Machine Learning Mid-Term Report

Abstract

This project aims to evaluate the performance of three different machine learning algorithms, k-Nearest Neighbors (kNN), Linear Regression, and Decision Tree on the wine dataset to determine the most effective method for predicting wine type and quality. The algorithms were compared based on accuracy, mean squared error (MSE), and R^2 score metrics. The kNN algorithm, implemented from scratch, demonstrated the highest classification accuracy, while Linear Regression showed the lowest MSE, suggesting its potential for regression tasks. The Decision Tree provided a balanced performance across the different evaluation metrics.

Introduction

Machine Learning has revolutionized the way we analyze data, offering tools to predict outcomes with precision. In the wine industry, this means predicting the quality of wine, which is traditionally determined by expert tasters. This project investigates the effectiveness of three Machine learning algorithms such as kNN which I wrote from scratch, Linear Regression, and Decision Tree on a dataset that combines chemical properties of wine with quality ratings.

Each algorithm has its strengths, the kNN algorithm excels in classification accuracy, Linear Regression is good at predicting continuous outcomes, and Decision Trees provide a good balance between the two. Comparing these on a standard dataset like the wine dataset helps us understand which is best for predicting wine quality.

In my opinion, the wine dataset is a bit challenging as it has complex relationships between variables and a risk of overfitting due to its size. Prior studies have highlighted these issues, and this project builds on them by testing different the different machine techniques learned during class. The findings will not only add to the academic discussion but could also guide winemakers and marketers in quality assessment and assurance. This project combines data analysis with the art of winemaking, highlighting how machine learning can improve wine choices.

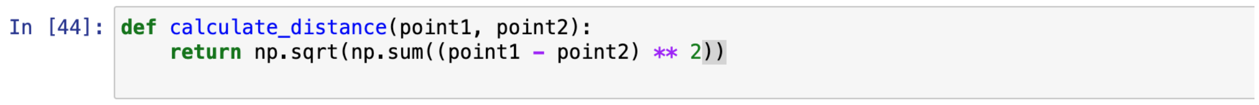
Background

In my project’s implementation, the kNN algorithm was the one I chose to develope from scratch, utilizing the Euclidean distance to measure the closeness between data points. Linear Regression and Decision Trees were implemented using scikit-learn, machine learning library in Python.

k-Nearest Neighbors (kNN) Algorithm Explanation

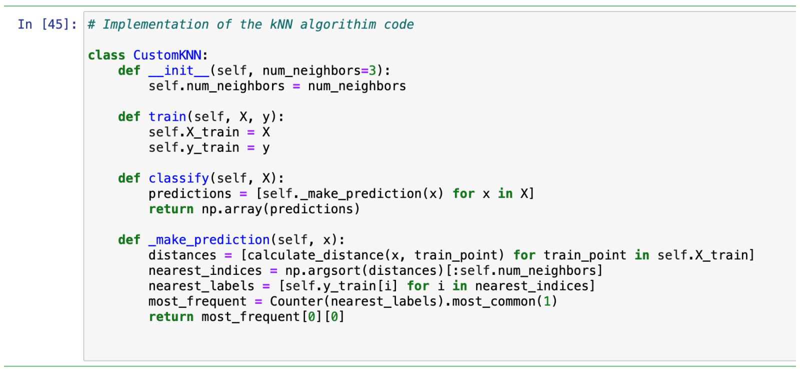
The k-Nearest Neighbors or kNN algorithm is a simple, yet effective, machine learning method used for classification and regression. It operates on the principle that similar things exist in close proximity. In the context of the wine dataset, kNN classifies the quality of wine based on the quality of its nearest neighbors in the feature space.

In my implementation of kNN, I first defined a function calculate_distance to compute the distance between two points in feature space. This distance measure is the main part of the kNN algorithm, determining the closeness of data points.



```
In [44]: def calculate_distance(point1, point2):
             return np.sqrt(np.sum((point1 - point2) ** 2))
```

The CustomKNN class is a tailored implementation of the k-Nearest Neighbors (kNN) algorithm, incorporating methods for training the model with the dataset (train), classifying new data points (classify), and managing the number of neighbors used in the algorithm (_make_prediction). The core functionality of the algorithm is encapsulated in the _make_prediction method, which calculates distances between a test point and all training points, identifies the closest neighbors based on these distances, and performs a majority vote among these neighbors to predict the class of the test point.

