DATA MINING PROJECT

Sophie Hu (1657847)Shakeara Langenbach (1630680)

Date: 24 October 2023

Lecturer: Witek Ten Hove

Table of Contents

Link to GitHub profile, including other necessary files:
https://github.com/shakearalangenbach/datamining/tree/main

Introduction

This report presents a basic study of Data Mining. In this study, we were tasked with running different models on a suitable dataset. The dataset we chose contains information on income and various factors that may influence it. Based on the models and the dataset, we aim to reach a conclusion from which it follows which model best fits the dataset and is the most accurate.

Logistical Regression

Logistic regression is a statistical model that is used to predict the probability of a binary outcome. It is a supervised learning algorithm, which means that it is trained on a dataset of known inputs and outputs. Once the model is trained, it can be used to predict the probability of a binary outcome for new inputs.

Logistic regression works by fitting a logistic function to the data. The logistic function is a sigmoid function that outputs a value between 0 and 1. The output of the logistic function is interpreted as the probability of the binary outcome.

The logistic function is fitted to the data using a process called maximum likelihood estimation. Maximum likelihood estimation is a method for finding the parameters of a model that maximize the probability of the observed data.

Once the logistic function has been fitted to the data, the model can be used to predict the probability of a binary outcome for new inputs. To do this, the model simply calculates the output of the logistic function for the new inputs.

Logistic regression is a very versatile algorithm and can be used to solve a wide variety of problems.

KNN modeling

The k-Nearest Neighbors (KNN) algorithm is a versatile and intuitive machine learning technique used for classification and regression tasks. It is a non-parametric and instance-based learning algorithm, meaning it doesn't make explicit assumptions about the underlying data distribution. Instead, KNN relies on the similarity between data points to make predictions.

How KNN Works:

1. **Data Representation**: KNN starts with a dataset where each data point is characterized by a set of features. These features represent the properties or attributes of the data points.
2. **Choosing 'k'**: The 'k' in KNN stands for the number of nearest neighbors to consider. This is a crucial parameter that you need to specify before applying the algorithm. A smaller 'k' can make predictions more sensitive to noise, while a larger 'k' can make predictions more stable but potentially less accurate.
3. **Distance Calculation**: KNN uses a distance metric, typically Euclidean distance, to measure the similarity between data points. It calculates the distance between the

data point for which you want to make a prediction (the target point) and all other data points in the dataset.

4. **Finding Neighbors**: KNN then identifies the 'k' data points with the shortest distances to the target point. These data points are considered the "nearest neighbors" to the target point.
5. **Majority Voting (Classification) or Weighted Averaging (Regression)**: For classification tasks, KNN assigns the class label that is most common among the 'k' nearest neighbors to the target point. In regression tasks, it calculates a weighted average of the target values of the 'k' nearest neighbors.
6. **Prediction**: The predicted class label (for classification) or the predicted value (for regression) is assigned to the target point based on the results of the majority voting or weighted averaging.

KNN is a simple yet effective algorithm and is often used as a baseline model in machine learning. Its performance can be influenced by the choice of 'k' and the distance metric used. It's important to perform proper data preprocessing and feature scaling when working with KNN to ensure reliable results.

Naïve Bayes

Naive Bayes is a classification algorithm that is based on Bayes' theorem. Bayes' theorem is a mathematical formula for calculating the probability of an event, given some evidence. Naive Bayes makes a simplifying assumption that all of the features of a data point are independent of each other. This assumption is often not true in practice, but it allows Naive Bayes to be very simple and efficient to train and predict.

Naive Bayes works by calculating the probability of each class, given the values of the features. The class with the highest probability is then predicted.

Business understanding

In the current landscape, comprehending the determinants of earnings is of paramount importance. Your wages are intricately shaped by a range of factors, including your gender, relationship status, level of education, and racial background. Notably, as we enter 2023, it has become increasingly evident that the wage gap between men and women is widening, signifying the urgency of understanding and addressing these multifaceted influences on income. This growing disparity highlights the pressing need for organizations to navigate the intricate tapestry of these elements, fostering equitable workplaces and championing diversity and inclusion as indispensable features of the contemporary job market.

