu_k$$

Hence, it is proved

2) Now $\Sigma_k = \mathrm{diag}(\mu_{ki}(1-\mu_{ki}))$

$$\mathrm{cov}\{x\} = \sum_{k=1}^{K} \pi_k (\Sigma_k + \mu_k \mu_k^T) - E\{x\} E\{x\}^T$$

$$E\{xx^T\} = \sum_{k=1}^{K} \pi_k \, E[xx^T \mid \mu_k]$$

For Bernoulli

$\mathrm{var}(x_i) = \mu_{ki}(1-\mu_{ki})$

$\Sigma_k = \mathrm{diag}(\mu_k(1-\mu_k))$

$$E[xx^T \mid \mu_k] = \Sigma_k + \mu_k \mu_k^T$$

$$E[xx^T] = \sum_{k=1}^{K} \pi_k (\Sigma_k + \mu_k \mu_k^T)$$

Now, $cov(x) = E[xx^T] - E[x]E[x]^T$

$$cov(x) = \sum_{k=1}^{K} \pi_k (\Sigma_k + \mu_k \mu_k^T) -$$

$$\left(\sum_{k=1}^{K} \pi_k \mu_k\right)\left(\sum_{k=1}^{K} \pi_k \mu_k\right)$$

It will become,

$$cov(x) = \sum_{k=1}^{K} \pi_k (\Sigma_k + \mu_k \mu_k^T) -$$

$$E[x]E[x]^T$$

Hence, it is proved.

Problem 4:

\+ Code + Text

```
import numpy as np
from sklearn.datasets import load_iris, load_breast_cancer, fetch_20newsgroups
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.exceptions import ConvergenceWarning
import warnings

warnings.filterwarnings("ignore", category=ConvergenceWarning)

def get_dataset(data_name):
    if data_name == "iris":
        dataset = load_iris()
        return dataset.data, dataset.target
    elif data_name == "breast_cancer":
        dataset = load_breast_cancer()
        return dataset.data, dataset.target
    elif data_name == "20newsgroups":
        topics = ['alt.atheism', 'sci.space', 'rec.sport.baseball', 'sci.med']
        data = fetch_20newsgroups(categories=topics)
        vectorizer = TfidfVectorizer(max_features=1000)
        X_transformed = vectorizer.fit_transform(data.data)
        return X_transformed, data.target

def manual_cv(data, labels, estimator, num_folds=10):
    indices = np.arange(len(labels))
    np.random.shuffle(indices)
    partition_size = len(labels) // num_folds
    fold_accuracies = []

    for fold_idx in range(num_folds):
        val_start = fold_idx * partition_size
        val_end = (fold_idx + 1) * partition_size if fold_idx < num_folds - 1 else len(labels)
        val_indices = indices[val_start:val_end]
        train_indices = np.concatenate([indices[:val_start], indices[val_end:]])
```

```
        train_data, val_data = data[train_indices], data[val_indices]
        train_labels, val_labels = labels[train_indices], labels[val_indices]

        estimator.fit(train_data, train_labels)
        fold_accuracies.append(estimator.score(val_data, val_labels))

    return np.mean(fold_accuracies)

def analyze_models(data_name):
    data, labels = get_dataset(data_name)
    train_data, test_data, train_labels, test_labels = train_test_split(data, labels, test_size=0.4, random_state=42)

    if isinstance(train_data, np.ndarray):
        scaler = StandardScaler()
        train_data = scaler.fit_transform(train_data)
        test_data = scaler.transform(test_data)

    hyperparameter_C = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
    feature_count = train_data.shape[1]
    layer_options = [int(feature_count * factor) for factor in [0.1, 0.2, 0.5, 1, 2, 5, 10] if int(feature_count * factor) > 0]

    print(f"\nPerforming Cross-Validation on: {data_name.upper()}")

    print("\nL2-LR Performance:")
    top_lr_accuracy = 0
    optimal_lr_param = None
    for reg_param in hyperparameter_C:
        lr_model = LogisticRegression(C=reg_param, max_iter=1000)
        accuracy = manual_cv(train_data, train_labels, lr_model)
        print(f"  hyperparameter_value={reg_param}: Mean Accuracy = {accuracy:.4f}")
        if accuracy > top_lr_accuracy:
            top_lr_accuracy = accuracy
            optimal_lr_param = reg_param

    print("\nSVM with Linear Kernel Performance:")
    best_linear_svm_score = 0
    best_svm_param = None
    for reg_param in hyperparameter_C:
        linear_svm = SVC(kernel="linear", C=reg_param)
        accuracy = manual_cv(train_data, train_labels, linear_svm)
```

```python
        print(f"  hyperparameter_value={reg_param}: Mean Accuracy = {accuracy:.4f}")
        if accuracy > best_linear_svm_score:
            best_linear_svm_score = accuracy
            best_svm_param = reg_param

    print("\nSVM with RBF Kernel Performance:")
    top_rbf_accuracy = 0
    best_rbf_param = None
    for reg_param in hyperparameter_C:
        rbf_svm = SVC(kernel="rbf", C=reg_param)
        accuracy = manual_cv(train_data, train_labels, rbf_svm)
        print(f"  hyperparameter_value={reg_param}: Mean Accuracy = {accuracy:.4f}")
        if accuracy > top_rbf_accuracy:
            top_rbf_accuracy = accuracy
            best_rbf_param = reg_param

    print("\nMLP Performance:")
    highest_mlp_accuracy = 0
    best_layer_config = None
    for layer_size in layer_options:
        mlp_model = MLPClassifier(hidden_layer_sizes=(layer_size,), max_iter=1000)
        accuracy = manual_cv(train_data, train_labels, mlp_model)
        print(f"  hidden_layer_sizes={layer_size}: Mean Accuracy = {accuracy:.4f}")