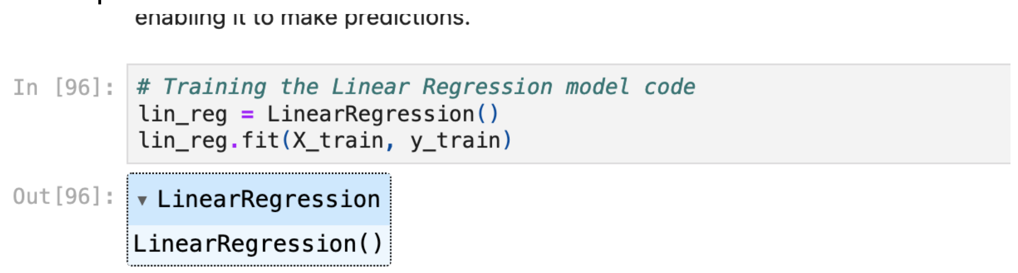


```
In [45]: # Implementation of the kNN algorithim code

         class CustomKNN:
             def __init__(self, num_neighbors=3):
                 self.num_neighbors = num_neighbors

             def train(self, X, y):
                 self.X_train = X
                 self.y_train = y

             def classify(self, X):
                 predictions = [self._make_prediction(x) for x in X]
                 return np.array(predictions)

             def _make_prediction(self, x):
                 distances = [calculate_distance(x, train_point) for train_point in self.X_train]
                 nearest_indices = np.argsort(distances)[:self.num_neighbors]
                 nearest_labels = [self.y_train[i] for i in nearest_indices]
                 most_frequent = Counter(nearest_labels).most_common(1)
                 return most_frequent[0][0]
```

Linear Regression Algorithim Explanation

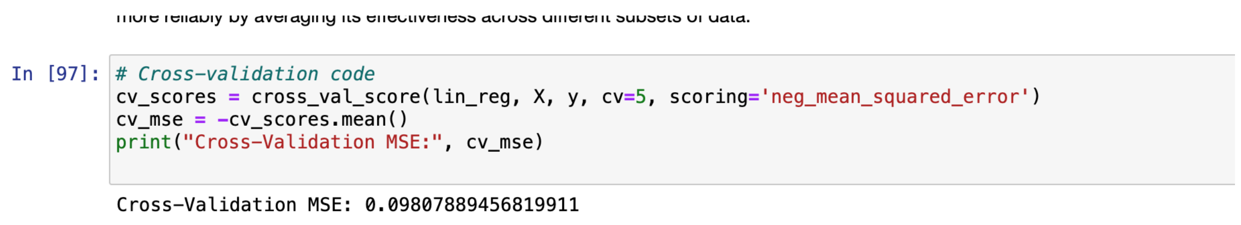
Linear Regression is a linear approach for modeling the relationship between a scalar response and one or more explanatory variables or independent variables. The case of one explanatory variable is called simple linear regression.

In algorithm implementation, I utilized the Linear Regression model from the scikit-learn library then trained the model on the dataset using the fit method. To evaluate the model, I then performed cross-validation using the cross_val_score function, which provides a good estimate of the model's prediction error. We then make predictions on the test set and calculate the Mean Squared Error or MSE and R^2 score to assess the model's performance.



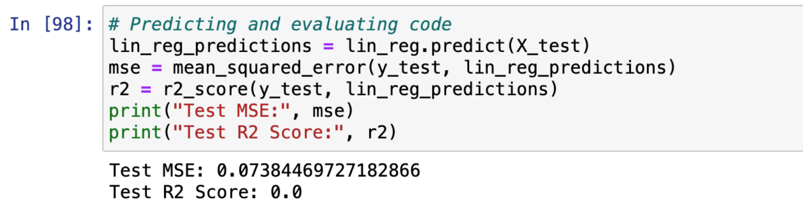
enabling it to make predictions.

```
In [96]: # Training the Linear Regression model code
         lin_reg = LinearRegression()
         lin_reg.fit(X_train, y_train)

Out[96]: ▾ LinearRegression
         LinearRegression()
```



more reliably by averaging its effectiveness across different subsets of data.

```
In [97]: # Cross-validation code
         cv_scores = cross_val_score(lin_reg, X, y, cv=5, scoring='neg_mean_squared_error')
         cv_mse = -cv_scores.mean()
         print("Cross-Validation MSE:", cv_mse)

         Cross-Validation MSE: 0.09807889456819911
```

