AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

Faculty of Science and Technology

Assignment Cover Sheet

Assignment Title:	IMAGE CLASSIFICATION USING CIFAR-10 DATASET: APPLYING K-NEAREST NEIGHBORS CLASSIFIER USING EUCLIDEAN (L2) DISTANCE WITH 5-FOLD CROSS-VALIDATION		
Assignment No:	1	Date of Submission:	10 April 2024
Course Title:	MACHINE LEARNING		
Course Code:	CSC4232	Section:	B
Semester:	Spring 2024-25	Course Teacher:	TAIMAN ARHAM SIDDIQUE

Group Name/No.:

No	Name	ID	Program	Signature
1	Borshon Alfred Gomes	21-44561-1	BSc [CSE]	
2			Choose an item.	
3			Choose an item.	
4			Choose an item.	
5			Choose an item.	
6			Choose an item.	
7			Choose an item.	
8			Choose an item.	
9			Choose an item.	
10			Choose an item.	

Faculty use only		
FACULTYCOMMENTS	**Marks Obtained**	
	Total Marks	

IMAGE CLASSIFICATION USING CIFAR-10 DATASET: APPLYING K-NEAREST NEIGHBORS CLASSIFIER USING EUCLIDEAN (L2) DISTANCE WITH 5-FOLD CROSS-VALIDATION

Introduction

This report delves into the application and assessment of the k-Nearest Neighbors (k-NN) algorithm, a cornerstone of supervised machine learning, tailored for the task of image classification on the CIFAR-10 dataset. The CIFAR-10 dataset, a benchmark in the machine learning community, comprises 60,000 32x32 pixel color images evenly distributed across 10 diverse classes, including vehicles and animals. This rich dataset, partitioned into 50,000 training images and 10,000 testing images, serves as an ideal playground for exploring image classification techniques and their nuances.

The k-Nearest Neighbors Classifier

At its core, the k-Nearest Neighbors (k-NN) classifier embodies simplicity. It classifies unseen instances based on their similarity to examples within the training dataset. The "k" in k-NN refers to the number of nearest neighbors in the training set that the algorithm considers when making a classification decision. The fundamental premise is that similar instances tend to be near each other in the feature space, and thus, by identifying the "k" closest neighbors, the algorithm can adopt a majority vote among these neighbors' classes to predict the class of the query.

⭘ Importing necessary libraries

In this project, several Python libraries each serving a specific role were utilized

```python
import numpy as np
import matplotlib.pyplot as plt
import data_utils
import download
```

numpy is used for efficient numerical computations, especially with arrays and matrices.

matplotlib.pyplot is plotting library that enables us to visualize data and results in a graphical format.

data_utils is a utility script that provides functions to load and preprocess the CIFAR-10 dataset.

download is a module that contains functions to download the CIFAR-10 dataset and extract it to a local directory.

⦿ Data Visualization

Before delving into the classification process, it is crucial to understand the nature of the data. The ***visualize_data*** function randomly selects and displays a handful of images from each class in the CIFAR-10 dataset. This visual inspection is not only instrumental for gaining intuition about the dataset but also for verifying the data's integrity and preprocessing steps.

```python
def visualize_data(X_train, y_train):
    classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
    num_classes = len(classes)
    samples_per_class = 7
    for y, cls in enumerate(classes):
        idxs = np.flatnonzero(y_train == y)
        idxs = np.random.choice(idxs, samples_per_class, replace=False)
        for i, idx in enumerate(idxs):
            plt_idx = i * num_classes + y + 1
            plt.subplot(samples_per_class, num_classes, plt_idx)
            plt.imshow(X_train[idx].astype('uint8'))
            plt.axis('off')
            if i == 0:
                plt.title(cls)
    plt.show()
```

Visualization of Dataset

This image showcases a grid of sample images from the CIFAR-10 dataset, which is a collection commonly used for machine learning and computer vision training.

