Analysis the performance of Naive Bayes and K-Nearest Neighbor Classifiers

Hubert Bojda[1], Dawid Gala[2]

[1]Faculty of Applied Mathematics, Silesian University of Technology, Kaszubska 23, 44100 Gliwice, POLAND
[2]Faculty of Applied Mathematics, Silesian University of Technology, Kaszubska 23, 44100 Gliwice, POLAND

Abstract

In our study, we implemented and compared two machine learning algorithms: K-Nearest Neighbors (KNN) and Naive Bayes. For each algorithm, we conducted 10 test runs to evaluate their performance. The results indicated that the KNN algorithm achieved an accuracy ranging from 0.80 to 0.82, demonstrating its robustness in predicting weather conditions based on the London's historical weather data. On the other hand, the Naive Bayes algorithm achieved an accuracy ranging from 0.74 to 0.76. Although slightly lower than KNN, these results still reflect the Naive Bayes algorithm's effectiveness in handling the weather data. Overall, this analysis provides valuable insights into the predictive capabilities of these algorithms.

Keywords

artificial intelligence, London weather data, dataset, machine learning algorithms, K-Nearest Neighbors (KNN), Naive Bayes, accuracy, F1-score

1. Introduction

Artificial intelligence methods show us examples and uses of machine learning algorithms. This is very important in today's world, because more and more systems have more or less developed ai algorithms implemented. For example, they can be used for deep neural network models for unbalanced medical data of IoT systems[1] or to predict COVID-19 virus spread [2] This artificial intelligence system was developed to explore and validate the effectiveness of the K-Nearest Neighbors (KNN) and Gaussian Naive Bayes algorithms. To achieve this, we selected a weather database, which is particularly well-suited for testing these algorithms due to its mix of numerical and categorical data. The database includes columns with numerical values such as temperature, humidity, and wind speed, alongside a column containing categorical information about weather conditions at the time of observation, including categories like 'Clear', 'Overcast', and 'Foggy'. This rich and diverse dataset facilitates effective training and testing, enabling a thorough evaluation of the algorithms' performance.
The numerical data is good for the KNN algorithm, which predicts outcomes based on the distance between data points. For KNN, we use Euclidean distance measure to find the closest neighbors to a given data point and make predictions based on these neighbors. On the other hand, the categorical weather classifications are well-suited for the Naive Bayes algorithm. Naive Bayes works by calculating the probability of each class based on the feature distributions and assumes that features are independent given the class label, making it efficient for categorical data.

We then divided the dataset into a 70:30 ratio for training and testing. This split provides a substantial amount of data for training the models while reserving enough data to accurately assess their performance.

Our benchmark tests involved evaluating the algorithms using standard performance metrics such as accuracy, precision, recall, and F1-score. These metrics offer a comprehensive view of the algorithms' ability to classify weather conditions correctly. Additionally, we performed cross-validation to ensure that our results were not overly dependent on a particular train-test split, further validating the robustness and reliability of our models.

SSI, project 2023/2024
hb305153@student.polsl.pl (H. Bojda); dg308257@student.polsl.pl (D. Gala)
© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

2. Methodology

2.1. K Nearest Neighbors

2.1.1. Description

The KNN classifier[3], or k nearest neighbor algorithm, is used to classify and predict the value based on the variable specified in the decision column in the database. The algorithm compares the values in the columns that explain the phenomenon with the values of the variables that are included in the learning set. It contains information about the k closest observations from the learning set.
An important aspect in the creation of a classifier is the selection of an appropriate metric that calculates the distance between the observations of the learning set and the training set. The most popular metrics are Euclidean, Minkowski or Manhattan.
With successive iterations, the division of the data is corrected against the given metric. The algorithm moves data between classes so that the variance within each class is as smallest

2.1.2. Formulas

1. Calculating Distance Between Points:
The Euclidean distance d between two points $x_i = (x_{i1}, x_{i2}, \ldots, x_{in})$ and $x_j = (x_{j1}, x_{j2}, \ldots, x_{jn})$ is given by:

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{n} (x_{ik} - x_{jk})^2} \tag{1}$$

2. Finding Nearest Neighbors
To find the k nearest neighbors for a test point, compute the Euclidean distances from the test point to all points in the training set and select the k points with the smallest distances.
3. Classification by Majority Voting
For classification, the class of the test point is determined by the classes of its k nearest neighbors. The class C of the test point is given by:

$$C = \text{argmax}_c \sum_{i=1}^{k} \mathbf{1}(y_i = c) \tag{2}$$

where $\mathbf{1}(y_i = c)$ is an indicator function that equals 1 if $y_i = c$ and 0 otherwise.

