Machine Learning Homework 02

Author: Jesus Arturo Sol Navarro

1 Problem 8

Prove or disprove: Consider two arbitrary points $x, y \in R^2$. If x is the nearest neighbor of y regarding the L2-norm, then x is the nearest neighbor of y regarding the L1-norm.

1.1 Solution

We will use the following points as a counterexample:
- Target point: $y = (0, 0)$
- Point $x_1 = (2, 0)$
- Point $x_2 = (1.5, 1)$

For the L2 - distance between $y = (0, 0)$ and $x_1 = (2, 0)$:

$$d_2(y, x_1) = \sqrt{(2-0)^2 + (0-0)^2} = \sqrt{4} = 2$$

For the L2 - distance between $y = (0, 0)$ and $x_2 = (1.5, 1)$:

$$d_2(y, x_2) = \sqrt{(1.5-0)^2 + (1-0)^2} = \sqrt{2.25+1} = \sqrt{3.25} \approx 1.803$$

Result in L2 norm: The nearest neighbor to $y = (0, 0)$ in the L2 norm is $x_2 = (1.5, 1)$, as it has a smaller distance of approximately 1.803, compared to the distance of 2 for $x_1 = (2, 0)$.
For the L1 - distance between $y = (0, 0)$ and $x_1 = (2, 0)$:

$$d_1(y, x_1) = |2-0| + |0-0| = 2 + 0 = 2$$

For the L1 - distance between $y = (0, 0)$ and $x_2 = (1.5, 1)$:

$$d_1(y, x_2) = |1.5-0| + |1-0| = 1.5 + 1 = 2.5$$

Result in L1 norm: The nearest neighbor to $y = (0, 0)$ in the L1 norm is $x_1 = (2, 0)$, as it has a smaller distance of 2, compared to the distance of 2.5 for $x_2 = (1.5, 1)$.
- In the L2 norm, the nearest neighbor to $y = (0, 0)$ is $x_2 = (1.5, 1)$ with a distance of approximately 1.803.
- In the L1 norm, the nearest neighbor to $y = (0, 0)$ is $x_1 = (2, 0)$ with a distance of 2.

This disproves the statement that if x is the nearest neighbor of y in the L2 norm, then x must also be the nearest neighbor in the L1 norm.