⭘ K-Nearest Neighbors Classifier Implementation

The ***k-Nearest Neighbors*** algorithm is the foundation of classification approach. Implementation in the ***KNearestNeighbor*** class consists of several key functions

```
 def train(self, X, y):
        self.X_train = X
        self.y_train = y
    def predict(self, X, k=1, distance='L2'):
        if distance == 'L2':
            dists = self.compute_distances_L2(X)
        elif distance == 'L1':
            dists = self.compute_distances_L1(X)
        else:
            raise ValueError('Invalid distance metric')
        return self.predict_labels(dists, k=k)
    def compute_distances_L2(self, X):
        num_test = X.shape[0]
        num_train = self.X_train.shape[0]
        dists = np.sqrt(np.sum(X**2, axis=1, keepdims=True) + np.sum(self.X_train**2, axis=1) - 2 *
X.dot(self.X_train.T))
        return dists
    def compute_distances_L1(self, X):
        num_test = X.shape[0]
        num_train = self.X_train.shape[0]
        dists = np.zeros((num_test, num_train))
        for i in range(num_test):
            dists[i, :] = np.sum(np.abs(self.X_train - X[i, :]), axis=1)
        return dists
    def predict_labels(self, dists, k=1):
        num_test = dists.shape[0]
        y_pred = np.zeros(num_test)
        for i in range(num_test):
            closest_y = self.y_train[np.argsort(dists[i, :])[:k]]
            y_pred[i] = np.argmax(np.bincount(closest_y))
        return y_pred
```

- ***train*** simply stores the training data and ***predict*** calculates the distances between test instances and all training instances to predict labels based on the nearest neighbors.
- ***predict_labels*** assigns a label to each test instance based on the majority vote of its nearest neighbors.
- ***compute_distances_L2*** and ***compute_distances_L1*** measure the similarity between instances using L2 (Euclidean) and L1 (Manhattan) distances.

⭘ Model Evaluation with 5-Fold Cross-Validation

To ensure model's performance is not biased by dataset's specific partitioning, k-fold cross-validation was implemented. This method divides the training set into k number of folds, using k-1 for training and the remaining one for validation. This process repeats k times with each fold serving as the validation set once. The ***perform_cross_validation*** function encapsulates this logic and is used to find the optimal hyperparameter k.

```python
def perform_cross_validation(classifier, X, y, k_choices, num_folds=5, distance='L2'):
    fold_size = X.shape[0] // num_folds
    X_folds = np.array_split(X, num_folds)
    y_folds = np.array_split(y, num_folds)
    k_to_accuracies = {}

    for k in k_choices:
        k_to_accuracies[k] = []
        for fold in range(num_folds):
            X_train = np.concatenate([X_folds[i] for i in range(num_folds) if i != fold])
            y_train = np.concatenate([y_folds[i] for i in range(num_folds) if i != fold])
            X_val = X_folds[fold]
            y_val = y_folds[fold]

            classifier.train(X_train, y_train)
            y_val_pred = classifier.predict(X_val, k=k, distance=distance)
            accuracy = np.mean(y_val_pred == y_val)
            k_to_accuracies[k].append(accuracy)

    return k_to_accuracies

def plot_cross_validation_results(k_choices, k_to_accuracies, title="Cross-validation Accuracy"):
    plt.figure(figsize=(12, 6))
    for k in k_choices:
        accuracies = k_to_accuracies[k]
        plt.scatter([k] * len(accuracies), accuracies)

    accuracies_mean = np.array([np.mean(v) for k, v in sorted(k_to_accuracies.items())])
    accuracies_std = np.array([np.std(v) for k, v in sorted(k_to_accuracies.items())])
    plt.errorbar(k_choices, accuracies_mean, yerr=accuracies_std, fmt='-o')
    plt.title(title)
    plt.xlabel('k')
    plt.ylabel('Cross-validation accuracy')
    plt.xticks(k_choices)
    plt.show()
```