2.1.3. Classifier Algorithm

The KNN classifier algorithms are shown below:

Algorithm 1 KNN Algorithm

Require: $X\_train, y\_train, X\_test, y\_test$
1: $predictions \leftarrow []$
2: **for** x **in** X_test **do**
3: Calculate and sort distances from x to X_train.
4: Select k nearest neighbors' labels.
5: Perform majority voting to determine the most frequent label.
6: Add the most frequent label to $predictions$.
7: Calculate the accuracy by comparing $predictions$ to y_test.
8: **end for**
9: **return** $predictions$

2.2. Naive Bayes

2.2.1. Description

Before describing Gaussian Naive Bayes, we would like to describe how the naive bayes algorithm works. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features. Based on prior knowledge of conditions that may be related to an event, Bayes theorem describes the probability of the event.
So what is Gaussian Naive Bayes?[4] Gaussian Naive Bayes is a type of Naive Bayes method where continuous attributes are considered and the data features follow a Gaussian distribution throughout the dataset. In Sklearn library terminology, Gaussian Naive Bayes is a type of classification algorithm working on continuous normally distributed features that is based on the Naive Bayes algorithm. Before diving deep into this topic we must gain a basic understanding of the principles on which Gaussian Naive Bayes work. Here are some terminologies that can help us gain knowledge and ease our further study. The Naive Bayes classifier is based on Bayes' theorem and the assumption of conditional independence of features. The formula is as follows:

2.2.2. Formulas

$$P(C_k|\mathbf{x}) = \frac{P(C_k) \cdot P(\mathbf{x}|C_k)}{P(\mathbf{x})} \quad (3)$$

Where:

- $P(C_k|\mathbf{x})$ is the posterior probability of class C_k given the sample $\mathbf{x}$,
- $P(C_k)$ is the prior probability of class C_k,
- $P(\mathbf{x}|C_k)$ is the likelihood of sample $\mathbf{x}$ given class C_k,
- $P(\mathbf{x})$ is the total probability of the sample $\mathbf{x}$.

Assuming conditional independence of features $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, we can write:

$$P(\mathbf{x}|C_k) = P(x_1, \ldots, x_n|C_k) = \prod_{i=1}^{n} P(x_i|C_k) \quad (4)$$

Therefore, the final formula for the Naive Bayes classifier is:

$$P(C_k|\mathbf{x}) \propto P(C_k) \cdot \prod_{i=1}^{n} P(x_i|C_k) \quad (5)$$

2.2.3. Classifier Algorithm

The Naive Bayes classifier algorithms are shown below:

Algorithm 2 Description of the Naive Bayes Algorithm

Require: X_train, y_train, X_test

$predictions \leftarrow []$
Calculate the prior probabilities for each class using y_train.
Calculate the mean and variance for each feature for each class using X_train and y_train.
for x **in** each point in X_test **do**
    Calculate the likelihood of x for each class using the Gaussian probability density function.
    Calculate posterior probabilities for each class based on the features of point x.
    Select the class with the highest posterior probability as the predicted label for point x.
    Add the predicted label to the predictions list.
end for
return $predictions$

3. Experiments

3.1. Dataset preparing

Weather Dataset[5] contains data from years 1979 to 2021., extracted by MUTHUKUMAR.J. Records, that did not meet following dependency have been removed from the original database:

- Formatted Date
- Apparent Temperature (C)
- Precip Type
- Loud Cover
- Daily Summary
- Humidity
- Wind Bearing (degrees)

In the first phase of testing, we worked on three abstract classes: 'rain,' 'clear,' and 'overcast.' The accuracy of the classifiers was around 90% for KNN and 80% for Naive Bayes. However, the confusion matrices revealed that the count of entities labeled 'rain' in the 'Summary' column was very low. Consequently, the next step was to identify the abstract class with the highest count of entities. To address this, we analyzed the distribution of the 'Summary' column values across the different classes. This analysis helped us determine which class had the highest representation, allowing us to focus our efforts on balancing the dataset and improving the overall performance of the classifiers.Based on this, records in which the abstraction class is not included were removed. These other classes had a negative impact[6] on the model's performance. For example: "Breezy and Mostly Cloudy", "Windy and Foggy", "Windy and Dry", "Dry and Partly Cloudy".

