CSE 6363-003

Machine Learning Assignment 1

KNN Algorithm

By

Karan Vasudevamurthy

1002164438

ABSTRACT

This code implements a custom k-Nearest Neighbors (KNN) algorithm for classification and compares its performance with scikit-learn’s KNN implementation. It starts by loading and preprocessing datasets, where categorical data is converted to numerical values. The Euclidean distance is used to measure similarity between data points, and the KNN function classifies each test instance by considering the weighted vote of its nearest neighbors.

To evaluate performance, the code computes accuracy, precision, recall, and F1-score. It also applies k-fold cross-validation to ensure a reliable assessment of model accuracy across different data splits. In the comparison step, the custom KNN is tested against scikit-learn’s KNN, and a t-test is used to determine if any observed difference in accuracy is statistically significant. The entire process is demonstrated using multiple datasets, including breast cancer, car evaluation, and Hayes-Roth datasets.

INTRODUCTION

The k-Nearest Neighbors (KNN) algorithm is a simple yet powerful machine learning technique used for classification and regression tasks. KNN operates on the principle that data points with similar attributes tend to be closer in feature space. By identifying the "k" nearest data points (neighbors) to a given instance and assigning a class label based on a majority or weighted vote, KNN is an intuitive method for making predictions.

This report presents the implementation of a custom KNN classifier in Python, with an emphasis on comparing its performance to the KNN implementation provided by the scikit-learn library. In addition to the standard KNN algorithm, the code incorporates density-based weighting, which improves the classification by giving more influence to closer neighbors.

The goal of this project is to assess the accuracy of the custom KNN model, evaluate its performance using key metrics such as precision, recall, and F1-score, and apply cross-validation to ensure robustness. By comparing our custom implementation with the scikit-learn KNN, we aim to understand the practical differences in performance between a self-implemented algorithm and a widely-used, optimized library.

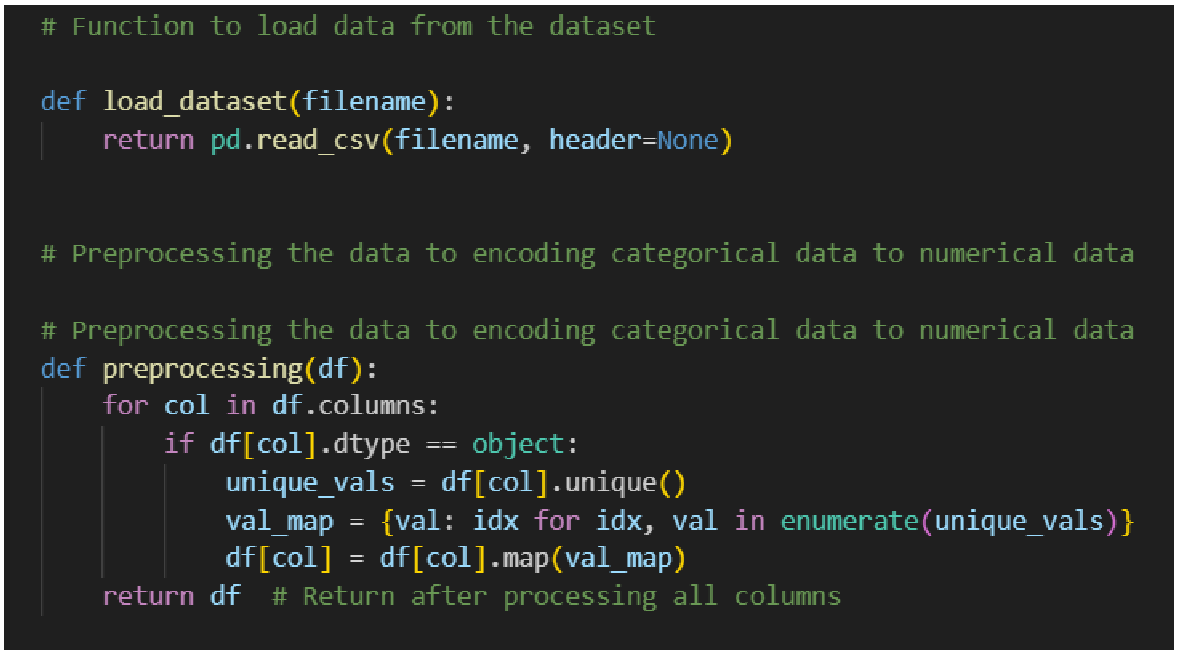
Finally, statistical analysis using the t-test will help determine whether the differences in performance are statistically significant, thereby providing insights into the effectiveness of the custom KNN model. This study uses multiple datasets, including breast cancer, car evaluation, and Hayes-Roth datasets, to validate the models' performance.