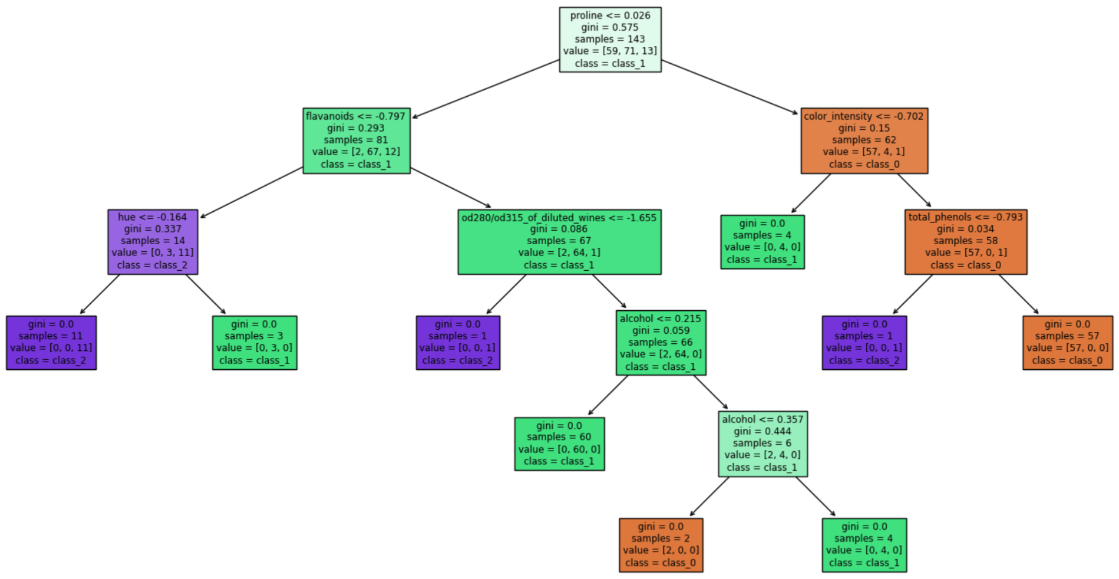


```
In [98]: # Predicting and evaluating code
         lin_reg_predictions = lin_reg.predict(X_test)
         mse = mean_squared_error(y_test, lin_reg_predictions)
         r2 = r2_score(y_test, lin_reg_predictions)
         print("Test MSE:", mse)
         print("Test R2 Score:", r2)

         Test MSE: 0.07384469727182866
         Test R2 Score: 0.0
```

Decision Tree

A Decision Tree is a flowchart-like structure in which each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label. Trees can represent a series of decisions that lead to a classification or regression value.

In my algorithm implementationm, the DecisionTreeClassifier from scikit-learn is used. The tree is also trained with the fit method, and its performance is evaluated with cross-validation. Finally, the tree is visualized using the plot_tree function, providing an visual representation of the decision-making process.The visual representation is particularly useful for understanding how the model makes decisions based on the features of the dataset, with the goal of accurately predicting the quality of wine.



