Assignment Report

Rain Prediction

AbubakerElsiddig Omer Ali
20210733

Requirements

1. **Task 1**: Preprocessing
    a. Does the dataset contain any missing data? **Identify them**.
    b. **Apply** the two techniques to handle missing data, dropping missing values and replacing them with the average of the feature.
    c. Does our data have the same scale? If not, you should **apply** feature scaling on them.
    d. **Splitting** our data to training and testing for training and evaluating our models
2. **Task 2**: Implement Decision Tree, k-Nearest Neighbors (kNN) and naïve Bayes
    a. Using scikit-learn **implement** Decision Tree, kNN and Naïve Bayes
    b. **Compare** the performance of your implementations by evaluating accuracy, precision, and recall metrics.
    c. **Implement** k-Nearest Neighbors (kNN) algorithm from scratch.
    d. **Report** the results and compare the performance of your custom k-Nearest Neighbors (kNN) implementation with the pre-built kNN algorithms in scikit-learn, using the evaluation metrics mentioned in point 2. Using any missing handling techniques, you chose from task 1.2.
3. **Task 3**: Interpreting the Decision Tree and Evaluation Metrics Report
    a. The effect of different data handling
        i. **Provide** a detailed report evaluating the performance of scikit-learn implementations of the Decision Tree, k-Nearest Neighbors (kNN) and naïve Bayes with respect to the different handling missing data technique.
    b. 2. Decision Tree Explanation Report
        i. **Create** a well-formatted report that includes a plot of the decision tree and a detailed explanation of how the tree makes predictions.
        ii. **Discuss** the criteria and splitting logic used at each node of the tree.
    c. Performance Metrics Report
        i. **Provide** a detailed report evaluating the performance of your implementations of the k-Nearest Neighbors (kNN) from scratch with different k values at least 5 values.
        ii. **Include** the accuracy, precision, and recall metrics for models.

NOTE: **I didn't join a team in this assignment due to a negative experience with my previous teammates, which resulted in a lower grade and a delayed submission.**

- ## Preprocessing

That data have some missing values “as shown in the figure below”

```
Missing Data for each feature:
Temperature    25
Humidity       40
Wind_Speed     32
Cloud_Cover    33
Pressure       27
Rain            0
dtype: int64
```

I replaced all missing values with the average “**mean**” and saved it into a new file

```
Preprocessed data has been saved to 'cleaned_weather_forecast_data.csv'.

Missing Data for each feature (after preprocessing):
Temperature    0
Humidity       0
Wind_Speed     0
Cloud_Cover    0
Pressure       0
Rain           0
```

And now let’s check if the data are on the same scale

```
Range of each numeric feature:
Temperature: 24.99337196709648
Humidity: 69.99240982328772
Wind_Speed: 19.98931297915363
Cloud_Cover: 99.98275706998673
Pressure: 69.9711069713037
```

As we can see, the features have **varying ranges,** therefore we need to perform **Feature Scaling**

Since Decision Tree and Naïve Bayes don't need feature Scaling, I will use **Standardization** for feature scaling.

I divided the data into :

- 20% testing
- 80% training

```
Training set size: 2000 samples
Testing set size: 500 samples
```

And here is the feature scaling results

	Temperature (1)	Humidity (2)	Wind_Speed (3)	Cloud_Cover (4)	Pressure (5)	Rain (6)
1	Temperature	Humidity	Wind_Speed	Cloud_Cover	Pressure	Rain
2	-0.4767723287667595	0.36774601235582244	0.8481604109290878	-0.038439328634538084	-1.3181859940620544	no rain
3	0.6222349073151711	1.0003752692979941	0.5882810861938995	-1.366169874222252	1.0474676817824762	no rain
4	-0.2934465500469677	-1.1144471124350204	-0.4694062817772389	-1.4944382628030262	0.957638021186079	no rain
5	-0.4108923600719755	-1.195779544641577	-0.9304418215991295	0.18077844896366468	1.2169551642895897	no rain
6	-0.37642628194969135	1.463262775961539	-1.7078735761103445	-0.6624585898244085	-0.24842443192883945	no rain
7	0.2973633173915426	0.7197786843296387	-1.0871513408457945	-1.1495654216694817	1.5215952814894191	no rain
8	1.2976673284516593	0.8679343835047083	-1.3775632247864185	-1.2216310387121148	-0.4003697833387268	no rain
9	1.1117172777026885	0.01610972162923415	0.19267583857269988	-1.4835374563906556	-0.7652718158686672	no rain

- ## Implement Decision Tree, k-Nearest Neighbors (kNN) and naïve Bayes

Using scikit-learn implement Decision Tree, kNN and Naïve Bayes.