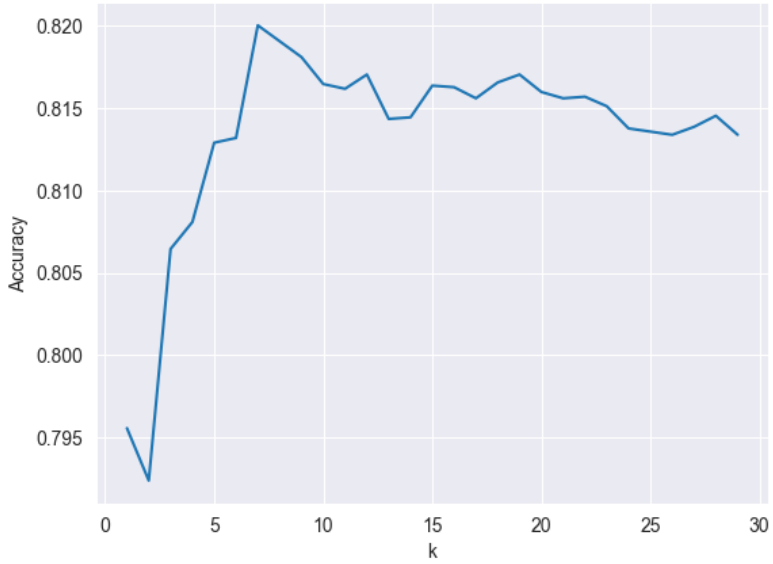

Figure 1: Choosing the best k value

3.2. Tests

3.2.1. KNN tests

The first phase of testing involved appropriately reducing the number of classes in the project's dataset to decrease the computational complexity of the model. After data preparation, model testing commenced. Next, the optimal value of k for the model was determined. The Matplotlib library[7], which generates graphs, was helpful in this regard. In Figure 1, we observe that our model performs best for $k = 6$. However, in the interval [1, 10], the values exhibit significant variability, with stabilization occurring only in the interval (10, 30).

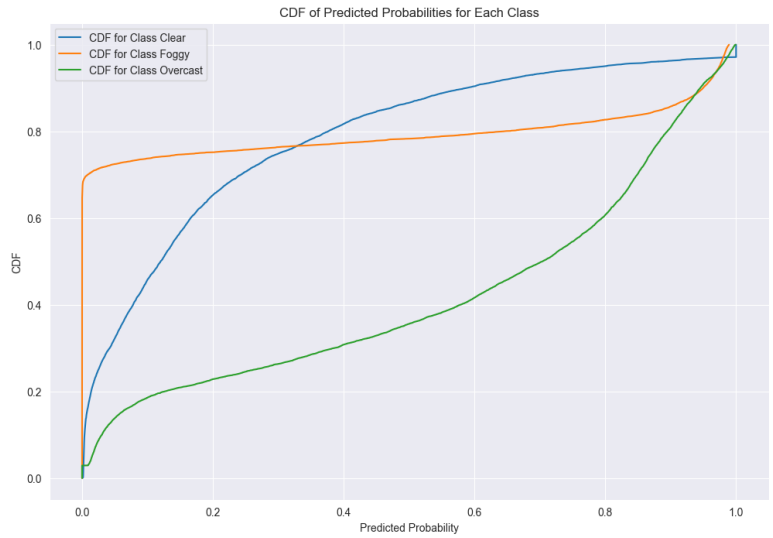

Figure 2: Cumulative Distribution Function for Naive Bayes Classifier

3.2.2. Naive Bayes tests

The Bayes classifier has a good distribution when:

- The lines for each class increase rapidly, indicating high probabilities assigned by the model to the correct classes.
- Lines for different classes should be separated from each other, indicating that the model distinguishes classes well.
- The CDF lines should be close to zero at low probabilities

As you can see from fig.2, all of these things are almost maintained, indicating that the classifier predicts quite well. We can confirm this because the classifier has an accuracy of about 75%. It is worth noting on the sudden intersection of the foggy class. The abrupt intersection of the line indicates that the model is uncertain about assigning probabilities to this particular class, which may be the result of an overlap in feature space between this class and other classes.

