Your Report Title

Your Name

October 26, 2024

1 Introduction

This is the introduction section of the report. It provides background information and context for the study, states the research question or hypothesis, and briefly outlines the approach taken. A report structure overview is also provided to guide the reader through the document.

1.1 Background Information

Classification in the context of machine learning is the process of predicting the class or category of a given data point based on its features. For example: given a set of emails, classify each email as spam or not spam. It is a supervised learning technique that is used to assign labels to data points based on their characteristics. Classification is an ideal technique for the required analysis as outlined in this report.

K-Nearest Neighbour (k-NN) is a supervised machine learning algorithm that has been chosen as the primary method for this study. It is a simple yet powerful algorithm that classifies new cases based on similarity to existing data points. While k-NN can be used for both classification and regression problems, our focus is on classification due to the nature of the datasets used.

This report details the analysis of two datasets: 'hepatitis' and 'mushroom'. These are two of many datasets provided to us as part of this assignment. We decided upon these two datasets due to the stark contrast between them in terms of complexity and size.

1.2 Research Question or Hypothesis

The primary research question addressed in this report is: 'Can the k-Nearest Neighbour algorithm effectively classify data points in the hepatitis and mushroom datasets with high accuracy'?.

The specific objectives of this study are:

1. To evaluate k-NN's classification performance on datasets of varying sizes and complexities
2. To determine optimal k-NN hyperparameter settings for each dataset
3. To assess the algorithm's robustness and limitations in different scenarios

We hypothesize that k-NN will perform excellently on the mushroom dataset due to the simplicity of the data, but may struggle with the hepatitis dataset due to the complexity of the data in that dataset.

1.3 Approach and Methodology

The approach taken in this report involves the following steps:

1. Data Preprocessing: Both datasets are preprocessed, ensuring the data is clean and ready for analysis.
2. Model Training: Different combinations of hyperparameters are used to train the k-NN algorithm on the training dataset.
3. Model Evaluation: The performance of the best k-NN algorithm from the previous step is evaluated using the test dataset.

1.4 Report Structure

The report is structured as follows:

1. Introduction (Section 1): Provides background information, states the research question, and outlines the approach.
2. Data (Section 2): Describes the data used in the study, along with the source, characteristics, relevance, and the preprocessing techniques that were applied.
3. Methods (Section 3): Describes the methodology used in the study, including the algorithms, techniques, and tools used.
4. Results and Analysis (Section 4): Findings of the study are presented, including relevant data, statistics, and figures to help visualise the data.
5. Conclusion (Section 5): Summarizes main findings of the study and their significance.

2 Data

2.1 Dataset

This section describes the dataset used in the study, including its source, characteristics, and relevance to the research question.

2.1.1 Dataset Overview

2.1.2 Data Collection

2.1.3 Data Characteristics

2.2 Data Preprocessing

This section outlines the steps taken to prepare the dataset for analysis, including cleaning, transformation, and feature engineering.

2.2.1 Data Cleaning

2.2.2 Data Transformation

2.2.3 Feature Engineering

3 Methods

This section describes the methodology used in the study. It should include details about data collection, experimental design, and analytical techniques.

3.1 k-Nearest Neighbors (kNN)

This section describes the k-Nearest Neighbors (kNN) algorithm and its implementation in our study.

3.1.1 Algorithm Overview

The k-NN algorithm operates on a simple yet effective principle: when classifying new data points, it examines the k closest training examples and assigns the most common class among these neighbors (where k is a user-defined hyperparameter). The algorithm's effectiveness relies on two fundamental assumptions:

- Locality: Points that are close to each other are likely to have the same class.
- Smoothness: The classification boundary between classes is relatively smooth.

One key feature of k-NN is it employs neighbor-based classification, where the classification of a new data point is determined by majority voting among its k-nearest neighbors. The value of k is one of the most important tunable hyperparameters, as it significantly influences the algorithm's behaviour:

- Small k values (e.g., k=1 or k=3): More sensitive to local patterns but susceptible to noise.
- Large k values: More robust to noise but may overlook important local patterns.
- Even vs. Odd k values: Even k values result in ties, which may require additional rules to break.

The dependent variable in our datasets we aim to predict is categorical, so while k-NN can be used for both classification and regression problems our focus is on classification.

While the k-NN algorithm has it's merits in terms of simplicity and interpretability, it also has several disadvantages:

- Computationally expensive: As the number of training examples grows, the algorithm's complexity increases[1].
- Sensitive to irrelevant features: The algorithm treats all features equally, so irrelevant features can negatively impact performance.
- Curse of dimensionality: As the number of features increases, the algorithm requires more data to maintain performance.

All three of the above mentioned disadvantages all relate to the features of the dataset, and how they can impact the performance of the algorithm. They create a compounding effect: more features lead to higher computational cost, while making the algorithm more susceptible to noise and irrelevant features, and also requiring more data to maintain performance. This is why the use of feature selection and reduction techniques are imperative when working with the k-NN algorithm to increase performance.