proline <= 0.026 gini = 0.575 samples = 143 value = [59, 71, 13] class = class_1
flavanoids <= -0.797 gini = 0.293 samples = 81 value = [2, 67, 12] class = class_1
color_intensity <= -0.702 gini = 0.15 samples = 62 value = [57, 4, 1] class = class_0
hue <= -0.164 gini = 0.337 samples = 14 value = [0, 3, 11] class = class_2
od280/od315_of_diluted_wines <= -1.655 gini = 0.086 samples = 67 value = [2, 64, 1] class = class_1
gini = 0.0 samples = 4 value = [0, 4, 0] class = class_1
total_phenols <= -0.793 gini = 0.034 samples = 58 value = [57, 0, 1] class = class_0
gini = 0.0 samples = 11 value = [0, 0, 11] class = class_2
gini = 0.0 samples = 3 value = [0, 3, 0] class = class_1
gini = 0.0 samples = 1 value = [0, 0, 1] class = class_2
alcohol <= 0.215 gini = 0.059 samples = 66 value = [2, 64, 0] class = class_1
gini = 0.0 samples = 1 value = [0, 0, 1] class = class_2
gini = 0.0 samples = 57 value = [57, 0, 0] class = class_0
gini = 0.0 samples = 60 value = [0, 60, 0] class = class_1
alcohol <= 0.357 gini = 0.444 samples = 6 value = [2, 4, 0] class = class_1
gini = 0.0 samples = 2 value = [2, 0, 0] class = class_0
gini = 0.0 samples = 4 value = [0, 4, 0] class = class_1

Methodology

This research project was conducted with a structured approach to explore the wine dataset and apply 3 machine learning algorithms to predict wine quality and compare their performance of the dataset. The dataset was first standardized using the StandardScaler from the sklearn.preprocessing module to ensure the features had a mean of zero and a standard deviation of one. This step is crucial for algorithms like kNN, which rely on distance calculations, as it ensures all features contribute equally to the result.

Three algorithms were chosen for their varied approaches to learning from data:

- k-Nearest Neighbors which is a non-parametric, instance-based learning algorithm that classifies new cases based on the majority vote of the k-nearest neighbors.
- Linear Regression which is a parametric approach to predict a continuous dependent variable based on the linear relationship with one or more independent variables.
- Decision Tree which is a non-parametric model used for classification and regression tasks that splits the data into subsets based on feature value conditions, forming a tree-like structure.

Each algorithm was implemented and tested on the wine dataset. For kNN, the model was built from scratch using Python, employing the Euclidean distance to identify the nearest neighbors. In our custom k-Nearest Neighbors (kNN) implementation, the CustomKNN class features two primary methods: train and classify. The train method stores the training data, which is crucial for the classification process. When new data instances are provided for classification, the classify method is invoked. This method utilizes the training data stored earlier to classify each new instance based on its proximity to the nearest training examples.

I implemented manual cross-validation as a key evaluation technique for the CustomKNN model. Cross-validation is a robust method for assessing the generalization capability of a machine learning model. It involves partitioning the dataset into multiple subsets, or "folds," and iteratively training and testing the model on these folds. This approach ensures that each data point is used for both training and testing, providing a comprehensive evaluation of the model's performance.

The decision to implement manual cross-validation was because the kNN algorithm was implemented from scratch, using manual cross-validation allowed for deeper insights into the functioning and performance of the model. It provided an opportunity to directly interact with the data splitting and validation process, offering a better understanding of how kNN behaves with different subsets of data. Also, Implementing cross-validation manually reinforced the understanding of this evaluation technique. It helped in comprehending the importance of using different data subsets for training and validation to avoid overfitting and to ensure that the model's performance is reliable and not just a result of specific data splits.

For Linear Regression and Decision Tree, the scikit-learn library was utilized. These models were trained on a portion of the dataset and validated using cross-validation to assess their generalization capabilities. Cross-validation was particularly necessary to mitigate the risk of overfitting and to ensure that the model's performance was not just a result of the specific way the data was split.

The models were evaluated using accuracy (for classification tasks), mean squared error (MSE), and the R² score. Accuracy measures the proportion of correct predictions over the total number of instances evaluated. MSE quantifies the difference between the predicted and actual values, with a lower score indicating a better fit. The R² score represents the proportion of variance in the dependent variable that is predictable from the independent variables, with a higher score indicating a better fit.

Lastly, the Decision Tree model was visualized to gain insights into the decision-making process and understand the feature importance and decision paths within the model.

This systematic application of three distinct algorithms provides a comprehensive understanding of the dataset's characteristics and the prediction models' strengths and weaknesses.

Results

The results of applying the k-Nearest Neighbors, Linear Regression, and Decision Tree algorithms on the wine dataset are summarized in the following table. These results are derived from the experiments detailed in the Methodology section, where each algorithm was subjected to both training/testing splits and cross-validation to assess their performance.