        if accuracy > highest_mlp_accuracy:
            highest_mlp_accuracy = accuracy
            best_layer_config = layer_size

    print("\nBest Model Summary:")
    print(f"  L2-LR: Best hyperparamter_value={optimal_lr_param}, Accuracy={top_lr_accuracy:.4f}")
    print(f"  SVM with Linear Kernel: Best hyperparamter_value={best_svm_param}, Accuracy={best_linear_svm_score:.4f}")
    print(f"  SVM with RBF Kernel: Best hyperparamter_value={best_rbf_param}, Accuracy={top_rbf_accuracy:.4f}")
    print(f"  MLP: Best hidden_layer_sizes={best_layer_config}, Accuracy={highest_mlp_accuracy:.4f}")

    return {
        "L2-LR": {"Parameter": optimal_lr_param, "Accuracy": top_lr_accuracy},
        "SVM with Linear Kernel": {"Parameter": best_svm_param, "Accuracy": best_linear_svm_score},
        "SVM with RBF Kernel": {"Parameter": best_rbf_param, "Accuracy": top_rbf_accuracy},
        "MLP": {"Layer Size": best_layer_config, "Accuracy": highest_mlp_accuracy},
    }
```


```python
    print(f"  SVM with Linear Kernel: Best hyperparamter_value={best_svm_param}, Accuracy={best_linear_svm_score:.4f}")
    print(f"  SVM with RBF Kernel: Best hyperparamter_value={best_rbf_param}, Accuracy={top_rbf_accuracy:.4f}")
    print(f"  MLP: Best hidden_layer_sizes={best_layer_config}, Accuracy={highest_mlp_accuracy:.4f}")

    return {
        "L2-LR": {"Parameter": optimal_lr_param, "Accuracy": top_lr_accuracy},
        "SVM with Linear Kernel": {"Parameter": best_svm_param, "Accuracy": best_linear_svm_score},
        "SVM with RBF Kernel": {"Parameter": best_rbf_param, "Accuracy": top_rbf_accuracy},
        "MLP": {"Layer Size": best_layer_config, "Accuracy": highest_mlp_accuracy},
    }
```


```python
if __name__ == "__main__":
    datasets_to_analyze = ["iris", "breast_cancer", "20newsgroups"]
    all_results = {}

    for dataset in datasets_to_analyze:
        all_results[dataset] = analyze_models(dataset)

    print("\nOverall Best and Worst Performing Models Across Datasets:")
    for dataset, results in all_results.items():
        best_model = max(results.items(), key=lambda model: model[1]["Accuracy"])
        worst_model = min(results.items(), key=lambda model: model[1]["Accuracy"])
        print(f"{dataset.upper()}:")
        print(f"  Best Model = {best_model[0]} with Accuracy = {best_model[1]['Accuracy']:.4f}")
        print(f"  Worst Model = {worst_model[0]} with Accuracy = {worst_model[1]['Accuracy']:.4f}")
```

```
Performing Cross-Validation on: IRIS

L2-LR Performance:
  hyperparameter_value=0.001: Mean Accuracy = 0.4111
  hyperparameter_value=0.01: Mean Accuracy = 0.8111
  hyperparameter_value=0.1: Mean Accuracy = 0.8889
  hyperparameter_value=1: Mean Accuracy = 0.9222
  hyperparameter_value=10: Mean Accuracy = 0.9444
  hyperparameter_value=100: Mean Accuracy = 0.9556
  hyperparameter_value=1000: Mean Accuracy = 0.9667

SVM with Linear Kernel Performance:
  hyperparameter_value=0.001: Mean Accuracy = 0.3111
  hyperparameter_value=0.01: Mean Accuracy = 0.6667
  hyperparameter_value=0.1: Mean Accuracy = 0.9444
  hyperparameter_value=1: Mean Accuracy = 0.9667
  hyperparameter_value=10: Mean Accuracy = 0.9667
  hyperparameter_value=100: Mean Accuracy = 0.9667
  hyperparameter_value=1000: Mean Accuracy = 0.9556

SVM with RBF Kernel Performance:
  hyperparameter_value=0.001: Mean Accuracy = 0.3333
  hyperparameter_value=0.01: Mean Accuracy = 0.2667
  hyperparameter_value=0.1: Mean Accuracy = 0.8667
  hyperparameter_value=1: Mean Accuracy = 0.9222
  hyperparameter_value=10: Mean Accuracy = 0.9333
  hyperparameter_value=100: Mean Accuracy = 0.9333
  hyperparameter_value=1000: Mean Accuracy = 0.9444

MLP Performance:
  hidden_layer_sizes=2: Mean Accuracy = 0.7333
  hidden_layer_sizes=4: Mean Accuracy = 0.9000
  hidden_layer_sizes=8: Mean Accuracy = 0.9333
  hidden_layer_sizes=20: Mean Accuracy = 0.9222
  hidden_layer_sizes=40: Mean Accuracy = 0.9333

Best Model Summary:
  L2-LR: Best hyperparamter_value=1000, Accuracy=0.9667
  SVM with Linear Kernel: Best hyperparamter_value=10, Accuracy=0.9667
  SVM with RBF Kernel: Best hyperparamter_value=1000, Accuracy=0.9444
  MLP: Best hidden_layer_sizes=8, Accuracy=0.9333
```