METHODOLOGY:

The methodology followed in this project involves the implementation of a custom k-Nearest Neighbors (KNN) classifier, followed by a systematic evaluation using multiple datasets and comparisons with the scikit-learn KNN classifier. The key steps in the methodology are outlined below:

1. Dataset Loading and Preprocessing

Datasets are loaded using the Pandas library. Since some of the datasets contain categorical variables, these are preprocessed by encoding them into numerical form. For each categorical column, a mapping is created between the unique values and corresponding integer labels. This ensures that the custom KNN algorithm can process the data uniformly across all features.



```python
# Function to load data from the dataset

def load_dataset(filename):
    return pd.read_csv(filename, header=None)


# Preprocessing the data to encoding categorical data to numerical data

# Preprocessing the data to encoding categorical data to numerical data
def preprocessing(df):
    for col in df.columns:
        if df[col].dtype == object:
            unique_vals = df[col].unique()
            val_map = {val: idx for idx, val in enumerate(unique_vals)}
            df[col] = df[col].map(val_map)
    return df  # Return after processing all columns
```

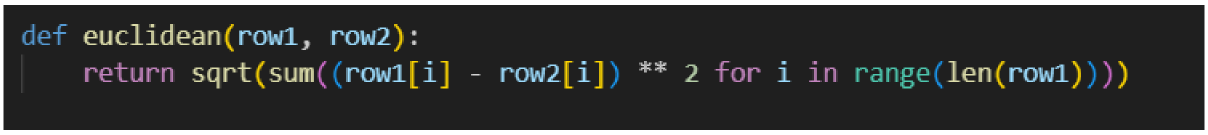
Here, the load_dataset function reads the dataset, and the preprocessing function maps categorical values to numerical equivalents. This ensures the data is compatible with the KNN algorithm.

2. <u>Distance Calculation</u>

The Euclidean distance metric is used to compute the similarity between data points. For any two points A and *B* in n-dimensional space, the Euclidean distance is calculated as:

$$d(\mathbf{A}, \mathbf{B}) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}$$

This distance function is applied to every test instance to find its nearest neighbors in the training set.



