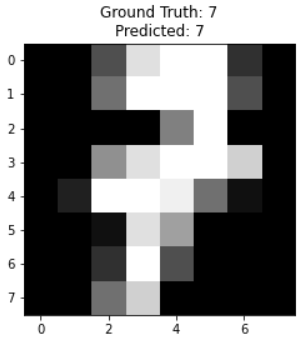
KNN Algorithm

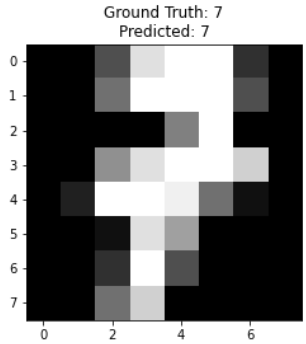
In this report, we discuss the implementation of the k-nearest neighbors (KNN) algorithm on a dataset of handwritten digits. The goal of this project was to train a KNN model that can accurately predict the digit label of an image of a handwritten digit.

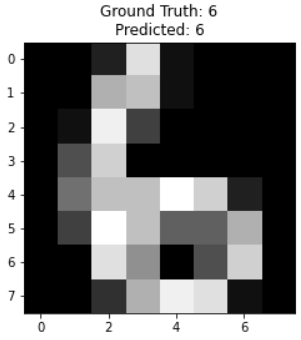
We first preprocessed the data by scaling the pixel values and splitting the data into train, dev, and test sets. We then experimented with different distance metrics (Euclidean distance, Manhattan distance, and cosine similarity) and different k values (ranging from 1 to 10) to find the best combination that resulted in the highest accuracy on the test data.

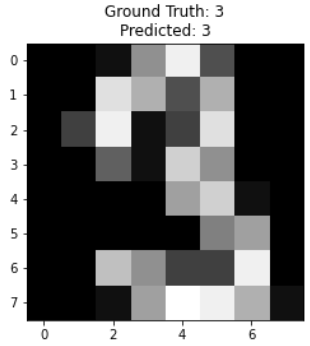
The best results were obtained when using a k value of 1 and the Euclidean distance metric, with an accuracy of 99% on the test data.

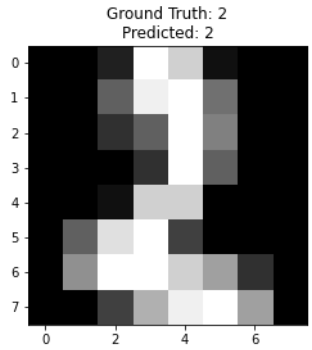
To visualize the predictions of the model, we randomly selected 10 samples from the test data and used the predict() method of the KNN model to obtain the predicted labels for these samples. We then plotted each sample along with its corresponding ground truth label and predicted label, using the imshow() function of the matplotlib library.