Data understanding

The dataset consists of 15 columns. The categories are briefly explained below to clarify how the data is structured.

1. **Age:** This is the age of the individual in question.
2. **Workclass:** This indicates the sector or industry in which the person works.
3. **FNLWGT:** This is the financial weight of the individual in question.
4. **Education:** This indicates the highest level of education attained.
5. **Educational Number:** This is a numerical representation of the education level. The number varies based on the actual education level. For example, a doctorate has a higher educational number than a high school diploma.
6. **Marital Status:** This denotes the person's marital status.
7. **Occupation:** This is the person's occupation or job.
8. **Relationship:** This indicates the person's relationship to the household.
9. **Race:** This indicates the person's ethnic background or race.
10. **Gender:** This denotes the person's gender.
11. **Capital Gain:** This indicates the amount of capital gain the person has earned.
12. **Capital Loss:** This indicates the amount of capital loss the person has incurred.
13. **Hours per Week:** This denotes the number of hours the person works per week.
14. **Native Country:** This indicates the country from which the person originates.
15. **Income >50K:** This appears to be a binary variable that indicates whether the person's income is greater than $50,000. “1” is above and “0” is below.

Data preparation

The code engages in a preliminary data understanding phase, including examining the dataset's shape, presenting summary statistics for numerical attributes, and assessing data types and non-null counts for each column. It then proceeds to address missing values within the dataset, identifying columns like "workclass," "occupation," and "native-country" as having missing entries. To ensure data completeness, the code eliminates rows containing these missing values through the **dropna()** method.

Detecting duplicate rows within the dataset is the subsequent task. The code employs the **duplicated()** method to identify and remove these duplicate records, leading to a clean dataset devoid of duplicates.

To prepare categorical variables for machine learning models, the code generates dummy variables for columns such as "workclass," "education," "marital-status," "occupation,"

"relationship," "race," "gender," and "native-country." This transformation enables the algorithm to work with categorical data effectively.

The code then segregates the features (X) from the target variable (y), where "income_>50K" is the target, and the features include all other columns in the dataset. The separation of these elements is crucial for constructing predictive models.

Lastly, the dataset is split into training and testing sets with an 80-20 ratio, facilitating model evaluation. The code verifies and presents the shape of both the training and testing datasets, signifying the number of rows and columns in each set.

In sum, this Python code adheres to a conventional data preprocessing workflow, encompassing the treatment of missing values, duplicate records, and the conversion of categorical data into a machine-learning-friendly format. The resultant clean and prepared dataset is now ready for the development and evaluation of predictive models.

Modelling

Model KNN

The code begins by importing the necessary libraries for K-Nearest Neighbors, specifically **KNeighborsClassifier** from scikit-learn. KNN is a machine learning algorithm used for classification tasks.

An instance of the KNN model is created, and it is initialized with default settings using **KNeighborsClassifier()**. This sets up the KNN model to be used for classification. The model is then trained on the training data, **X_train** and **y_train**, using the **fit()** method. During this training phase, the KNN model learns to classify data points based on the proximity to their nearest neighbors.

After the KNN model is trained, it is used to make predictions on the testing dataset (**X_test**) using the **predict()** method. The KNN algorithm assigns a class label to each data point based on the majority class among its nearest neighbors.
To evaluate the performance of the KNN model, the code prints the confusion matrix. This matrix provides a summary of the model's predictions and the actual outcomes, helping to assess true positives, true negatives, false positives, and false negatives.

Model Logistic regression

The provided Python code demonstrates the application of logistic regression to the prepared dataset for predictive modeling. Here's an elaboration of the steps involved:

The code starts by importing essential libraries for logistic regression, specifically the **LogisticRegression** model and metrics for evaluation, including **classification_report**, **confusion_matrix**, and **accuracy_score**. These libraries are crucial for training the logistic regression model and assessing its performance.