3.3. Analysis

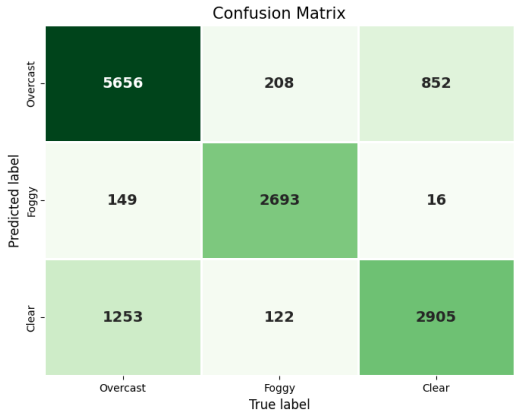

Figure 3: Confusion Matrix for K Nearest Neighbors

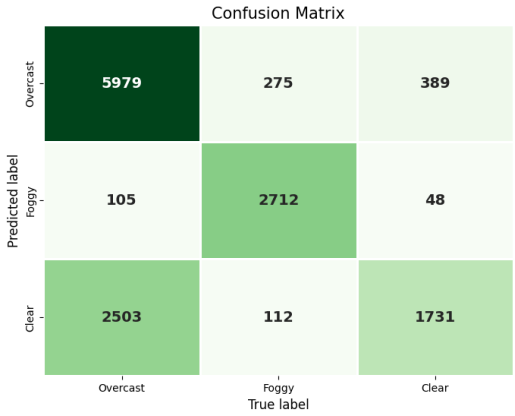

Figure 4: Confusion Matrix for Naive Bayes

On fig.3 and fig.4 are confusion matrixes[8] for both classifiers. As you can see, most classes are predicted as needed. An interesting aspect is that both classifiers have a problem predicting “overcast”. If they do not predict well, their second choice is “clear”.

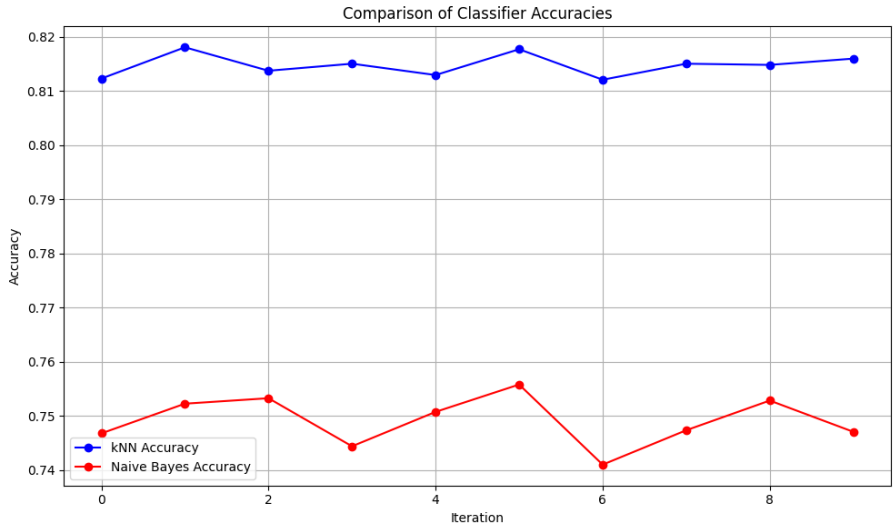

Figure 5: Comparison of the classifier accuracies

The k-nearest neighbor classifier has about 8 percentage points higher accuracy compared to the Gaussian Naive Bayes classifier on fig.5. This difference can be attributed to several factors. First, KNN is a non-parametric algorithm, meaning that it does not assume any particular distribution of the data. This flexibility allows it to effectively capture complex, nonlinear relationships in the feature space. GNB, on the other hand, assumes that the features have a Gaussian distribution and are independent of the class label. When the actual data distribution deviates from these assumptions, GNB’s performance can suffer. Second, KNN relies on the proximity of data points in the feature space, adapting well to different data distributions without making strong assumptions. In addition, KNN can mitigate the impact of outliers and noisy data by considering multiple nearest neighbors, which helps smooth out the impact of anomalous data points. On the other hand, GNB can be inaccurate under the significant influence of outliers, as they can distort the estimation of the parameters of the mean and variance of the Gaussian distribution for each trait. Together, these factors contribute to the higher accuracy we observed for KNN in our tests.