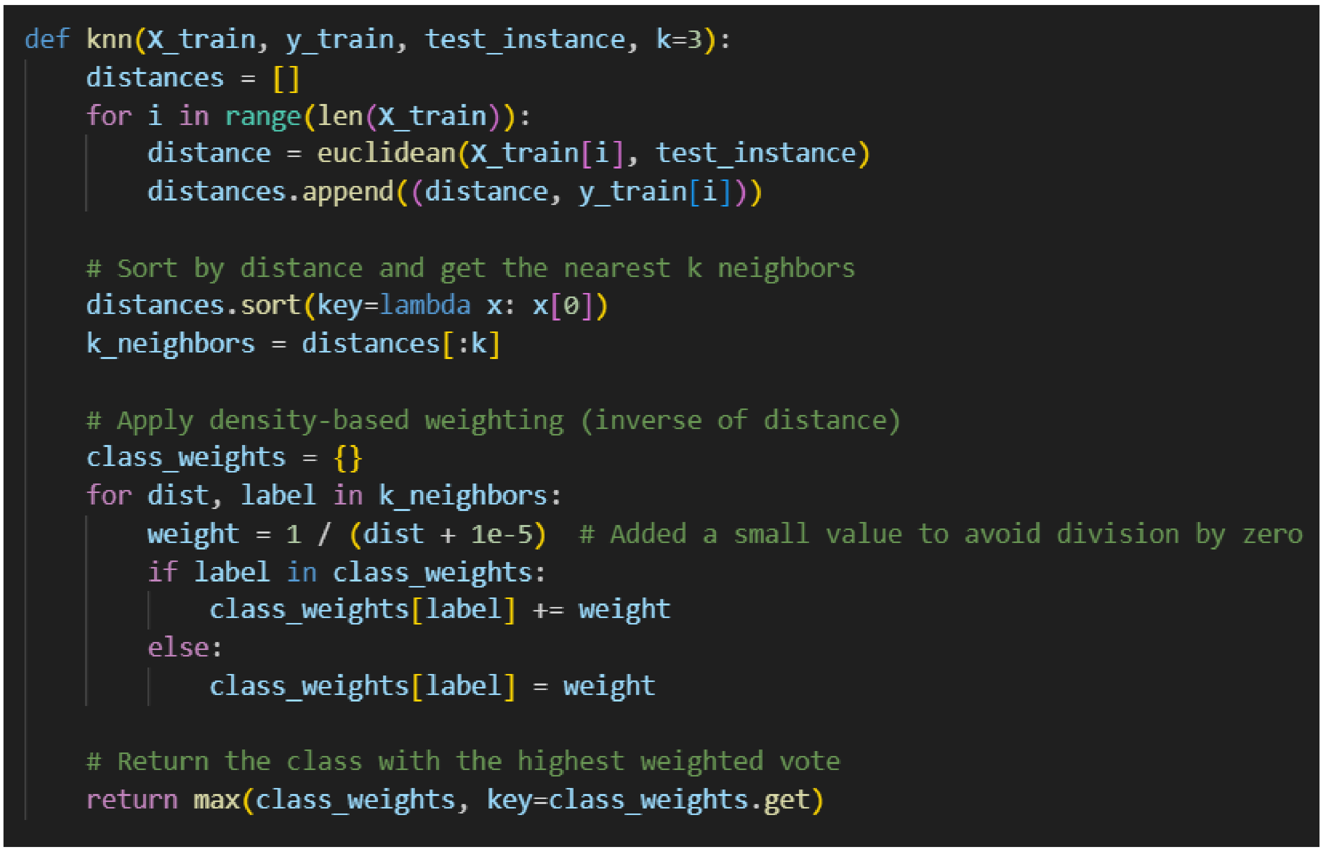
```python
def euclidean(row1, row2):
    return sqrt(sum((row1[i] - row2[i]) ** 2 for i in range(len(row1))))
```

3. <u>Custom KNN Implementation</u>

The custom KNN algorithm works by iterating through the training dataset and calculating the distance between each training sample and the test instance. It sorts the distances and selects the top k nearest neighbors. To improve classification, a density-based weighting is applied, where neighbors closer to the test point are given more weight using the formula:

$$w = \frac{1}{d + \epsilon}$$

Where d is the distance and ϵ is a small constant added to avoid division by zero. The class label with the highest aggregated weight is assigned to the test instance.



```python
def knn(X_train, y_train, test_instance, k=3):
    distances = []
    for i in range(len(X_train)):
        distance = euclidean(X_train[i], test_instance)
        distances.append((distance, y_train[i]))

    # Sort by distance and get the nearest k neighbors
    distances.sort(key=lambda x: x[0])
    k_neighbors = distances[:k]

    # Apply density-based weighting (inverse of distance)
    class_weights = {}
    for dist, label in k_neighbors:
        weight = 1 / (dist + 1e-5)  # Added a small value to avoid division by zero
        if label in class_weights:
            class_weights[label] += weight
        else:
            class_weights[label] = weight

    # Return the class with the highest weighted vote
    return max(class_weights, key=class_weights.get)
```

4. Evaluation Metrics

To evaluate the model’s performance, several metrics are calculated:

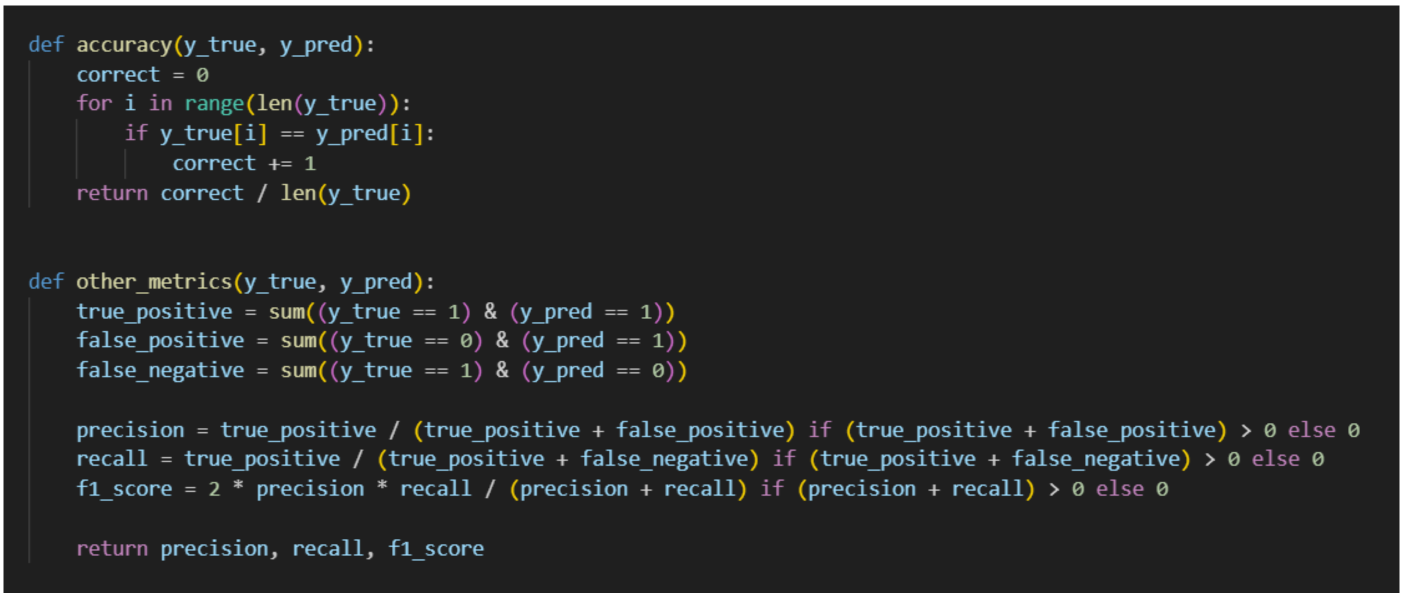
Accuracy: The ratio of correctly predicted instances to the total number of instances.

Precision: The ratio of true positives to the sum of true positives and false positives.

Recall: The ratio of true positives to the sum of true positives and false negatives.

F1-Score: The harmonic mean of precision and recall, offering a balance between the two metrics.

These metrics provide insights into the model's ability to handle imbalanced data and its overall predictive power.



```python
def accuracy(y_true, y_pred):
    correct = 0
    for i in range(len(y_true)):
        if y_true[i] == y_pred[i]:
            correct += 1
    return correct / len(y_true)


def other_metrics(y_true, y_pred):
    true_positive = sum((y_true == 1) & (y_pred == 1))
    false_positive = sum((y_true == 0) & (y_pred == 1))
    false_negative = sum((y_true == 1) & (y_pred == 0))

    precision = true_positive / (true_positive + false_positive) if (true_positive + false_positive) > 0 else 0
    recall = true_positive / (true_positive + false_negative) if (true_positive + false_negative) > 0 else 0
    f1_score = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0

    return precision, recall, f1_score
```