● Implementation in the Main Function

In the main execution of the program, classifier and initiated the cross-validation process was implemented for both L1 and L2 distance metrics. Iteration continued over a predefined list of k values.

```
k_choices = [1, 3, 5, 8, 10, 12, 15, 20, 50, 100]
k_to_accuracies_L2 = perform_cross_validation(classifier, X_train,
y_train, k_choices, num_folds=5, distance='L2')
```

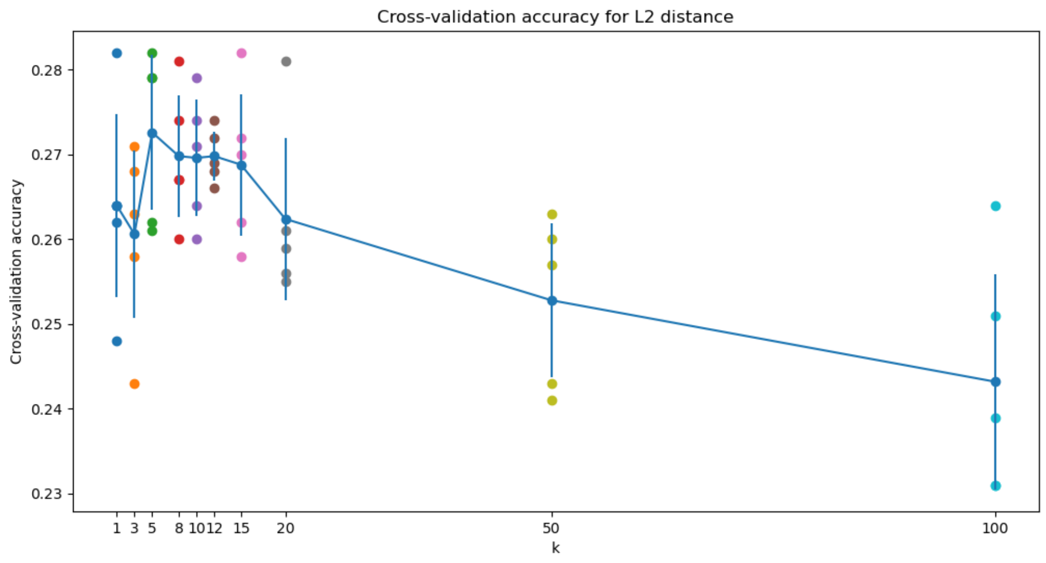
With the **k_choices** array defined, the **perform_cross_validation** function was called with the L2 distance metric, storing the accuracies for each k value in the **k_to_accuracies_L2** dictionary Performing Cross-Validation.

● Visualizing Cross-Validation Results of L2

The function ***plot_cross_validation_results*** was called to visualize the L2 Cross-Validation accuracy. The following parameters were passed- ***k_choice, k_to_accuracies_L2*** and the Title to the function. ***k_choices*** is an array or list of the different values of k to evaluate in the k-NN algorithm. k represents the number of nearest neighbors to consider when making a prediction. For example, if ***k_choices*** is [1, 3, 5, 8, 10], the function will plot the cross-validation results for these specific k values. ***k_to_accuracies_L2*** is a dictionary where each key corresponds to a value of k from ***k_choices***

```
plot_cross_validation_results(k_choices, k_to_accuracies_L2, "Cross-
validation accuracy for L2 distance")
```

Which plots the graph produced by the function ***plot_cross_validation_results*** visually represents the cross-validation accuracy as a function of varying k values for the k-NN classifier using the L2 distance metric. Here is a detailed description of the graph based on its typical features:



Cross-Validation accuracy for L2 distance

X-axis (k values): This axis represents the different k values specified in the ***k_choices*** array. It shows the range of k values for which the model's accuracy has been evaluated.

Y-axis (Accuracy): The vertical axis quantifies the accuracy obtained from the cross-validation process. Each point on this axis represents the average accuracy across all folds for a particular k.