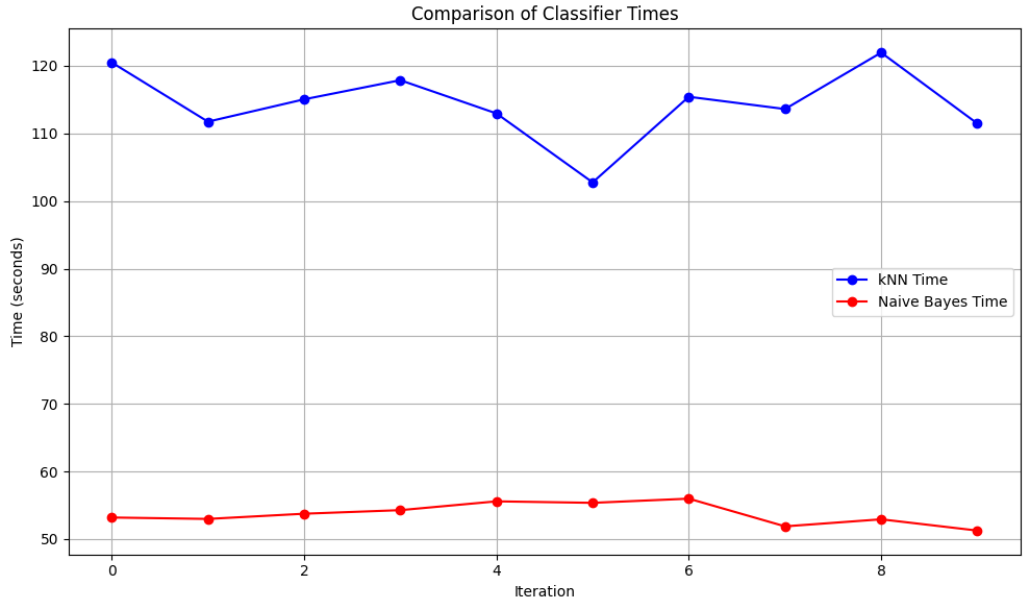

Figure 6: Comparison of the classifier times

From the time comparison on fig.6, the classifiers have completely different execution times. The KNN algorithm can be more time-consuming, especially for large datasets, due to the need to calculate the distance between each pair of points in the training set. Naive Bayes, on the other hand, being based on a simple probabilistic model, often exhibits lower computational complexity. In addition, differences in running times may also be due to differences in implementations of these algorithms and characteristics of specific data, such as the number of dimensions or the size of the dataset.