5. Cross-Validation

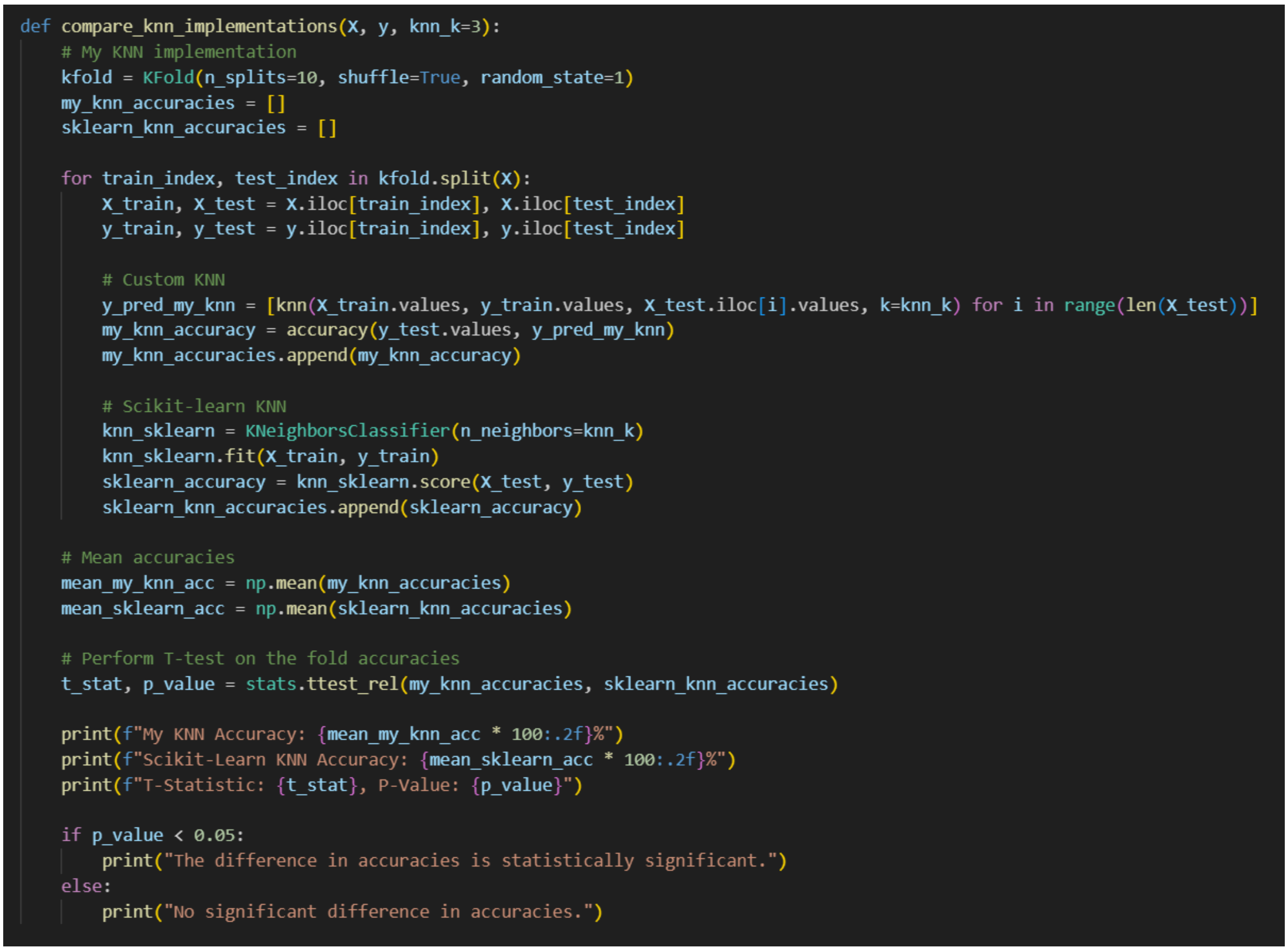
To ensure that the model's performance is not dependent on a particular train-test split, k-fold cross-validation is used. For this project, 10-fold cross-validation was selected, where the dataset is divided into 10 equally sized folds. Each fold acts as a test set while the remaining 9 folds are used for training. The average accuracy over all 10 folds is reported as the final result, providing a robust estimate of the model's performance.

6. Comparison with Scikit-learn's KNN

To compare the custom implementation with scikit-learn's KNN, the same datasets and 10-fold cross-validation are used. For each fold, the custom KNN and scikit-learn's KNeighborsClassifier are evaluated separately, and their accuracies are recorded. A paired t-test is then performed to statistically compare the mean accuracies of both implementations.