2 Problem 9

2.1 Load dataset (Preparation)

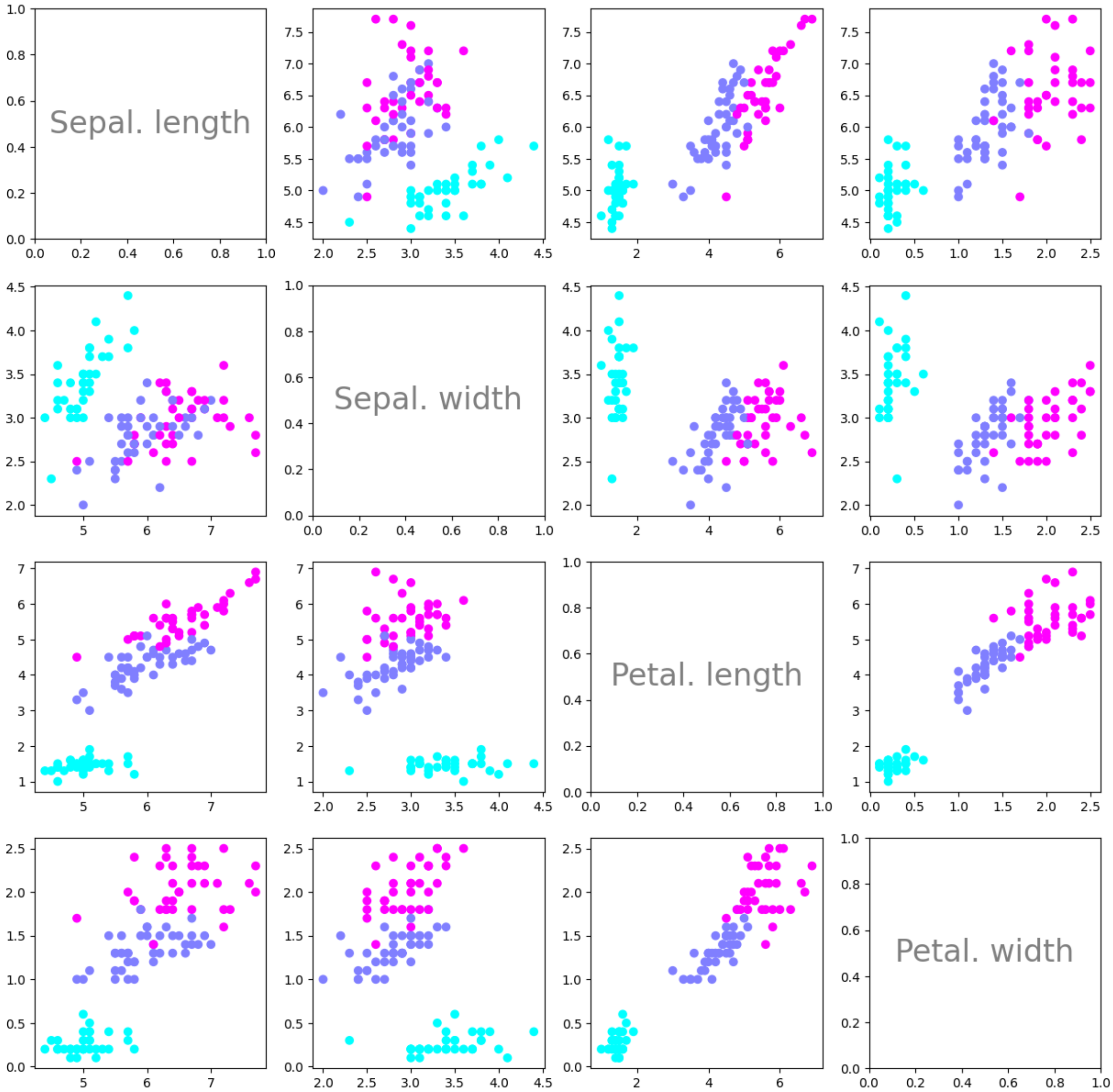
```
def load_dataset(split):
    """Load and split the dataset into training and test parts.

    Parameters
    ----------
    split : float in range (0, 1)
        Fraction of the data used for training.

    Returns
    -------
    X_train : array, shape (N_train, 4)
        Training features.
    y_train : array, shape (N_train)
        Training labels.
    X_test : array, shape (N_test, 4)
        Test features.
    y_test : array, shape (N_test)
        Test labels.
    """
```

```python
    dataset = datasets.load_iris()
    X, y = dataset['data'], dataset['target']
    X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, random_state=123,
                                                                        test_size=(1 - split))
    return X_train, X_test, y_train, y_test

# prepare data
split = 0.75
X_train, X_test, y_train, y_test = load_dataset(split)
f, axes = plt.subplots(4, 4,figsize=(15, 15))
for i in range(4):
    for j in range(4):
        if j == 0 and i == 0:
            axes[i,j].text(0.5, 0.5, 'Sepal. length', ha='center', va='center', size=24, alpha=.5)
        elif j == 1 and i == 1:
            axes[i,j].text(0.5, 0.5, 'Sepal. width', ha='center', va='center', size=24, alpha=.5)
        elif j == 2 and i == 2:
            axes[i,j].text(0.5, 0.5, 'Petal. length', ha='center', va='center', size=24, alpha=.5)
        elif j == 3 and i == 3:
            axes[i,j].text(0.5, 0.5, 'Petal. width', ha='center', va='center', size=24, alpha=.5)
        else:
            axes[i,j].scatter(X_train[:,j],X_train[:,i], c=y_train, cmap=plt.cm.cool)
```