An instance of the logistic regression model is created and initialized with default settings using **LogisticRegression()**. This sets up the logistic regression model to be used for classification tasks.

The logistic regression model is fitted to the training data, **X_train** and **y_train**, using the **fit()** method. This step involves training the model to learn the relationships between the input features (X) and the target variable (y) based on the training data. The model is designed to make predictions based on this learned relationship.

Once the model is trained, predictions are generated for the testing dataset (X_test) using the **predict()** method. This step applies the trained logistic regression model to unseen data to predict the target variable values.

The code proceeds to evaluate the model's performance using various metrics. It prints the confusion matrix, which is a table that summarizes the model's predictions and actual outcomes. The matrix provides insight into true positives, true negatives, false positives, and false negatives.

Additionally, the code presents the classification report, which offers a detailed summary of key classification metrics. It includes precision (the accuracy of positive predictions), recall (the ratio of true positives to the total number of actual positives), F1-score (the harmonic mean of precision and recall), and support (the number of occurrences of each class). This report provides a comprehensive overview of the model's performance across different metrics.

Model NB

The provided Python code demonstrates the application of a Gaussian Naive Bayes classifier to the prepared dataset for predictive modeling. Here's a detailed description of the steps involved:

The code starts by importing the necessary libraries for Gaussian Naive Bayes (GaussianNB) and the required evaluation metrics from scikit-learn. These libraries are essential for training the Naive Bayes model and assessing its performance.

An instance of the Gaussian Naive Bayes model is created and initialized with default settings using **GaussianNB()**. This sets up the Naive Bayes model to be used for classification tasks. The model is then trained on the training data, **X_train** and **y_train**, using the **fit()** method. During training, the model learns the probability distribution of features based on the target variable.

Once the model is trained, it is used to make predictions on the testing dataset, **X_test**, using the **predict()** method. The model applies the Naive Bayes algorithm to estimate the probabilities of each class and assigns the class with the highest probability to each data point.

To assess the performance of the Naive Bayes model, the code prints the confusion matrix. This matrix provides a summary of the model's predictions and the actual outcomes, helping to evaluate true positives, true negatives, false positives, and false negatives.

Evaluation

NB

Classification Report: The classification report provides various metrics for evaluating the model's performance:

- **Precision (Positive Predictive Value):** For class 0, it is 0.81, which means that out of the instances predicted as class 0, 81% were actually class 0. For class 1, it is 0.66, indicating that 66% of instances predicted as class 1 were actually class 1.
- **Recall (Sensitivity or True Positive Rate):** For class 0, it is 0.95, meaning that the model correctly identified 95% of the actual class 0 instances. For class 1, it is 0.30, indicating that the model only identified 30% of the actual class 1 instances.
- **F1-Score:** It is the harmonic mean of precision and recall. For class 0, the F1-score is 0.87, and for class 1, it is 0.41.
- **Support:** This represents the number of instances in each class. There are 6137 instances of class 0 and 2001 instances of class 1.
- **Accuracy:** The overall accuracy of the model is 0.79, which means that it correctly predicted 79% of all instances.
- **Macro Average:** This is the average of precision, recall, and F1-score for both classes. In this case, it's 0.73, indicating a decent performance considering both classes.
- **Weighted Average:** This is a weighted average based on the number of instances in each class. It's 0.77 for precision, 0.79 for recall, and 0.76 for the F1-score.

In summary, the model performs well in correctly classifying class 0 instances (high precision and recall) but struggles with class 1 (lower precision and recall). The F1-scores also indicate that the model's performance is unbalanced between the two classes. It's essential to consider the specific context and objectives when interpreting these results and making decisions about model deployment or further improvement.

