Logistic Regression vs K Nearest Neighbors for the Purpose of Predicting Diabetes in Clinical Patients

Team Name: Group 64 | Github Link | Video Link

Aidan Hurwitz (jamesus88), Noah Adelson (NoahA6624), Christopher Bowers (CBowers28)

Problem Statement:

Diabetes is a widespread chronic disease affecting millions of people worldwide. Predicting diabetes based on patient data can significantly aid in early detection and preventive care. It can also serve as an initial screening tool to allow doctors to focus their attention on patients who are truly ill. Given a dataset of clinical patient records, our goal is to determine whether a person is likely to have diabetes using machine learning algorithms. Specifically, we will compare the performance of Logistic Regression and K-Nearest Neighbors (KNN) in terms of accuracy and computational efficiency.

Motivation:

Diabetes disproportionately affects individuals in lower-income demographics due to limited access to healthcare and expensive diagnostic procedures. The ability to use an accurate and computationally efficient algorithm for diabetes prediction could provide a cost-effective alternative for early diagnosis. By comparing Logistic Regression and KNN, we aim to determine which algorithm is better suited for this purpose in terms of predictive performance and computational cost.

Features:

We will consider the problem solved when we identify the algorithm that provides the highest prediction accuracy with the lowest computational complexity. This will be measured through standard classification metrics such as accuracy, precision, recall, F1-score, and computational complexity from both the training and prediction phase.

Data Overview:

We chose to use kaggle as our method of finding appropriate data. The specific dataset we will be using can be found at the link below and contains a great deal of information but most importantly it contains a sample size of 100,000 individuals allowing us to establish a large enough sample size for a reasonable test.
https://www.kaggle.com/datasets/ziya07/diabetes-clinical-dataset100k-rows/data

Data Explanation:

This section provides a more detailed overview of the dataset we selected. One of the primary reasons for choosing this dataset is its high quality — it contains no missing values and most features are already numerically encoded, which streamlines the preprocessing phase. The dataset includes important health indicators such as BMI, HbA1c levels, blood glucose levels, and other conditions like hypertension and heart disease. While it doesn't contain an extensive range of clinical data, it provides enough relevant information for building a reliable model without introducing excessive noise that is adequate in its complexity to make accurate predictions.

To prepare the data for analysis, we will apply encoding techniques to categorical variables such as gender.. By selecting a focused set of 5–7 well-engineered features, we aim to strike a balance between model simplicity and predictive accuracy, helping to minimize the risk of overfitting while maximizing performance.

Tools:

Language: Python
Development: Jupyter Notebook/pycharm, GitHub
Libraries: NumPy, Pandas, Scikit-learn, Matplotlib, Seaborn
Note: Scikit-learn will be used for data processing and we will be making our own implementation of logistic regression and KNN

Algorithms Implemented:

The project focuses on using two different AI models to predict the likelihood that a person has diabetes based on certain input features. The two models used are Logistic Regression and K-Nearest Neighbors (KNN). In this section, each method will be explained conceptually, followed by an overview of its implementation and an analysis of its time complexity for both training and prediction, assuming a single input example.

Following the order in the code, we begin with a conceptual explanation of Logistic Regression. Logistic regression is a model that outputs values between 0 and 1, representing the probability that an input belongs to a particular class. A threshold is then applied to determine the predicted class; in this project, a value of 0.8 is used to classify a positive case.
Logistic regression works by applying a weighted sum (a dot product) of the input features and model parameters, known as theta. These weights are learned using a process called gradient descent, which iteratively updates them to minimize prediction error.

Moving on to K-Nearest Neighbors (KNN), this algorithm is conceptually simple. It stores all the training examples and, when making a prediction, calculates the distance—typically using Euclidean distance—between the new input and each training example. It then identifies the K closest data points and uses majority voting to determine the predicted class. KNN is what is called a lazy learner meaning that it does not compute the input data into a function like in logistic regression rather it stores that data and directly uses that in the prediction phase.

For example, if K = 5, and the nearest neighbors are labeled [1, 1, 1, 1, 0], the algorithm would classify the input as 1, since that label appears most frequently. KNN relies on the assumption that similar examples tend to be located near each other in the feature space. This idea underpins any machine learning method that uses a distance function or kernel: proximity implies similarity.

Other Data Structures and Algorithms:

Very few other data structures were used as this was more of a test between the algorithms and the only notable uses were arrays and sets. However there were several algorithms used from the imported various libraries. These were mainly used for the cleaning of the dataset, one such is isnull() which finds the number of null values in the dataset. However these other algorithms were only used when cleaning the dataset along with some other tasks that do not include the training, execution, or practice of defining the models.