3.1.2 Implementation Details

The K-Nearest Neighbors (k-NN) algorithm is implemented using Python with various libraries and tools. Below are the specific implementation details:

Distance calculations

The Euclidean distance is used to measure the distance between data points. This distance metric is the most commonly used distance metric in k-NN algorithms due to its simplicity and effectiveness in measuring the similarity between data points[2]. It operates on the principle of calculating the straight-line distance between two points in a Euclidean space, hence it's simplicity.

The formula for Euclidean distance between two vectors x and y is:

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

where x_i and y_i are the ith elements of vectors x and y, respectively.

The Manhattan distance is another distance metric that can be used in k-NN algorithms. It is also frequently used with the k-NN algorithm, albeit not as common as the Euclidean distance. The Manhattan distance is calculated by summing the absolute differences between the coordinates of two points.

The formula for Manhattan distance between two vectors x and y is:

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

where x_i and y_i are the ith elements of vectors x and y, respectively.

The Chebychev distance less commonly used than the Euclidean and Manhattan distances in relation to the k-NN algorithm, but it is still a valid distance metric and operates effectively in measuring the similarity between data points. The Chebychev distance is calculated by taking the maximum absolute difference between the coordinates of two points. This differs from Manhattan distance in that it takes the maximum absolute difference, rather than the sum.

The formula for Chebychev distance between two vectors x and y is:

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

where x_i and y_i are the ith elements of vectors x and y, respectively.

Along with the Chebychev distance, the Malahanobis distance is another distance metric used in k-NN algorithms. It is also less commonly used than the Euclidean and Manhattan distance metrics. The Mahalanobis distance is calculated by taking the square root of the sum of the squared differences between the coordinates of two points, where the squared differences are divided by the covariance matrix of the data.

The formula for Mahalanobis distance between two vectors x and y is:

$$d = \sqrt{(x-y)^T \cdot S^{-1} \cdot (x-y)}$$

where x and y are the vectors, and S is the covariance matrix of the data.

Weighting Schemes

In the k-NN algorithm, the choice of weighting scheme can significantly impact the classification results. The following weighting schemes were implemented in this study:

In uniform weighting, all neighbours have equal weight in the voting process. This is the default weighting scheme in k-NN. An advantage of uniform weighting is that it is simple and computationally efficient, but it may not be optimal for imbalanced datasets.

With ReliefF weighting, neighbours are weighted based on their relevance to the target class. This provides a more nuanced approach to weighting as it considers the importance of each neighbour in the classification process, potentially improving performance of the algorithm.

Information gain weighting, Neighbours are weighted based on gain in information in the context of the target variable [3]. Similarly to ReliefF weighting, this weighting scheme assigns higher weights to neighbours that provide more information about the target class.

Voting Schemes

In the k-NN algorithm, the voting scheme determines how the class label of a new data point is determined based on the class labels of its k-nearest neighbors. The following voting schemes were implemented in this study:

In the majority voting scheme, the class label with the highest frequency among the k-nearest neighbors is assigned to the new data point. As well as this, each vote is given equal weight in the voting process [4]. This is the default voting scheme in k-NN and is simple and easy to implement.

With inverse distance weighting voting, the class labels of the k-nearest neighbors are weighted based on their distance from the new data point. While there are different methods to perform this weighting, the most simple version is to take a neighbour's vote to be the inverse of its distance to q:

$$w_i = \frac{1}{d(q, x_i)}$$

where w_i is the weight of the ith neighbour, $d(q, x_i)$ is the distance between the new data point q and the ith neighbour x_i.

Then the votes are summed and the class with the highest votes is returned [4].

Shepard's method is another voting scheme that can be used in k-NN algorithms. It employs the use of an exponential function to weight the votes of the neighbours based on their distance from the new data point, rather than the inverse of the distance [5].

The formula for Shepard's voting scheme is:

$$Vote(y_j) = \sum_{i=1}^{k} e^{-d(\mathbf{q},\mathbf{x}_i)^2} 1(y_j, y_c) \tag{1}$$

where $Vote(y_j)$ is the vote for class y_j, $d(\mathbf{q}, \mathbf{x}_i)$ is the distance between the new data point $\mathbf{q}$ and the ith neighbour $\mathbf{x}_i$, and $1(y_j, y_c)$ is the indicator function that returns 1 if y_j is the same as the class label y_c of the ith neighbour, and 0 otherwise.