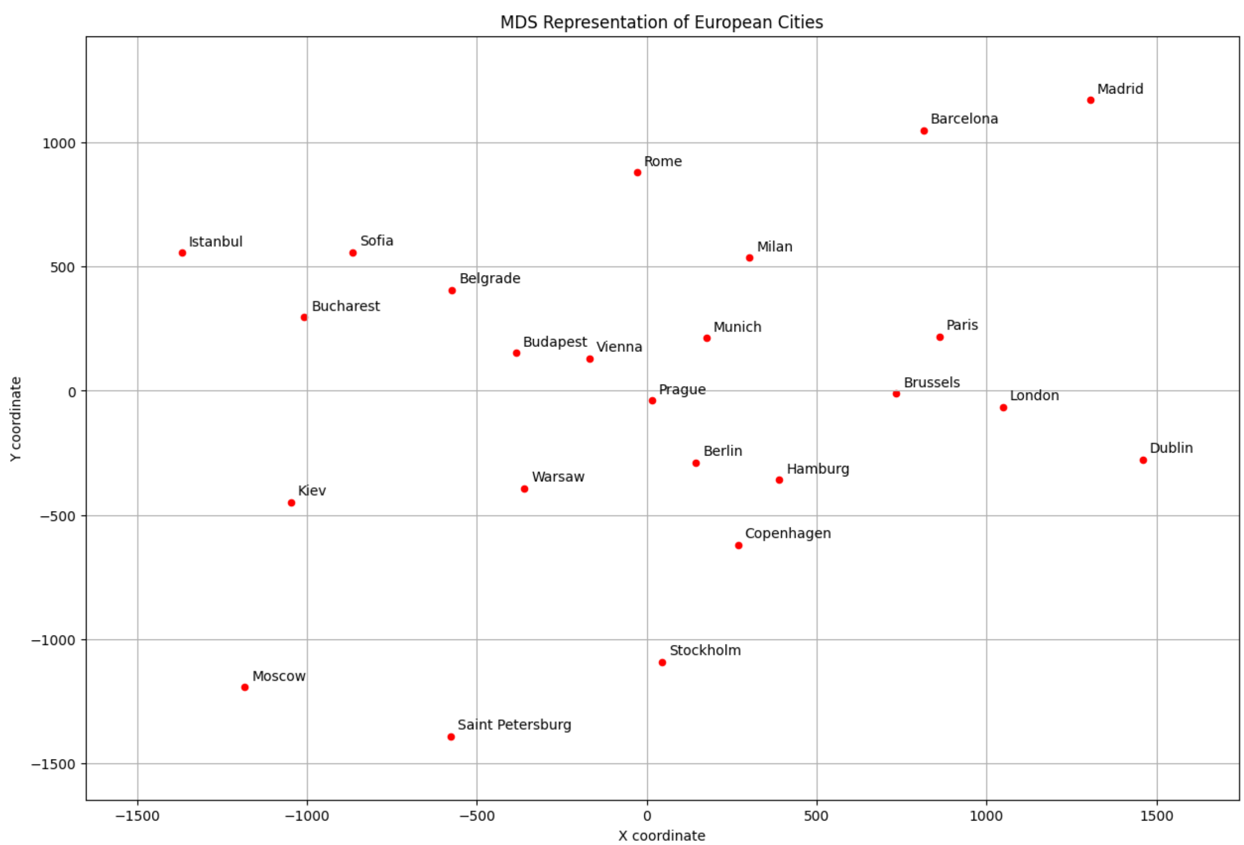
```
Performing Cross-Validation on: BREAST_CANCER

L2-LR Performance:
  hyperparameter_value=0.001: Mean Accuracy = 0.8858
  hyperparameter_value=0.01: Mean Accuracy = 0.9413
  hyperparameter_value=0.1: Mean Accuracy = 0.9708
  hyperparameter_value=1: Mean Accuracy = 0.9736
  hyperparameter_value=10: Mean Accuracy = 0.9647
  hyperparameter_value=100: Mean Accuracy = 0.9590
  hyperparameter_value=1000: Mean Accuracy = 0.9561

SVM with Linear Kernel Performance:
  hyperparameter_value=0.001: Mean Accuracy = 0.9266
  hyperparameter_value=0.01: Mean Accuracy = 0.9561
  hyperparameter_value=0.1: Mean Accuracy = 0.9647
  hyperparameter_value=1: Mean Accuracy = 0.9648
  hyperparameter_value=10: Mean Accuracy = 0.9708
  hyperparameter_value=100: Mean Accuracy = 0.9471
  hyperparameter_value=1000: Mean Accuracy = 0.9354

SVM with RBF Kernel Performance:
  hyperparameter_value=0.001: Mean Accuracy = 0.6128
  hyperparameter_value=0.01: Mean Accuracy = 0.6131
  hyperparameter_value=0.1: Mean Accuracy = 0.9384
  hyperparameter_value=1: Mean Accuracy = 0.9618
  hyperparameter_value=10: Mean Accuracy = 0.9679
  hyperparameter_value=100: Mean Accuracy = 0.9620
  hyperparameter_value=1000: Mean Accuracy = 0.9648

MLP Performance:
  hidden_layer_sizes=3: Mean Accuracy = 0.9588
  hidden_layer_sizes=6: Mean Accuracy = 0.9649
  hidden_layer_sizes=15: Mean Accuracy = 0.9677
  hidden_layer_sizes=30: Mean Accuracy = 0.9706
  hidden_layer_sizes=60: Mean Accuracy = 0.9708
  hidden_layer_sizes=150: Mean Accuracy = 0.9735
  hidden_layer_sizes=300: Mean Accuracy = 0.9706

Best Model Summary:
  L2-LR: Best hyperparamter_value=1, Accuracy=0.9736
  SVM with Linear Kernel: Best hyperparamter_value=10, Accuracy=0.9708
  SVM with RBF Kernel: Best hyperparamter_value=10, Accuracy=0.9679
  MLP: Best hidden_layer_sizes=150, Accuracy=0.9735
```

```
Performing Cross-Validation on: 20NEWSGROUPS

L2-LR Performance:
  hyperparameter_value=0.001: Mean Accuracy = 0.3971
  hyperparameter_value=0.01: Mean Accuracy = 0.7021
  hyperparameter_value=0.1: Mean Accuracy = 0.8882
  hyperparameter_value=1: Mean Accuracy = 0.9424
  hyperparameter_value=10: Mean Accuracy = 0.9469