Here are the results using Decision tree KNN and Naive Bayes from scikit-learn:

```
----------------------------------------
Decision Tree Results
Accuracy: 0.99

k-Nearest Neighbors Results
Accuracy: 0.96

Naïve Bayes Results
Accuracy: 0.95

----------------------------------------
```

Now let's Compare the performance of the implementations by evaluating accuracy, precision, and recall metrics.

```
Decision Tree Results
Accuracy: 0.99
              precision    recall  f1-score   support

     no rain       0.99      1.00      0.99       437
        rain       0.97      0.94      0.95        63
```

```
k-Nearest Neighbors Results
Accuracy: 0.96
              precision    recall  f1-score   support

     no rain       0.97      0.98      0.98       437
        rain       0.86      0.81      0.84        63

    accuracy                           0.96       500
```

```
Naïve Bayes Results
Accuracy: 0.95
              precision    recall  f1-score   support

     no rain       0.95      1.00      0.97       437
        rain       0.95      0.65      0.77        63

    accuracy                           0.95       500
```

Decision Tree: the most accurate, and best model to use, “but it may overfit”.

KNN: has fewer risks of overfitting.

Naïve Bayes: has the lowest recall, so it might struggle to correctly identify all positive instances of "rain."

- **Implement k-Nearest Neighbors (kNN) algorithm from scratch.**

I took the code provided in the lab and implemented the missing function “which was predict()”

```python
class KNN:
    new *
    def __init__(self, k=3):
        self.k = k

    1 usage  new *
    def fit(self, X, y):
        self.X_train = np.array(X)
        self.y_train = np.array(y)

    2 usages  new *
    def predict(self, X):
        X = np.array(X)
        predictions = []

        for x in X:
            # Compute distances between x and all examples in the training set
            distances = np.sqrt(np.sum((self.X_train - x) ** 2, axis=1))

            # Sort by distance and return indices of the first k neighbors
            k_indices = np.argsort(distances)[:self.k]

            # Extract the labels of the k nearest neighbor training samples
            k_nearest_labels = [self.y_train[i] for i in k_indices]

            # Find the most common class label
            label_counts = {}
            for label in k_nearest_labels:
                if label in label_counts:
                    label_counts[label] += 1
```

- **Report the results and compare the performance of your custom k-Nearest Neighbors (kNN) implementation with the pre-built kNN algorithms in scikit-learn, using the evaluation metrics mentioned in point 2.**

My KNN Results :

```
----------------------------------------
KNN Predictions:
Accuracy of the implemented is 0.962
Precision of the implementation for 'no rain' is 0.97
Precision of the implementation for 'rain' is 0.90
Recall of the implementation for 'no rain' is 0.98
Recall of the implementation for 'rain' is 0.83

----------------------------------------
```

Pre-built KNN scikit-learn Results :

```
k-Nearest Neighbors Results
Accuracy: 0.96
             precision    recall  f1-score   support

     no rain       0.97      0.99      0.98       431
        rain       0.92      0.81      0.86        69
```

As we can see in the pictures Both implementations perform very similarly.

Metric	Custom kNN	Scikit-learn kNN
Accuracy	**96.6%**	96%
Precision (No Rain)	97%	97%
Precision (Rain)	91%	**92%**
Recall (No Rain)	99%	99%
Recall (Rain)	**84%**	81%

- **Interpreting the Decision Tree and Evaluation Metrics Report**
- **The effect of different data handling**

Provide a detailed report evaluating the performance of scikit-learn implementations of the Decision Tree, k-Nearest Neighbors (kNN) and naïve Bayes with respect to the different handling missing data technique.

The Mean technique was provided in the report earlier, so now let's see the dropping missing values technique.

- **Dropping Missing Values** technique
- Decision Tree

```
Decision Tree Results
Accuracy: 0.82
              precision    recall  f1-score   support

     no rain       0.90      0.90      0.90       448
        rain       0.15      0.15      0.15        52
```

- KNN

```
k-Nearest Neighbors Results
Accuracy: 0.97
              precision    recall  f1-score   support

     no rain       0.98      0.99      0.99       448
        rain       0.91      0.83      0.87        52
```