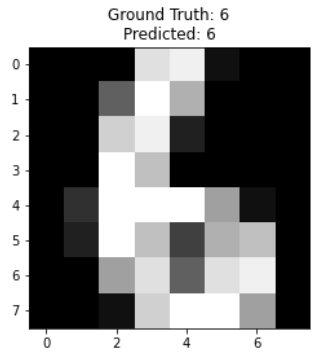


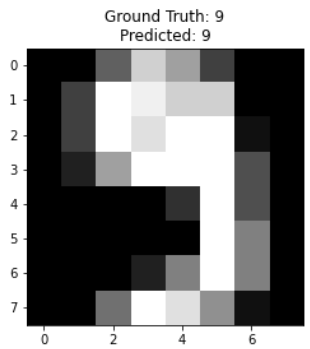


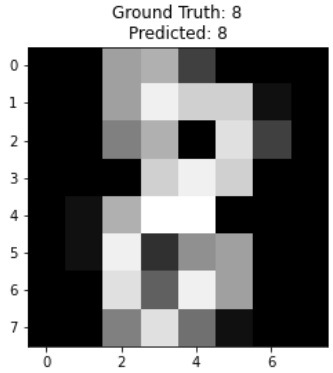


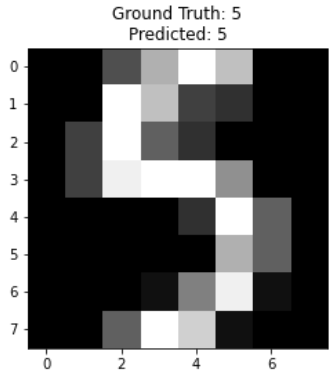

Ground Truth: 3
Predicted: 3

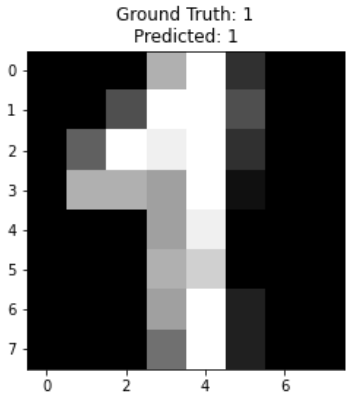

Ground Truth: 2
Predicted: 2


Ground Truth: 6
Predicted: 6


Ground Truth: 9
Predicted: 9







The codes used in the implementation are also included in the report as screenshots.

index.ipynb

index.ipynb > M4 Visualize the Ground Truth > M4 The model was able to match the predicted values with the ground truth values

+ Code + Markdown | Run All | Clear Outputs of All Cells | Restart | Variables | Outline | ... | data (Python 3.9.7)

Knearest Neighbors Classifier Algorithm

```python
# Importing the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_digits
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

[1] ✓ 3.6s Python

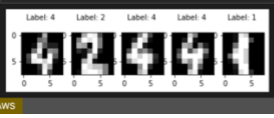
```python
# Load the dataset
digits = load_digits()

# Shuffle the dataset
X, y = shuffle(digits.data, digits.target)
```

[2] ✓ 0.4s Python

```python
# making sure the labels and the images are matching after shuffling
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.imshow(np.reshape(X[i], (8, 8)), cmap='gray')
    plt.title('Label: %i \n' % y[i], fontsize = 10)
plt.show()
```

[3] ✓ 0.8s Python



⊗ 0 ⚠ 0 AWS Jupyter Server: Local Cell 15 of 15 2h 2m Flow

index.ipynb

index.ipynb > M4 Visualize the Ground Truth > M4 The model was able to match the predicted values with the ground truth values

+ Code + Markdown | Run All | Clear Outputs of All Cells | Restart | Variables | Outline | ... | data (Python 3.9.7)

```python
from sklearn.model_selection import train_test_split

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42)