  hyperparameter_value=100: Mean Accuracy = 0.9536
  hyperparameter_value=1000: Mean Accuracy = 0.9507

SVM with Linear Kernel Performance:
  hyperparameter_value=0.001: Mean Accuracy = 0.2291
  hyperparameter_value=0.01: Mean Accuracy = 0.2248
  hyperparameter_value=0.1: Mean Accuracy = 0.8708
  hyperparameter_value=1: Mean Accuracy = 0.9404
  hyperparameter_value=10: Mean Accuracy = 0.9449
  hyperparameter_value=100: Mean Accuracy = 0.9483
  hyperparameter_value=1000: Mean Accuracy = 0.9462

SVM with RBF Kernel Performance:
  hyperparameter_value=0.001: Mean Accuracy = 0.2320
  hyperparameter_value=0.01: Mean Accuracy = 0.2830
  hyperparameter_value=0.1: Mean Accuracy = 0.7047
  hyperparameter_value=1: Mean Accuracy = 0.9396
  hyperparameter_value=10: Mean Accuracy = 0.9432
  hyperparameter_value=100: Mean Accuracy = 0.9433
  hyperparameter_value=1000: Mean Accuracy = 0.9498

MLP Performance:
  hidden_layer_sizes=100: Mean Accuracy = 0.9500
  hidden_layer_sizes=200: Mean Accuracy = 0.9549
  hidden_layer_sizes=500: Mean Accuracy = 0.9544
  hidden_layer_sizes=1000: Mean Accuracy = 0.9572
  hidden_layer_sizes=2000: Mean Accuracy = 0.9616
  hidden_layer_sizes=5000: Mean Accuracy = 0.9573
  hidden_layer_sizes=10000: Mean Accuracy = 0.9521

Best Model Summary:
  L2-LR: Best hyperparamter_value=100, Accuracy=0.9536
  SVM with Linear Kernel: Best hyperparamter_value=100, Accuracy=0.9483
  SVM with RBF Kernel: Best hyperparamter_value=1000, Accuracy=0.9498
  MLP: Best hidden_layer_sizes=2000, Accuracy=0.9616
```

```
Best Model Summary:
  L2-LR: Best hyperparamter_value=100, Accuracy=0.9536
  SVM with Linear Kernel: Best hyperparamter_value=100, Accuracy=0.9483
  SVM with RBF Kernel: Best hyperparamter_value=1000, Accuracy=0.9498
  MLP: Best hidden_layer_sizes=2000, Accuracy=0.9616

Overall Best and Worst Performing Models Across Datasets:
IRIS:
  Best Model = SVM with Linear Kernel with Accuracy = 0.9667
  Worst Model = MLP with Accuracy = 0.9333
BREAST_CANCER:
  Best Model = L2-LR with Accuracy = 0.9736
  Worst Model = SVM with RBF Kernel with Accuracy = 0.9679
20NEWSGROUPS:
  Best Model = MLP with Accuracy = 0.9616
  Worst Model = SVM with Linear Kernel with Accuracy = 0.9483
```