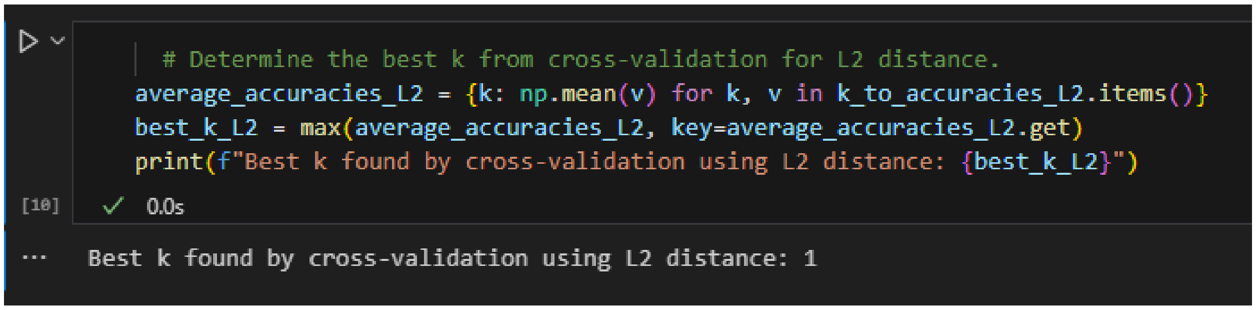
Data Points: Each data point on the graph corresponds to the average accuracy of a particular k value computed from the 5-fold cross-validation. The specific position on the Y-axis indicates the accuracy level, while its position on the X-axis shows the k value it's associated with.

Error Bars: The vertical lines extending from each data point represent the changeability in accuracy across the folds.

Line Connecting Points: The line threading through the data points helps visualize the trend in accuracy as k increases.

● Optimal K value

The best k value based on which one provided the highest average accuracy was selected across all folds for the L2 distance (as the L2 distance was the primary focus of the assignment). This output helped us identify the k value that generally performs best with the L2 distance metric. So the below code was implemented.



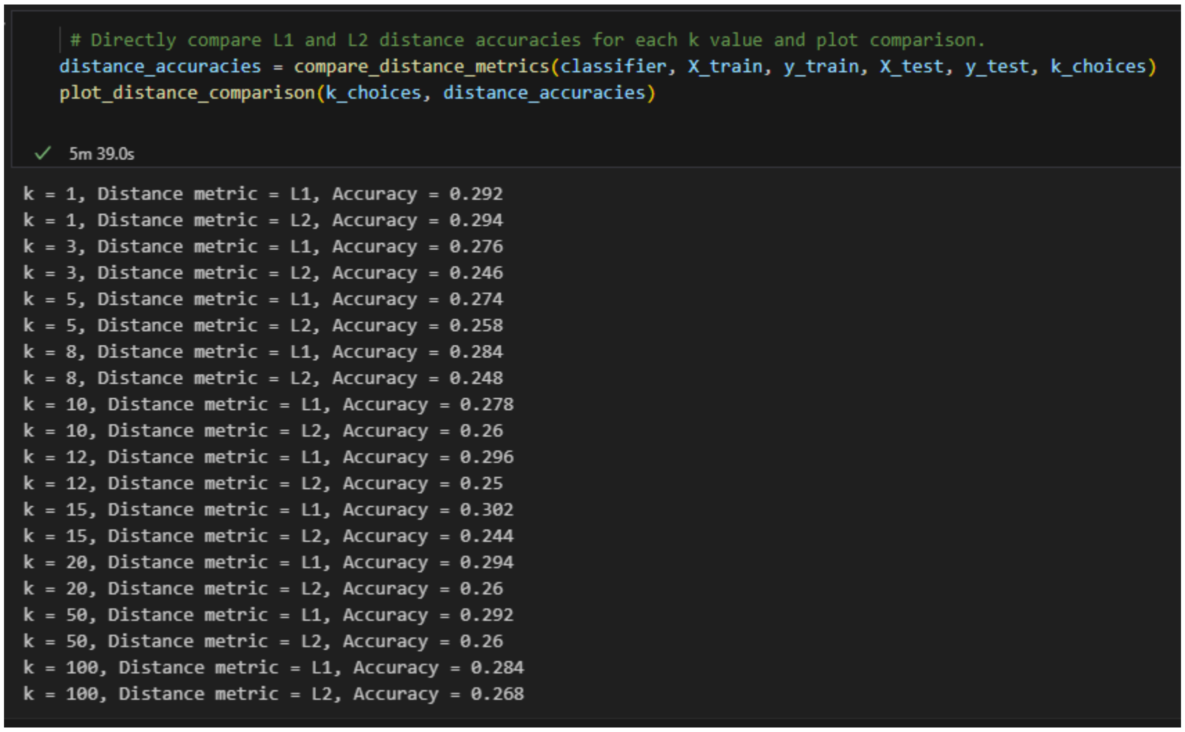
```
# Determine the best k from cross-validation for L2 distance.
average_accuracies_L2 = {k: np.mean(v) for k, v in k_to_accuracies_L2.items()}
best_k_L2 = max(average_accuracies_L2, key=average_accuracies_L2.get)
print(f"Best k found by cross-validation using L2 distance: {best_k_L2}")
```

```
Best k found by cross-validation using L2 distance: 1
```