- Naïve Bayes

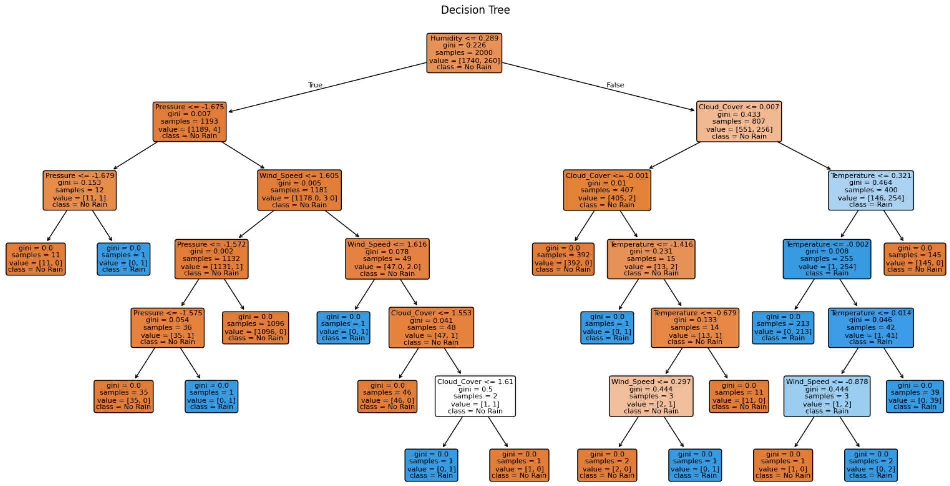
```
Naïve Bayes Results
Accuracy: 0.96
              precision    recall  f1-score   support

     no rain       0.96      1.00      0.98       448
        rain       0.97      0.63      0.77        52
```

And this table to make it simpler for Accuracy comparison :

	Decision Tree	KNN	Naïve Bayes
Replacing with average	99%	96%	96%
Dropping missing values	82%	97%	96%

- **Create a well-formatted report that includes a plot of the decision tree and a detailed explanation of how the tree makes predictions.**

First of all here is the Decision tree



- **criteria and splitting logic used at each node of the tree.**

1. Root Node: Humidity ≤ 0.289

- **Samples**: 2,000
- **Class Distribution**: [1,740 No Rain, 260 Rain]
- **Splitting Logic**:
  - If `Humidity ≤ 0.289`, the left branch is chosen.

---

2. Left Branch (True): Pressure ≤ -1.675

- **Samples**: 1,193
- **Class Distribution**: [1,189 No Rain, 4 Rain]
- **Splitting Logic**:
  - If `Pressure ≤ -1.675`, proceed left.

---

3. Right Branch (False): Cloud Cover ≤ -0.001

- **Samples**: 807
- **Class Distribution**: [551 No Rain, 256 Rain]
- **Splitting Logic**:
  - If `Cloud Cover ≤ -0.001`, follow the left branch.

There are so nodes so I just explain these ones and the others follows the same logic

- **Provide a detailed report evaluating the performance of your implementations of the k-Nearest Neighbors (kNN) from scratch with different k values at least 5 values.**

Here are the 5 values I will test KNN with:

- 5 , 100 , 500 , 1000 , 5000

- 5 :

```
----------------------------------------
KNN Predictions:
Accuracy of the implemented is 0.954
Precision of the implementation for 'no rain' is 0.97
Precision of the implementation for 'rain' is 0.82
Recall of the implementation for 'no rain' is 0.98
Recall of the implementation for 'rain' is 0.78
----------------------------------------
```

- 100:

```
----------------------------------------
KNN Predictions:
Accuracy of the implemented is 0.966
Precision of the implementation for 'no rain' is 0.96
Precision of the implementation for 'rain' is 1.00
Recall of the implementation for 'no rain' is 1.00
Recall of the implementation for 'rain' is 0.70
----------------------------------------
```

- 500:

```
----------------------------------------
KNN Predictions:
Accuracy of the implemented is 0.88
Precision of the implementation for 'no rain' is 0.88
Precision of the implementation for 'rain' is 0.00
Recall of the implementation for 'no rain' is 1.00
Recall of the implementation for 'rain' is 0.00
----------------------------------------
```

- 1000:

```
----------------------------------------
KNN Predictions:
Accuracy of the implemented is 0.864
Precision of the implementation for 'no rain' is 0.86
Precision of the implementation for 'rain' is 0.00
Recall of the implementation for 'no rain' is 1.00
Recall of the implementation for 'rain' is 0.00

----------------------------------------
```

- 5000:

```
----------------------------------------
KNN Predictions:
Accuracy of the implemented is 0.868
Precision of the implementation for 'no rain' is 0.87
Precision of the implementation for 'rain' is 0.00
Recall of the implementation for 'no rain' is 1.00
Recall of the implementation for 'rain' is 0.00

----------------------------------------
```

As we can observe, higher K values causes overfitting, the table summarizes the relationship between high n value and accuracy

KNN “k” value	Accuracy
5	95%
100	96%
500	88%
1000	86%
5000	86%