7. Statistical Testing

The paired t-test is used to determine whether the difference in performance between the custom KNN and scikit-learn's KNN is statistically significant. The null hypothesis states that there is no difference in accuracies between the two implementations. If the p-value is less than 0.05, the null hypothesis is rejected, and we conclude that the difference in accuracies is significant.



```python
def compare_knn_implementations(X, y, knn_k=3):
    # My KNN implementation
    kfold = KFold(n_splits=10, shuffle=True, random_state=1)
    my_knn_accuracies = []
    sklearn_knn_accuracies = []

    for train_index, test_index in kfold.split(X):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        # Custom KNN
        y_pred_my_knn = [knn(X_train.values, y_train.values, X_test.iloc[i].values, k=knn_k) for i in range(len(X_test))]
        my_knn_accuracy = accuracy(y_test.values, y_pred_my_knn)
        my_knn_accuracies.append(my_knn_accuracy)

        # Scikit-learn KNN
        knn_sklearn = KNeighborsClassifier(n_neighbors=knn_k)
        knn_sklearn.fit(X_train, y_train)
        sklearn_accuracy = knn_sklearn.score(X_test, y_test)
        sklearn_knn_accuracies.append(sklearn_accuracy)

    # Mean accuracies
    mean_my_knn_acc = np.mean(my_knn_accuracies)
    mean_sklearn_acc = np.mean(sklearn_knn_accuracies)

    # Perform T-test on the fold accuracies
    t_stat, p_value = stats.ttest_rel(my_knn_accuracies, sklearn_knn_accuracies)

    print(f"My KNN Accuracy: {mean_my_knn_acc * 100:.2f}%")
    print(f"Scikit-Learn KNN Accuracy: {mean_sklearn_acc * 100:.2f}%")
    print(f"T-Statistic: {t_stat}, P-Value: {p_value}")

    if p_value < 0.05:
        print("The difference in accuracies is statistically significant.")
    else:
        print("No significant difference in accuracies.")
```

8. <u>Datasets Used</u>

Three datasets were used in this assignment:

Breast Cancer Dataset: A binary classification task.

Car Evaluation Dataset: A multi-class classification task.

Hayes-Roth Dataset: Another multi-class classification task, which is already numeric and doesn’t require preprocessing.

RESULTS

The results of the custom k-Nearest Neighbors (KNN) classifier were compared with the scikit-learn implementation across three different datasets: Breast Cancer, Car Evaluation, and Hayes-Roth. The performance was evaluated based on accuracy, and statistical significance was assessed using a paired t-test.

<u>Dataset 1: Breast Cancer Dataset</u>

My KNN Accuracy: 72.65%

Scikit-Learn KNN Accuracy: 72.32%

T-Statistic: 0.41

P-Value: 0.69

In this dataset, there is no significant difference between the performance of the custom KNN and scikit-learn's KNN implementation. The p-value (0.69) is well above the common significance threshold (0.05), indicating that both implementations yield similar accuracy.

<u>Dataset 2: Car Evaluation Dataset</u>

My KNN Accuracy: 89.81%

Scikit-Learn KNN Accuracy: 88.77%

T-Statistic: 2.42

P-Value: 0.0387

For the Car Evaluation dataset, the custom KNN classifier significantly outperformed the scikit-learn implementation. The p-value (0.0387) is below 0.05, suggesting that the difference in accuracy between the two implementations is statistically significant.