2.2 Task: Euclidean distance

```python
def euclidean_distance(x1, x2):
    """Compute pairwise Euclidean distances between two data points.

    Parameters
    ----------
    x1 : array, shape (N, 4)
        First set of data points.
    x2 : array, shape (M, 4)
        Second set of data points.

    Returns
    -------
    distance : float array, shape (N, M)
        Pairwise Euclidean distances between x1 and x2.
    """
    # TODO

    dis=np.linalg.norm(x1-x2)

    return dis
```

2.3 Task: get k nearest neighbors' labels

```
def get_neighbors_labels(X_train, y_train, X_new, k):
    """
    Get the labels of the k nearest neighbors of the datapoints x_new.

    Parameters
    ----------
    X_train : array, shape (N_train, D)
        Training features.
    y_train : array, shape (N_train,)
        Training labels.
    X_new : array, shape (M, D)
        Data points for which the neighbors have to be found.
    k : int
        Number of neighbors to return.

    Returns
    -------
    neighbors_labels : array, shape (M, k)
        Array containing the labels of the k nearest neighbors for each data point in X_new.
    """
    distances = np.sqrt(((X_new[:, np.newaxis, :] - X_train) ** 2).sum(axis=2))
    nearest_indices = np.argsort(distances, axis=1)[:, :k]
    neighbors_labels = y_train[nearest_indices]

    return neighbors_labels
```

2.4 Task: get the majority label

```
def get_response(neighbors_labels, num_classes=3):
    """Predict label given the set of neighbors.

    Parameters
    ----------
    neighbors_labels : array, shape (M, k)
        Array containing the labels of the k nearest neighbors per data point.
    num_classes : int
        Number of classes in the dataset.

    Returns
    -------
    y : int array, shape (M,)
        Majority class among the neighbors.
    """
    # TODO
    M = neighbors_labels.shape[0]
    predictions = np.zeros(M, dtype=int)

    for i in range(M):
        counts = np.bincount(neighbors_labels[i], minlength=num_classes)
        predictions[i] = np.argmax(counts)

    return predictions
```

2.5 Task: compute accuracy

```
    def compute_accuracy(y_pred, y_test):
    """Compute accuracy of prediction.

    Parameters
    ----------
    y_pred : array, shape (N_test)
        Predicted labels.
    y_test : array, shape (N_test)
        True labels.
    """
    # TODO

    assert y_pred.shape == y_test.shape, "Shapes of y_pred and y_test must match"
    correct_predictions = np.sum(y_pred == y_test)
    accuracy = correct_predictions / len(y_test)

    return accuracy
```

2.6 Testing

```
    # This function is given, nothing to do here.
def predict(X_train, y_train, X_test, k):
    """Generate predictions for all points in the test set.

    Parameters
    ----------
    X_train : array, shape (N_train, 4)
        Training features.
    y_train : array, shape (N_train)
        Training labels.
    X_test : array, shape (N_test, 4)
        Test features.
    k : int
        Number of neighbors to consider.

    Returns
    -------
    y_pred : array, shape (N_test)
        Predictions for the test data.
    """
    neighbors = get_neighbors_labels(X_train, y_train, X_test, k)
    y_pred = get_response(neighbors)
    return y_pred
```

```
# prepare data
split = 0.75
X_train, X_test, y_train, y_test = load_dataset(split)
print('Training set: {0} samples'.format(X_train.shape[0]))
print('Test set: {0} samples'.format(X_test.shape[0]))

# generate predictions
k = 3
y_pred = predict(X_train, y_train, X_test, k)
accuracy = compute_accuracy(y_pred, y_test)
print('Accuracy = {0}'.format(accuracy))
```

2.7 Result:

Training set: 112 samples
Test set: 38 samples
Accuracy = 0.9473684210526315