KNN

The classification report provides various metrics for evaluating the model's performance:

- **Precision (Positive Predictive Value):** For class 0, it is 0.81, which means that out of the instances predicted as class 0, 81% were actually class 0. For class 1, it is 0.56, indicating that 56% of instances predicted as class 1 were actually class 1.
- **Recall (Sensitivity or True Positive Rate):** For class 0, it is 0.92, meaning that the model correctly identified 92% of the actual class 0 instances. For class 1, it is 0.32, indicating that the model only identified 32% of the actual class 1 instances.
- **F1-Score:** It is the harmonic mean of precision and recall. For class 0, the F1-score is 0.86, and for class 1, it is 0.41.
- **Support:** This represents the number of instances in each class. There are 6137 instances of class 0 and 2001 instances of class 1.
- **Accuracy:** The overall accuracy of the model is 0.77, which means that it correctly predicted 77% of all instances.
- **Macro Average:** This is the average of precision, recall, and F1-score for both classes. In this case, it's 0.68, indicating a moderate performance considering both classes.

- **Weighted Average:** This is a weighted average based on the number of instances in each class. It's 0.74 for precision, 0.77 for recall, and 0.75 for the F1-score.

In summary, the KNN model performs reasonably well in correctly classifying class 0 instances, with high precision and recall. However, it struggles with class 1, where precision and recall are lower. The F1-scores reflect this performance imbalance between the two classes. The model's overall accuracy is 77%, and it achieves a moderate performance according to macro and weighted averages. As with any model evaluation, it's important to consider the specific context and objectives when interpreting these results.

Moreover, we focused on training and evaluating the KNN model:
It follows these key steps:

1. A list of different values for the number of neighbors is defined.
2. Lists to store accuracy values are initialized.
3. The code iterates through the list of neighbor values, training and evaluating KNN models for each value.
4. A plot is created to visualize the relationship between the number of neighbors and model accuracy.
5. The code selects 25 neighbors as the optimal choice for maximum accuracy.
6. A final KNN model is trained with 25 neighbors.
7. Predictions are made on the test data using the trained model.
8. The model's performance is evaluated using a confusion matrix and a classification report, which includes metrics like precision, recall, and accuracy for both classes (0 and 1).

Logistic regression

The classification report provides various metrics for evaluating the model's performance:

- **Precision (Positive Predictive Value):** For class 0, it is 0.80, which means that out of the instances predicted as class 0, 80% were actually class 0. For class 1, it is 0.72, indicating that 72% of instances predicted as class 1 were actually class 1.
- **Recall (Sensitivity or True Positive Rate):** For class 0, it is 0.97, meaning that the model correctly identified 97% of the actual class 0 instances. For class 1, it is 0.25, indicating that the model only identified 25% of the actual class 1 instances.
- **F1-Score:** It is the harmonic mean of precision and recall. For class 0, the F1-score is 0.88, and for class 1, it is 0.37.
- **Support:** This represents the number of instances in each class. There are 6137 instances of class 0 and 2001 instances of class 1.
- **Accuracy:** The overall accuracy of the model is 0.79, which means that it correctly predicted 79% of all instances.
- **Macro Average:** This is the average of precision, recall, and F1-score for both classes. In this case, it's 0.76, indicating a decent performance considering both classes.
- **Weighted Average:** This is a weighted average based on the number of instances in each class. It's 0.78 for precision, 0.79 for recall, and 0.75 for the F1-score.

In summary, the Logistic Regression model performs well in correctly classifying class 0 instances, with high precision and recall. However, it struggles with class 1, where precision and recall are lower. The F1-scores reflect this performance imbalance between the two classes. The model's overall accuracy is 79%, and it achieves a decent performance according

to macro and weighted averages. As with any model evaluation, it's important to consider the specific context and objectives when interpreting these results.

Deployment

KNN has the highest precision for Class 0, but lowest recall for Class 1 --> Good at identifying datapoint in Class 0, but is bad at identifying in Class 1.
NB has the highest recall for Class 0, lowest precision for Class 1. NB is good at finding all of the data points in Class 0, but sometimes wrongly classifies Class 1 as Class 0
Logistical regression has a good balance for precision and recall for both classes. It is good at identifying both classes.