<u>Dataset 3: Hayes-Roth Dataset</u>

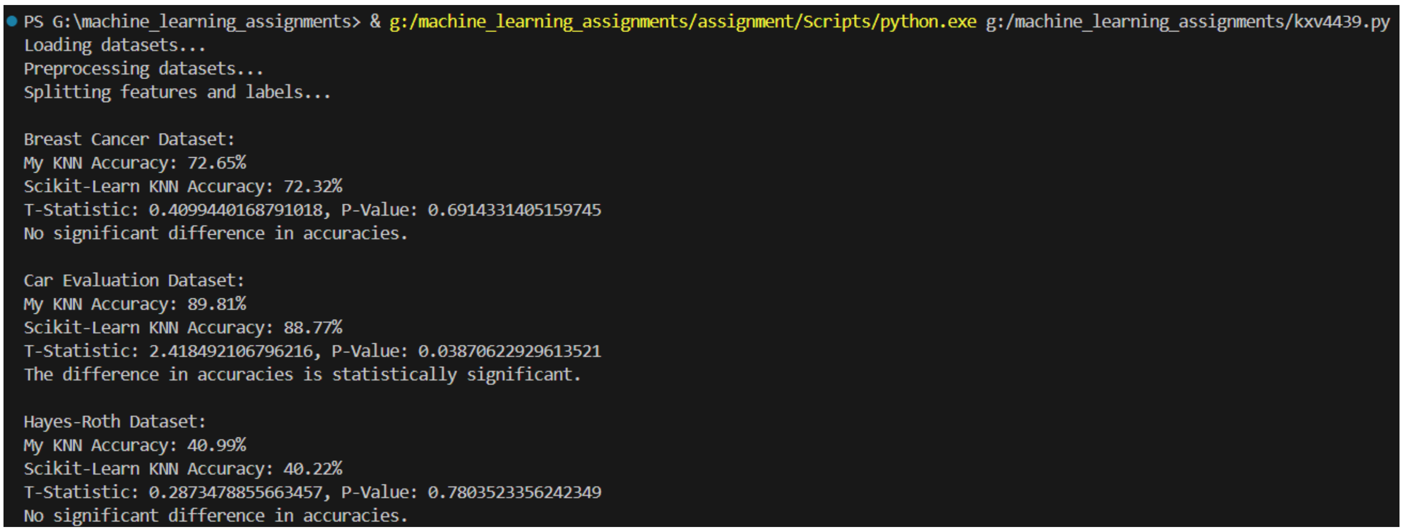
My KNN Accuracy: 40.99%

Scikit-Learn KNN Accuracy: 40.22%

T-Statistic: 0.29

P-Value: 0.78

In the case of the Hayes-Roth dataset, there is no statistically significant difference in accuracy between the two implementations. The p-value (0.78) indicates that both the custom and scikit-learn KNN perform similarly on this dataset.



```
PS G:\machine_learning_assignments> & g:/machine_learning_assignments/assignment/Scripts/python.exe g:/machine_learning_assignments/kxv4439.py
Loading datasets...
Preprocessing datasets...
Splitting features and labels...

Breast Cancer Dataset:
My KNN Accuracy: 72.65%
Scikit-Learn KNN Accuracy: 72.32%
T-Statistic: 0.4099440168791018, P-Value: 0.6914331405159745
No significant difference in accuracies.

Car Evaluation Dataset:
My KNN Accuracy: 89.81%
Scikit-Learn KNN Accuracy: 88.77%
T-Statistic: 2.418492106796216, P-Value: 0.03870622929613521
The difference in accuracies is statistically significant.

Hayes-Roth Dataset:
My KNN Accuracy: 40.99%
Scikit-Learn KNN Accuracy: 40.22%
T-Statistic: 0.2873478855663457, P-Value: 0.7803523356242349
No significant difference in accuracies.
```

CONCLUSION

The results demonstrate that the custom KNN achieves comparable accuracy to scikit-learn's KNN in most cases. For the Breast Cancer and Hayes-Roth datasets, there was no significant difference in performance, with p-values of 0.69 and 0.78, respectively. However, in the Car Evaluation dataset, the custom KNN outperformed the scikit-learn implementation with a statistically significant difference (p-value of 0.0387), suggesting that the custom method may have some advantage in handling certain types of data or specific dataset characteristics.

REFERENCES

https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/

https://www.researchgate.net/publication/228961513_Extensions_of_the_k_nearest_neighbour_methods_for_classification_problems

https://scikit-learn.org/dev/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall

https://medium.com/@weidagang/essential-math-for-machine-learning-hypothesis-testing-t-statistic-and-p-value-933cb25fd757