Problem 5:

```
[3] import pandas as pd
    import numpy as np
    from sklearn.manifold import MDS
    import matplotlib.pyplot as plt

    cities_df = pd.read_csv('/content/cities.csv', sep=';', header=None)
    city_names = cities_df[0].tolist()
    distances = cities_df.iloc[:, 1:25].values.astype(float)

    mds = MDS(n_components=2, dissimilarity='precomputed', random_state=42)
    positions = mds.fit_transform(distances)

    plt.figure(figsize=(15, 10))
    plt.scatter(positions[:, 0], positions[:, 1], color='red', s=20)

    for i, city in enumerate(city_names):
        plt.annotate(city, (positions[i, 0], positions[i, 1]),
                     xytext=(5, 5), textcoords='offset points')

    plt.title('MDS Representation of European Cities')
    plt.xlabel('X coordinate')
    plt.ylabel('Y coordinate')
    plt.grid(True)

    plt.margins(0.1)
    plt.show()
```



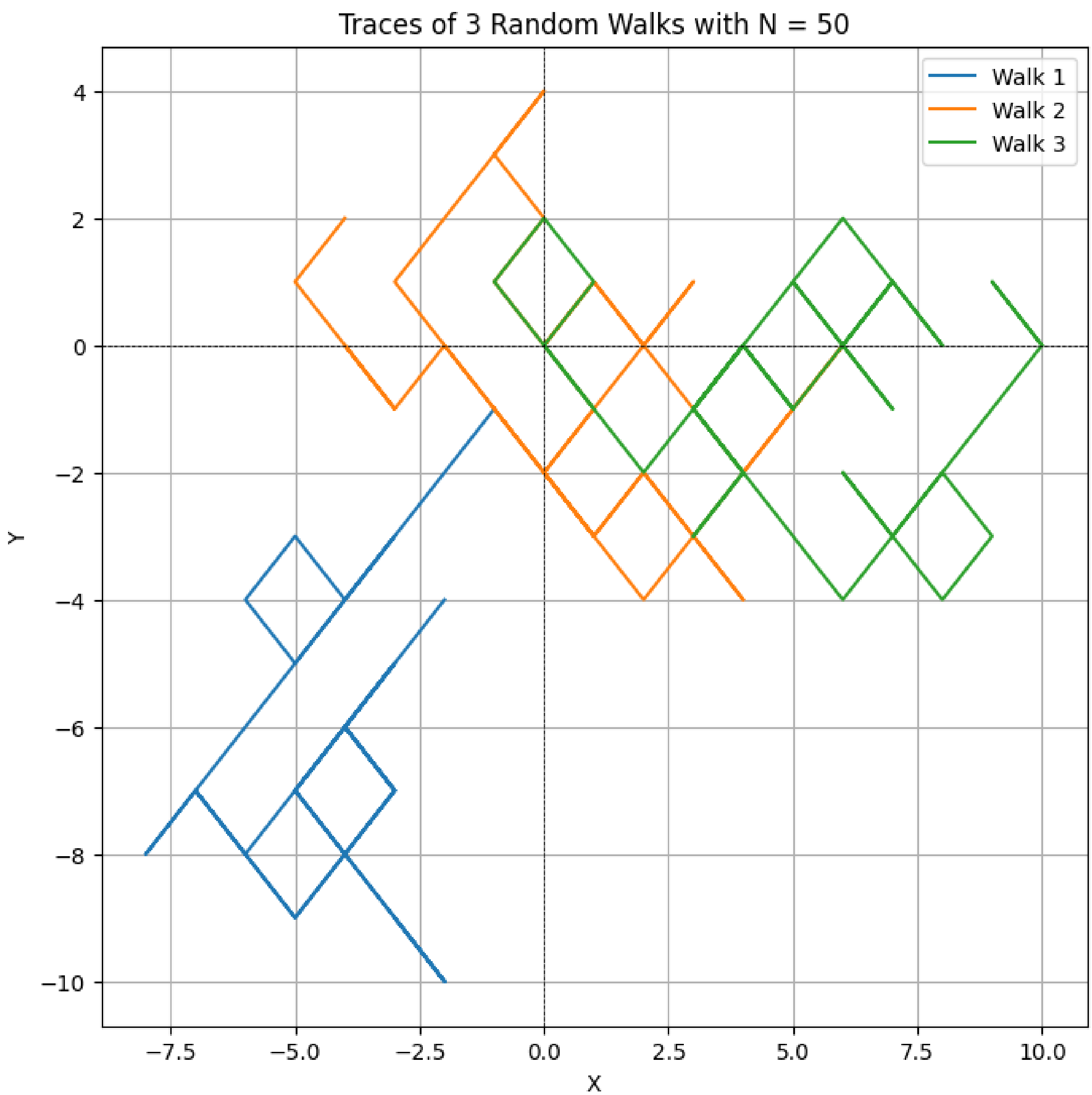
Problem 6:

[4]

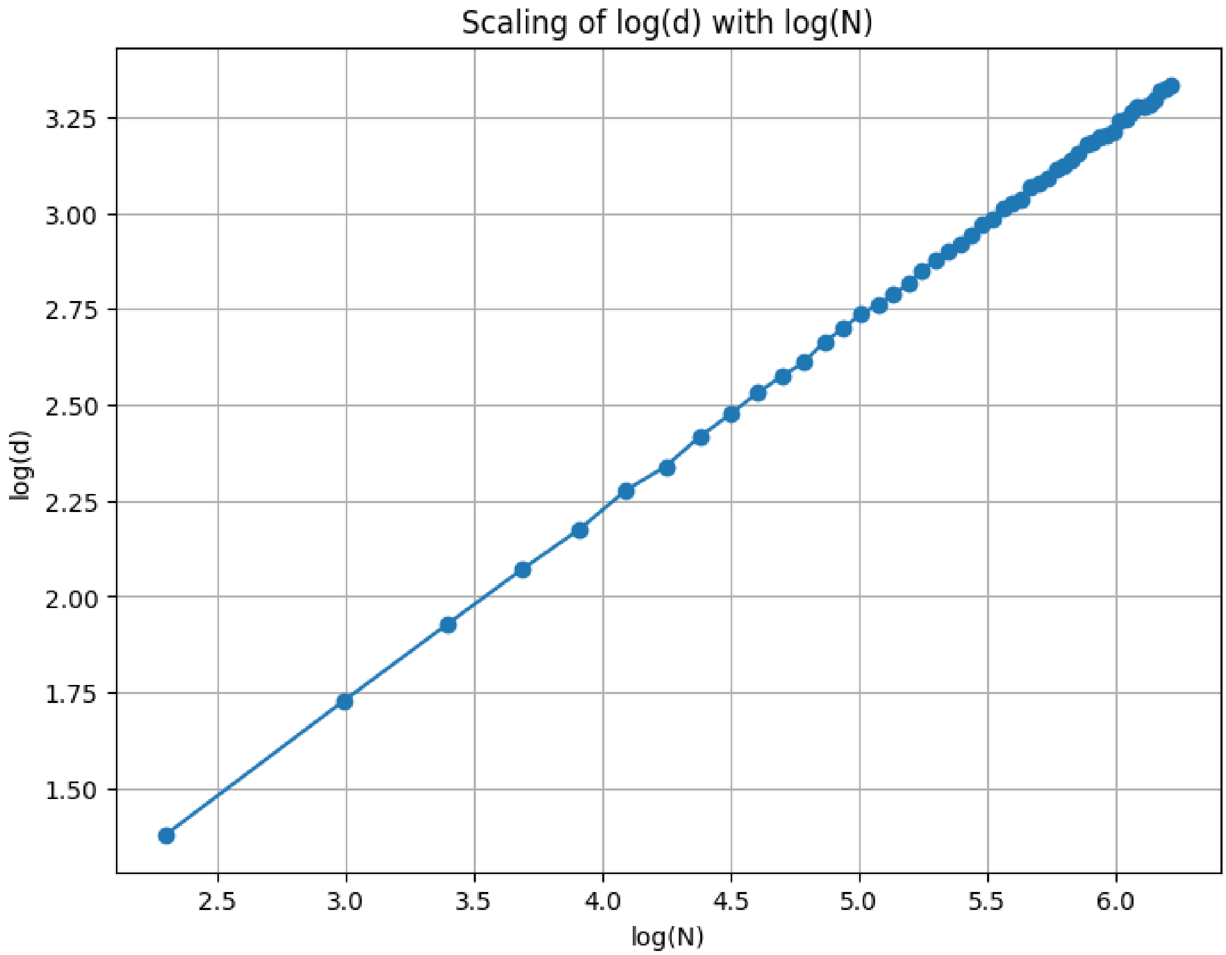
```
import numpy as np
import matplotlib.pyplot as plt

def random_walk_vectorized(n_steps, n_simulations=1):
    steps = np.random.choice([-1, 1], size=(n_simulations, n_steps, 2))
    positions = np.cumsum(steps, axis=1)
    return positions

N = 50
plt.figure(figsize=(8, 8))
for i in range(3):
    positions = random_walk_vectorized(N)[0]
    x, y = positions[:, 0], positions[:, 1]
    plt.plot(x, y, label=f"Walk {i+1}")
plt.title(f"Traces of 3 Random Walks with N = {N}")
plt.xlabel("X")
plt.ylabel("Y")
plt.axhline(0, color='black', linewidth=0.5, linestyle='--')
plt.axvline(0, color='black', linewidth=0.5, linestyle='--')
plt.legend()
plt.grid()
plt.show()

def calculate_d_vectorized(n_steps, n_simulations):
    positions = random_walk_vectorized(n_steps, n_simulations)
    final_positions = positions[:, -1, :]
    distances = np.sqrt(np.sum(final_positions**2, axis=1))
    return np.mean(distances)

n_simulations = 10000
average_d = calculate_d_vectorized(N, n_simulations)
print(f"Average distance for N = {N} over {n_simulations} simulations: {average_d:.2f}")

n_values = np.arange(10, 501, 10)
average_distances = [calculate_d_vectorized(n, 10000) for n in n_values]

plt.figure(figsize=(8, 6))
plt.plot(np.log(n_values), np.log(average_distances), marker='o', linestyle='-')
plt.title("Scaling of log(d) with log(N)")
```

[4]

```
plt.figure(figsize=(8, 6))
plt.plot(np.log(n_values), np.log(average_distances), marker='o', linestyle='-')
plt.title("Scaling of log(d) with log(N)")
plt.xlabel("log(N)")
plt.ylabel("log(d)")
plt.grid()

coefficients = np.polyfit(np.log(n_values), np.log(average_distances), 1)
slope = coefficients[0]
print(f"Slope of log-log plot: {slope:.2f}")
plt.show()
```




```
Average distance for N = 50 over 10000 simulations: 8.89
Slope of log-log plot: 0.50
```



Scaling of log(d) with log(N)
log(d)
log(N)
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0