For this problem, Logistical regression is probably the best algorithm to use to predict if a person with certain variables is likely to get an income above or below 50K.

Personal reflection

Shakeara Langenbach 1630680

I chose this minor based on my internship, where I built my first data model in PowerBI. This sparked my interest so much that I decided to find out more about the Data Driven Decisionmaking minor at the minor fair. I actually decided to sign up pretty quickly, because it seemed interesting to me to go further into data analysis. Although I have little to no experience with programming languages, I decided to take on the challenge.

Looking back on the past period, especially comparing the first Data Mining lesson with now, I can say that I have made quite a few steps. My knowledge of Python was not there, so the first lesson was overwhelming for me. It felt like I was 1-0 behind after the first lesson. I didn't know anything about programming, scripting languages, Anaconda... It caused a lot of chaos in my head. To create some clarity in my head, I watched videos at home. Videos of Anaconda and videos of Python. Based on this, I gained more and more knowledge and started to recognize things.

I also learned to work with Spyder more often during other lessons. To learn Python, I used Codédex. This is a platform for children and adults to learn Python in a fun and easy way. This is how I taught myself the basics. I also learned to work more with ChatGPT and Google Bard during this minor. Instead of seeing the use of it as laziness, I could appreciate it more and used it as my right-hand man. It was a quick and easy option to get answers to questions about Python codes and why something happened in a certain way. I learned to use ChatGPT in a good way during this assignment.

If I look at the data mining assignment itself, I didn't know at the beginning what was expected of me. Different algorithms were discussed, for example. I didn't even know what an algorithm was and that it is actually quite easy to do this with Python. Because it was actually completely unclear what the assignment actually entailed, I decided to approach our classmate Mahmoud together with Sophie. He had offered to help several times, because he had experience with it.

Initially, we had a dataset about YouTubers. We wanted to develop a prediction model that would predict the popularity of a channel based on certain variables. Afterwards, we realized that the dataset was not good enough for what we wanted to achieve. So in week 4 we had to change everything...

With the help of Mahmoud, we came up with the dataset about Income. With the help of this dataset, we could create a model that predicts whether a man or woman with variables x earns more or less than 50k per year. The idea was to perform a logistic regression on this and to test it for bias by variables such as Race and Gender with the help of a Fairness package. However, even with the help of ChatGPT, this did not work, so we were unable to do this. I found this very disappointing.

During the Data Mining process, Mahmoud supported us a lot. I liked that I could ask him questions and I found it very helpful that Mahmoud could explain things well. I have learned a lot from these moments.

Looking back on the first lesson and comparing it to now, I can say that I have certainly grown in knowledge and skills. It has also sparked my interest more, so much so that I have even started to think about whether I might want to do the Master.

Sophie Hu 1657947

Our project set learning goals, exploration and preparation, training, and evaluation. In my view, we achieved these objectives. We did encounter significant data cleaning challenges during the exploration and preparation phases. However, our dataset choice highlighted the role of thorough preparation, underscoring its direct impact on model performance.
The project involved implementing three algorithms. Understanding these models proved challenging, given the mix of coding and machine learning components. Yet, with the help of available resources, including Chat GPT, I can now affirm my grasp of these models and their real-world applications. During our brainstorm sessions, we came up with small changes and improvements that made it easier to see how well our project was doing. This project not only enhanced my understanding of the basics of machine learning models and their application to datasets, but also boosted my coding skills. Thanks to this project, I have become more confident at finding solutions and tackling coding tasks. Throughout my learning journey, I had certain goals in mind. With this project, my aim was to explore the programming world and Python's capabilities. Before entering this module, I was unaware of the crucial role of data preparation. During the exploration and preparation phases, I gained valuable experience, allowing me to contribute more to the coding process than I initially thought possible.

References

- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer.
- Scikit-learn documentation: K-Nearest Neighbors