Neighbor Selection

The choice of the number of neighbors (k) is a critical hyperparameter in the k-NN algorithm, as previously discussed in this report. In this study, the following values of k where examined:

[1, 3, 5, 7]

The following libraries and tools were used for the implementation of our study: - Python: The primary programming language used for the implementation of the k-NN algorithm. - NumPy: A useful package for scientific computing with Python, used for numerical operations. - Scikit-learn: A machine learning library in Python, used for implementing the k-NN algorithm. - Pandas: A data manipulation library in Python, used for data preprocessing and analysis. - Matplotlib: A plotting library in Python, used for data visualization. - CHECK FOR MORE

3.1.3 Parameter Tuning

3.2 Dimensionality Reduction Algorithms

This section outlines the dimensionality reduction techniques applied in our study.

3.3 Support Vector Machines (SVM)

This section describes the methodology used in the study. It should include details about data collection, experimental design, and analytical techniques.

This section describes the Support Vector Machines (SVM) algorithm and its implementation in our study.

3.3.1 Algorithm Overview

3.3.2 Kernel Selection

3.3.3 Implementation Details

4 Results and Analysis

4.1 Results

Here, present the findings of the study. Include relevant data, statistics, and any figures or tables that help illustrate the results.

Table 1: Results from KNN models for the mushroom dataset

	k	distance func	voting func	weighting func	accuracy	f1
1	7	ManhattanDistance	ShepardsWorkVote	EqualWeighting	0.9510	0.9517
2	5	ManhattanDistance	ShepardsWorkVote	EqualWeighting	0.9510	0.9517
3	3	ManhattanDistance	ShepardsWorkVote	EqualWeighting	0.9510	0.9517
4	1	ManhattanDistance	MajorityClassVote	EqualWeighting	0.9500	0.9509
5	1	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting	0.9500	0.9509
6	1	ManhattanDistance	ShepardsWorkVote	EqualWeighting	0.9500	0.9509
7	5	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting	0.9330	0.9328
8	7	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting	0.9300	0.9291
9	3	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting	0.9280	0.9289
10	3	ManhattanDistance	MajorityClassVote	EqualWeighting	0.9260	0.9267

Table 2: Results from KNN models for the hepatitis dataset

	k	distance func	voting func	weighting func	accuracy	f1
1	1	EuclideanDistance	ShepardsWorkVote	ReliefFWeighting	0.9548	0.9717
2	1	EuclideanDistance	InverseDistanceWeightedVote	ReliefFWeighting	0.9548	0.9717
3	1	EuclideanDistance	MajorityClassVote	ReliefFWeighting	0.9548	0.9717
4	1	ChebyshevDistance	ShepardsWorkVote	EqualWeighting	0.9484	0.9685
5	1	ChebyshevDistance	InverseDistanceWeightedVote	EqualWeighting	0.9484	0.9685
6	1	ChebyshevDistance	MajorityClassVote	EqualWeighting	0.9484	0.9685
7	7	EuclideanDistance	ShepardsWorkVote	EqualWeighting	0.9484	0.9677
8	5	EuclideanDistance	ShepardsWorkVote	EqualWeighting	0.9484	0.9677
9	1	ManhattanDistance	MajorityClassVote	EqualWeighting	0.9484	0.9675
10	7	ManhattanDistance	ShepardsWorkVote	EqualWeighting	0.9484	0.9675

Table 3: Results from SVM models for the mushroom dataset

	C	kernel type	accuracy	f1
1	1	poly	0.9280	0.9276
2	7	rbf	0.9260	0.9261
3	5	rbf	0.9250	0.9246
4	3	rbf	0.9240	0.9235
5	7	poly	0.9200	0.9200
6	5	poly	0.9160	0.9158
7	3	poly	0.9160	0.9155
8	3	linear	0.9140	0.9124
9	1	linear	0.9110	0.9098
10	1	rbf	0.9060	0.9019

Table 4: Results from SVM models for the hepatitis dataset

	C	kernel type	accuracy	f1
1	7	rbf	0.9548	0.9719
2	3	rbf	0.9484	0.9680
3	5	rbf	0.9484	0.9680
4	1	poly	0.9484	0.9677
5	3	poly	0.9484	0.9675
6	5	poly	0.9484	0.9672
7	7	poly	0.9484	0.9672
8	1	rbf	0.9032	0.9412
9	3	linear	0.8903	0.9295
10	1	linear	0.8839	0.9274

4.2 Discussion

In this section, interpret the results, discuss their implications, and relate them back to the research question or hypothesis. Address any limitations of the study and suggest areas for future research.

4.2.1 k-Nearest Neighbors (kNN) Analysis

This section interprets the results of the kNN algorithm, discussing its performance and implications.

4.2.2 Dimensionality Reduction Analysis

This section interprets the results of the dimensionality reduction techniques, discussing their impact on the analysis and visualization.

4.2.3 Support Vector Machines (SVM) Analysis

This section interprets the results of the SVM algorithm, discussing its performance and implications.

5 Conclusion

Summarize the main findings of the study and their significance. Restate the key points and provide a final perspective on the research.