Algorithm	Cross-Validation Score	Test Accuracy	Test MSE	Test R^2 score
kNN	0.9159 (Manual CV)	0.9714	-	-
Linear Regression	0.0981 (MSE CV)	-	0.0738	0.0
Decision Tree	0.8739	0.8857	0.2	0.0

Interpretation of Results:

k-Nearest Neighbors

Demonstrated the highest test accuracy (0.9714), indicating its effectiveness in classifying wine quality correctly. The manual cross-validation accuracy of 0.9159 further corroborates its reliability.

Linear Regression
Exhibited the lowest test MSE (0.0738), suggesting its precision in predicting continuous values. However, the R^2 score of 0.0 indicates a lack of fit to the variance in the dataset.

Decision Tree
Showed a balanced performance with a test accuracy of 0.8857 and a test MSE of 0.2. Its versatility is evident, though it lags slightly behind kNN in terms of accuracy.

The results suggest that for classification tasks in the wine dataset, kNN is the most effective. Linear Regression, while precise in continuous predictions, lacks explanatory power. The Decision Tree offers a middle ground in performance, being neither the most accurate nor the most precise.

Evaluation

In evaluating the effectiveness of the k-Nearest Neighbors, Linear Regression, and Decision Tree algorithms on the wine dataset, several insights and limitations emerged, highlighting both the strengths and areas for improvement in this project.

Strengths:

Algorithm Testing
The project successfully implemented and tested three distinct machine learning algorithms, each with different strengths and applications. This comprehensive approach provided a well-rounded understanding of how different methods perform on the same dataset.

Evaluation Techniques
Utilizing cross-validation and various metrics like accuracy, MSE, and R^2 Score ensured a thorough assessment of each model's performance. These methods helped mitigate overfitting and provided a more generalizable understanding of each algorithm's effectiveness.

Practical Insights
The project yielded practical insights into the wine dataset, particularly highlighting kNN's suitability for classification tasks and Linear Regression's precision in continuous value prediction.

Weaknesses:

Limited Scope
While three algorithms were tested, the scope remained limited to basic implementations. Exploring more complex methods could potentially reveal more about the dataset

Algorithm Limitations
Each algorithm had its limitations e.g kNN's performance might degrade with a larger dataset due to its computational intensity, Linear Regression's low R^2 Score indicates a poor fit, which could be improved with more complex models, the Decision Tree could benefit from hyperparameter tuning to avoid overfitting and improve accuracy.

Conclusions

In this exploration of machine learning algorithms on the wine dataset, several key insights have emerged:

k-Nearest Neighbors (kNN) for Wine Type Classification
When it comes to categorizing wines, perhaps by quality or type, kNN emerged as the star. Its impressive accuracy rates show it's adept at sorting wines into the correct categories in our dataset.

Linear Regression for Precise Predictions
Linear Regression, though not the top performer in variance explanation, excelled in making precise continuous predictions, like estimating the alcohol content or acidity levels in the wines. Its lower Mean Squared Error (MSE) indicates it's a strong choice for tasks needing exact numerical outcomes.
Decision Tree

A Versatile Contender
The Decision Tree showed a balanced act. It may not have beaten kNN in sorting wines or Linear Regression in precision, but its ability to handle both classification and regression makes it a versatile choice for various wine dataset tasks.

In summary, If it's about classifying wines, kNN is your go-to. For numerical predictions, like forecasting wine quality scores, turn to Linear Regression. And for a bit of both? The Decision Tree is your all-rounder. This exploration highlights the pivotal role of selecting the right machine learning tool based on the task and the specific nature of the data at hand.

References

1. CholletF. (2018). *Deep learning with Python*. Manning Publications.
2. Harrison, O. (2018, September 10). *Machine Learning Basics with the K-Nearest Neighbors Algorithm*. Medium; Towards Data Science. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761
3. GeeksForGeeks. (2017, October 16). *Decision Tree - GeeksforGeeks*. GeeksforGeeks. https://www.geeksforgeeks.org/decision-tree/
4. Mali, K. (2021, October 4). *Linear Regression | Everything you need to Know about Linear Regression*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/10/everything-you-need-to-know-about-linear-regression/