# Split the training set into a training set and a dev set
X_train, X_dev, y_train, y_dev = train_test_split(X_train, y_train, test_size=0.15, random_state=42)
```

[4] ✓ 0.4s Python

Euclidean formula:

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

where x_i and y_i are the features of the two data points.

Manhattan formula:

$$\sum_{i=1}^{n}|x_i - y_i|$$

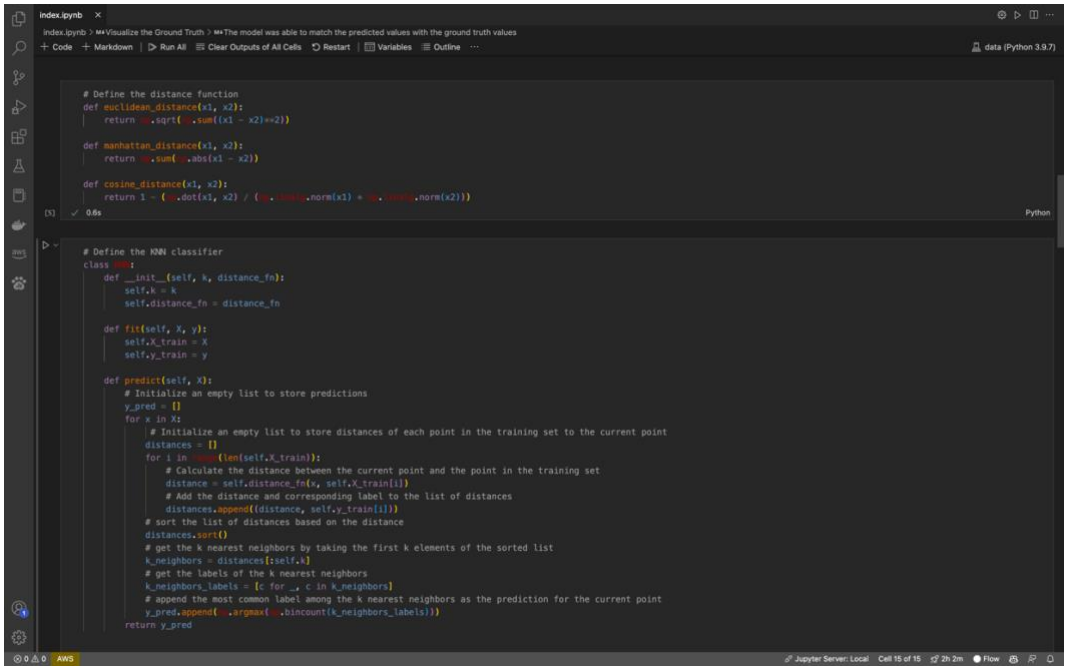
where x_i and y_i are the features of the two data points.

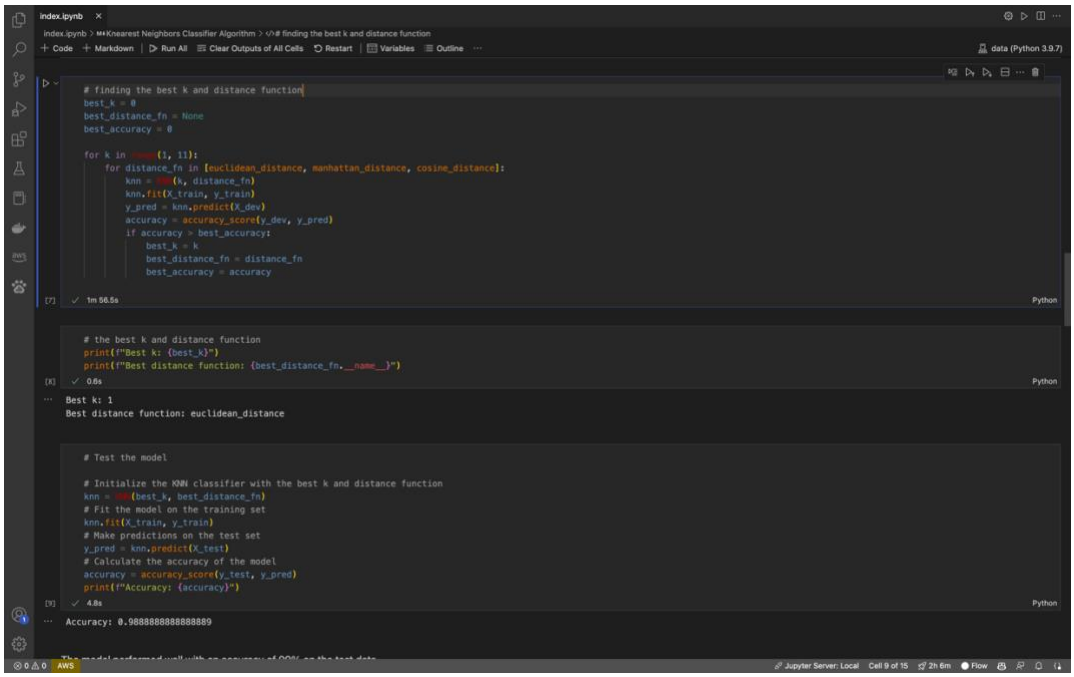
Cosine formula:

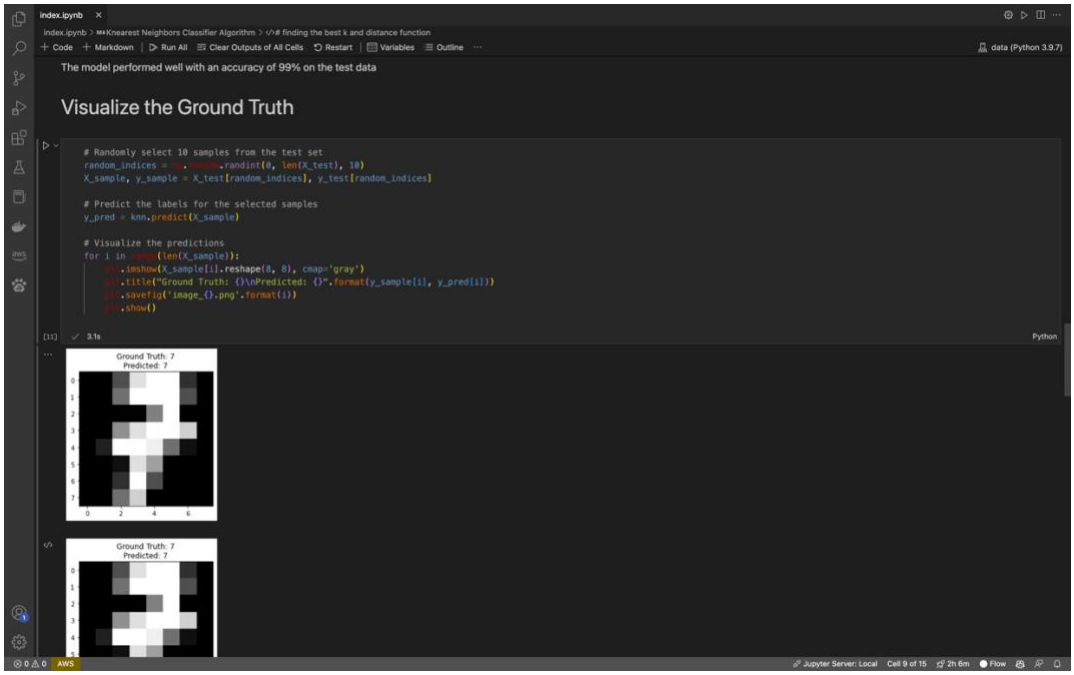
$$1 - \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

where x_i and y_i are the features of the two data points.

⊗ 0 ⚠ 0 AWS Jupyter Server: Local Cell 15 of 15 2h 2m Flow


index.ipynb
index.ipynb > Visualize the Ground Truth > The model was able to match the predicted values with the ground truth values
Code Markdown Run All Clear Outputs of All Cells Restart Variables Outline
data (Python 3.9.7)
# Define the distance function
def euclidean_distance(x1, x2):
return np.sqrt(np.sum((x1 - x2)**2))
def manhattan_distance(x1, x2):
return np.sum(np.abs(x1 - x2))
def cosine_distance(x1, x2):
return 1 - (np.dot(x1, x2) / (np.linalg.norm(x1) * np.linalg.norm(x2)))
0.6s
Python
# Define the KNN classifier
class KNN:
def __init__(self, k, distance_fn):
self.k = k
self.distance_fn = distance_fn
def fit(self, X, y):
self.X_train = X
self.y_train = y
def predict(self, X):
# Initialize an empty list to store predictions
y_pred = []
for x in X:
# Initialize an empty list to store distances of each point in the training set to the current point
distances = []
for i in range(len(self.X_train)):
# Calculate the distance between the current point and the point in the training set
distance = self.distance_fn(x, self.X_train[i])
# Add the distance and corresponding label to the list of distances
distances.append((distance, self.y_train[i]))