In this following case the best k by cross-validation using L2 distance was 1

⭘ Distance Metric Comparison

After establishing the best k value using the L2 metric, both L1 and L2 metrics were compared. The ***compare_distance_metrics*** function provided us with the accuracies for each metric.



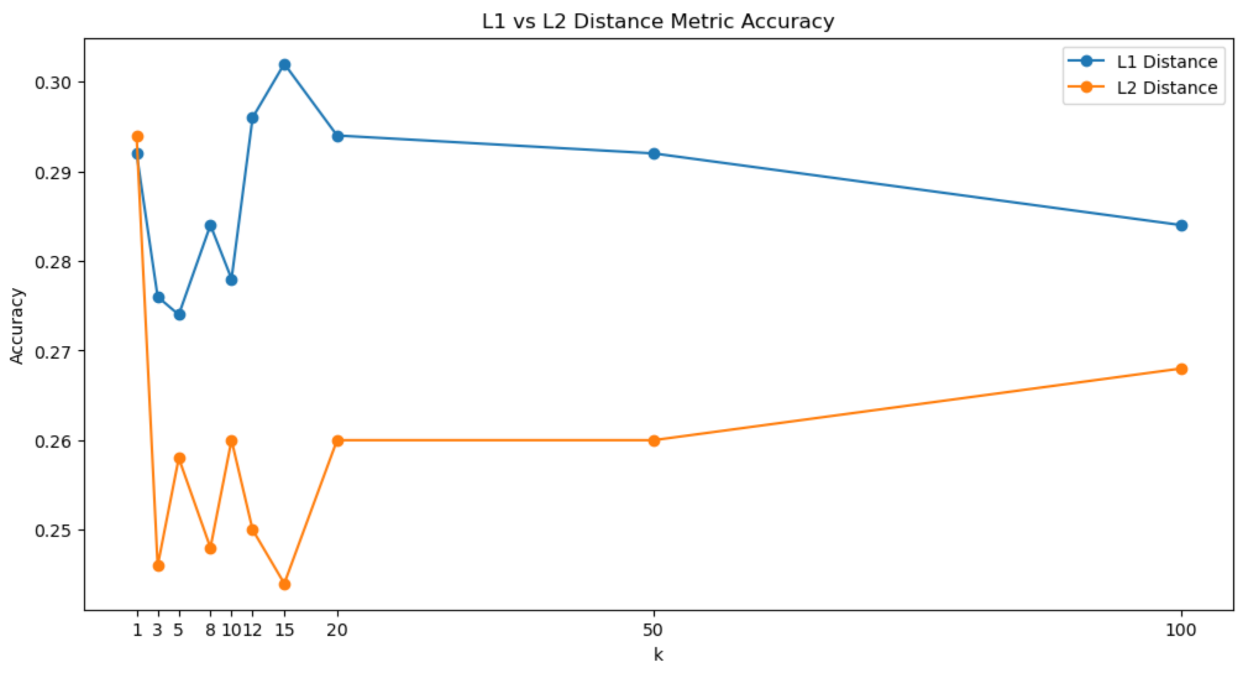
```
# Directly compare L1 and L2 distance accuracies for each k value and plot comparison.
distance_accuracies = compare_distance_metrics(classifier, X_train, y_train, X_test, y_test, k_choices)
plot_distance_comparison(k_choices, distance_accuracies)
```