3.3.1. F1-Score

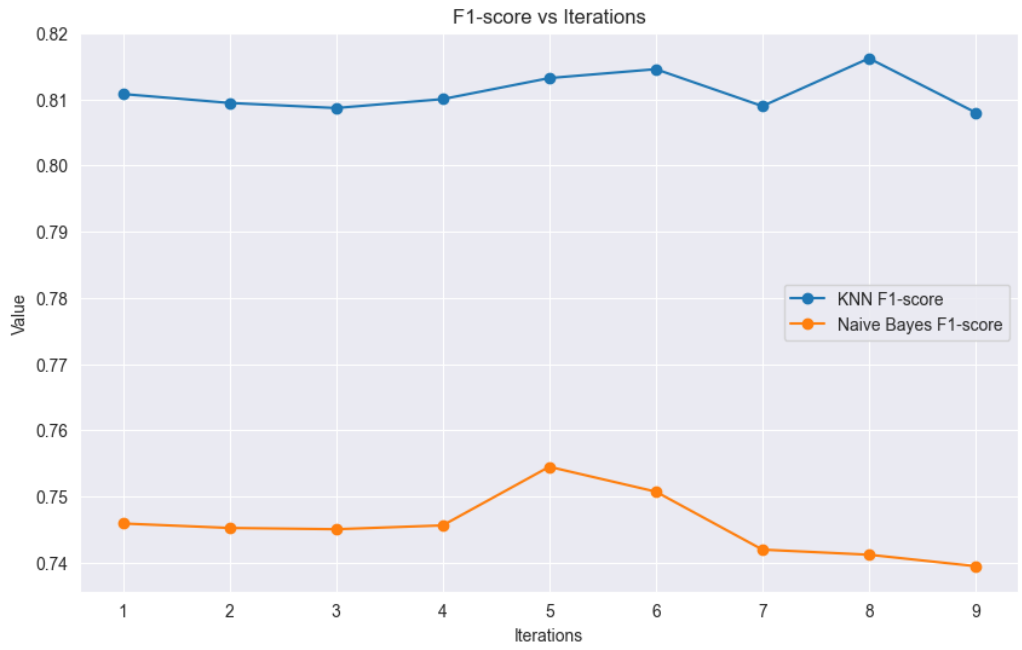

Figure 7: F1 Score

The F1-score or F1-measure is a measure of predictive performance. It is calculated from the precision and recall of the test, where the precision is the number of true positive results divided by the number of all samples predicted to be positive, including those not identified correctly, and the recall is the number of true positive results divided by the number of all samples that should have been identified as positive. Precision is also known as positive predictive value, and recall is also known as sensitivity in diagnostic binary classification. Using the built-in F1 metric from the sklearn library[9], nine iterations were conducted with data shuffling to calculate the results. This approach ensured robustness in evaluating the model's

performance across multiple trials and varying data distributions. Each iteration involved computing the F1-score, which provides a balanced measure of the classifier's precision and recall, thus capturing its ability to correctly classify positive instances while minimizing false positives and false negatives. The iterative process allowed for a comprehensive assessment of the model's effectiveness in handling different data configurations and revealed insights into its consistency and reliability. As seen on the fig.7, the results obtained by the F1-score from the sklearn library closely align with the results obtained using the accuracy calculation algorithm implemented by the authors in the tested classifier.

4. Conclusion

To sum up and recap, our study using the London weather dataset provided valuable insights into the functioning and performance of K-Nearest Neighbors (KNN) and the Gaussian Naive Bayes classifier (GNB). KNN is much easier to implement. Its concept is straightforward: it classifies new data points based on the most common class among the nearest neighbors. This simplicity in implementation makes KNN an agttractive option for quick and easy classification tasks. However, it has its limitations. KNN can be slower to classify, especially for large datasets, because it is necessary to calculate the distance between a new point and each point in the training set. This distance calculation can become computationally expensive as the size of the dataset increases, leading to longer classification times. On the other hand, the Bayes classifier, particularly the Naive Bayes classifier, may require more effort at the implementation stage. This is due to the need to calculate and model conditional probabilities and to make feature independence assumptions. Despite this initial complexity, the Naive Bayes classifier can be faster during the classification phase. It only requires calculating the conditional probabilities for each feature and applying Bayes' rule. Throughout of this study, we gained a large dose of knowledge. Implementing these algorithms and benchmarking their performance allowed us to gain practical experience with both KNN and GNB. We discovered firsthand the trade-offs between ease of implementation and computational efficiency.

References

[1] Marcin Woźniak, Michał Wieczorek, and Jakub Siłka. "BiLSTM deep neural network model for imbalanced medical data of IoT systems". In: *Future Generation Computer Systems* 141 (2023), pp. 489–499. URL: https://www.sciencedirect.com/science/article/pii/S0167739X22004095.

[2] Michał Wieczorek, Jakub Siłka, and Marcin Woźniak. "Neural network powered COVID-19 spread forecasting model". In: *Chaos, Solitons & Fractals* 140 (2020), p. 110203. ISSN: 0960-0779. URL: https://www.sciencedirect.com/science/article/pii/S0960077920305993.

[3] Rizwana Yasmeen. "K-Nearest Neighbor(KNN) Algorithm in Machine Learning". In: (2023). URL: https://medium.com/@rizwanayasmeen06/k-nearest-neighbor-knn-algorithm-in-machine-learning-d38d9638d7e0.

[4] R Nagapadma BS Sharmila. "Intrusion Detection System using Naive Bayes algorithm". In: (2019). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9019921%5C&casa_token=FEZMEU72iF8AAAAA:4D1LZ_1ZcT7dqbDdxFSbDGfqnG8TMb-vwrGeDgnZRzxV7YMyJGNupv8dmhmhkpsq2C6SJqZAmxc.

[5] MUTHUKUMAR.J. "Weather Dataset". In: 1 (2018). URL: https://www.kaggle.com/datasets/muthuj7/weather-dataset.

[6] Rocio Alaiz-Rodriguezand Nathalie Japkowicz. "Assessing the Impact of Changing Environments on Classifier Performance". In: (2008). URL: https://link.springer.com/chapter/10.1007/978-3-540-68825-9_2.

[7] Sandro Tosi. "Matplotlib for Python Developers". In: (2009).

[8] Aurélien Géron. "Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow, Wydanie II, aktualizacja do modułu TensorFlow 2". In: (2020).

[9] scikit-learn. "sklearn.metrics.f1_score". In: (2024). URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.