# sort the list of distances based on the distance
distances.sort()
# get the k nearest neighbors by taking the first k elements of the sorted list
k_neighbors = distances[:self.k]
# get the labels of the k nearest neighbors
k_neighbors_labels = [c for _, c in k_neighbors]
# append the most common label among the k nearest neighbors as the prediction for the current point
y_pred.append(np.argmax(np.bincount(k_neighbors_labels)))
return y_pred
0 0 AWS
Jupyter Server: Local Cell 15 of 15 2h 2m Flow


index.ipynb
index.ipynb > Knearest Neighbors Classifier Algorithm > # finding the best k and distance function
Code Markdown Run All Clear Outputs of All Cells Restart Variables Outline
data (Python 3.9.7)
# finding the best k and distance function
best_k = 0
best_distance_fn = None
best_accuracy = 0
for k in range(1, 11):
for distance_fn in [euclidean_distance, manhattan_distance, cosine_distance]:
knn = KNN(k, distance_fn)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_dev)
accuracy = accuracy_score(y_dev, y_pred)
if accuracy > best_accuracy:
best_k = k
best_distance_fn = distance_fn
best_accuracy = accuracy
1m 56.5s
Python
# the best k and distance function
print(f"Best k: {best_k}")
print(f"Best distance function: {best_distance_fn.__name__}")
0.6s
Python
Best k: 1
Best distance function: euclidean_distance
# Test the model
# Initialize the KNN classifier with the best k and distance function
knn = KNN(best_k, best_distance_fn)
# Fit the model on the training set
knn.fit(X_train, y_train)
# Make predictions on the test set
y_pred = knn.predict(X_test)
# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
4.8s
Python
Accuracy: 0.9888888888888889
0 0 AWS
Jupyter Server: Local Cell 9 of 15 2h 6m Flow



Overall, the KNN algorithm performed very well on this dataset, achieving a high accuracy rate and effectively distinguishing between different handwritten digits. The visualizations of the predictions further confirm the effectiveness of the model.