✓ 5m 39.0s

```
k = 1, Distance metric = L1, Accuracy = 0.292
k = 1, Distance metric = L2, Accuracy = 0.294
k = 3, Distance metric = L1, Accuracy = 0.276
k = 3, Distance metric = L2, Accuracy = 0.246
k = 5, Distance metric = L1, Accuracy = 0.274
k = 5, Distance metric = L2, Accuracy = 0.258
k = 8, Distance metric = L1, Accuracy = 0.284
k = 8, Distance metric = L2, Accuracy = 0.248
k = 10, Distance metric = L1, Accuracy = 0.278
k = 10, Distance metric = L2, Accuracy = 0.26
k = 12, Distance metric = L1, Accuracy = 0.296
k = 12, Distance metric = L2, Accuracy = 0.25
k = 15, Distance metric = L1, Accuracy = 0.302
k = 15, Distance metric = L2, Accuracy = 0.244
k = 20, Distance metric = L1, Accuracy = 0.294
k = 20, Distance metric = L2, Accuracy = 0.26
k = 50, Distance metric = L1, Accuracy = 0.292
k = 50, Distance metric = L2, Accuracy = 0.26
k = 100, Distance metric = L1, Accuracy = 0.284
k = 100, Distance metric = L2, Accuracy = 0.268
```

Graphical Representation

following the presentation of the plot generated by the ***plot_distance_comparison*** function, we can observe a graphical trend that provides valuable insight into the performance of the k Nearest Neighbors classifier using two distinct distance metrics.



L1 vs L2 Metric accuracy

By examining the plot, it can be observed how the classifier's accuracy varied between using the L1 and L2 distance metrics, providing valuable insights into the impact of distance metric selection on classifier performance.

It is apparent that the accuracy associated with the L1 distance metric undergoes significant fluctuations. Initially, there is a marked variance in accuracy, which suggests a sensitivity to the choice of k. where as the accuracy of the L2 distance metric exhibits a less volatile behavior. Both metrics show a decreasing trend in accuracy at higher values of k, specifically at k=50 and k=100.

Conclusion of the Implementation

In the exploration of the CIFAR-10 dataset using the k-Nearest Neighbors (k-NN) classifier with a Euclidean (L2) distance metric, the efficacy of the model was substantiated through a 5-fold cross-validation approach.

Results indicate that while k-NN is capable of decent performance, it may not be the most efficient or powerful method available, especially in comparison to more advanced architectures like Neural Networks (NN). These advanced models could leverage the data's categorized structure and spatial relationships more effectively, potentially leading to higher accuracy and better generalization. Nevertheless, k-NN serves as an excellent baseline and educational tool for understanding the foundational concepts of image classification and machine learning.

Moreover, the comparison between Manhattan (L1) and Euclidean (L2) distance metrics offered valuable insights. Although L1 and L2 performed similarly, the slight variances in their accuracies could guide the selection of a distance metric based on the specific geometry of the dataset’s feature space.

In conclusion, this not only highlighted the practical aspects of implementing k-NN but also brought forth the importance of cross-validation and hyperparameter tuning in machine learning workflows. Despite the inherent limitations of k-NN and the challenges posed by high-dimensional datasets like CIFAR-10, the analytical techniques employed here lay a solid foundation for future exploration and development of more advanced image classification systems.