Distribution of Responsibility:

Chris: Data Preprocessing and selection, Analysis Section
Aidan: KNN, record the video
Noah: setup GitHub, Logistic Regression, Report Editing and Reflection

Analysis and Changes:

The project remained mostly unchanged from our proposal except for the fact that we decided not to develop a UI so that single inputs could be tested. This was due to our group finding that implementing these models from scratch was very time consuming along with the fact that a UI for this type of project didn't really contribute anything to it other than just being additional work.

In this implementation for logistic regression, we use three key functions: sigmoid, cost_function, and gradient_descent. The sigmoid function transforms the output of the weighted sum into a value between 0 and 1, making it suitable for classification. The cost function measures how well the current weights match the actual labels in the training data. Gradient descent then takes the derivative of the cost function to determine the direction of steepest descent and updates the weights accordingly. This iterative process continues until the model converges on an optimal set of parameters.

The computational complexity of the training phase (using gradient descent) is $O(K \cdot M \cdot N)$, where:
K is the number of iterations,
M is the number of training examples

N is the number of feature columns.

Once the model is trained, the prediction phase has a complexity of $O(M \cdot N)$, where:
M is the number of input examples, and
N is the number of features.

If we're predicting for only one input example, such as an individual patient, and the number of features is fixed (e.g., 17), the complexity simplifies to $O(1)$—a constant-time operation.

In this implementation for K-Nearest Neighbors (KNN), the main logic is contained within the knn_predict function. This function takes a labeled training dataset and predicts the labels for a set of test inputs based on the majority class of their nearest neighbors. For each test example, it calculates the Euclidean distance to every point in the training set. It then selects the k nearest neighbors by sorting these distances and retrieves their corresponding labels. A majority vote among these labels determines the predicted class.

The computational complexity of the prediction phase is $O(M \cdot N \cdot \log N)$, where:
M is the number of test examples,
N is the number of training examples, and
log N comes from sorting the distances to find the nearest neighbors.

If the number of test examples is 1 (i.e., a single prediction), the complexity becomes $O(N \cdot \log N)$.
KNN also does not require a training phase. Instead, it is a lazy learning algorithm, meaning all computation happens during prediction.

What can be concluded from this is that if you want to implement a simple model that has a longer testing phase but does not need a formal training KNN is best. However, if you prioritize the speed of the results, logistic regression is a much better algorithm/method as to produce a result for a single input after the model is trained the complexity is $O(1)$.

Reflection:
As a group the overall experience was a positive one. We all enjoyed working on a team that, at least in our minds, mimicked the real world. This was also a chance for all of us to learn something new that we had never implemented or researched before. We settled on this project because it was a challenge that allowed us to take our understanding of algorithm complexity and transfer it onto two very current algorithms that are used. Nevertheless, we found this a very challenging project due to the sheer scope of the information we needed to understand for it to be successful. Splitting it up made it manageable but it still required each of us to know what the

other two were working on so that everything integrated well. We still encountered challenges from inefficient runtime, to bad data, to reading pages of panda's documentation.

An example of us solving one of these problems is when we were implementing KNN it took ages to run as we had devised a separate function for euclidean distance that was called and then was added to a vector. To solve this we did two things. First we took a subset of the larger dataset and then we vectorized the euclidean distance function so another function call was unnecessary. That being said, if we were to do it over again we would all probably read more and understand the concepts first rather than diving in head first. Despite all of this we all came out of this project better programs than we were before it.

Chris's Comments: I enjoyed working on the project and bouncing ideas off of my other groupmates. It took a great deal of researching to develop this program and I am proud of what we as a team have achieved.

Noah's Comments: I enjoyed this project a lot and I learned a ton throughout the process. It was interesting collaborating on github with an interactive python notebook instead of .py files. It was also difficult to get my head around both the logistics regression and KNN algorithms. However I am super proud of the product we produced.

Aidan's Comments: This project was a great learning experience for me. We revised our workflow a few times to meet the project requirements, but in general our group was excellent at making a plan and executing our parts. I learned a lot about statistical models like log regression and KNN, and I am excited to use some of this knowledge in future research and studies.

References:

Logistic Regression: https://www.geeksforgeeks.org/understanding-logistic-regression/
KNN: https://www.geeksforgeeks.org/k-nearest-neighbours/
Numpy: https://numpy.org/doc/
Scikit-Learn: https://scikit-learn.org/stable/
Kaggle: https://www.kaggle.com/datasets/ziya07/diabetes-clinical-dataset100k-rows/data
Model Evaluation: https://www.geeksforgeeks.org/machine-learning-model-evaluation/
Seaborn: https://seaborn.pydata.org/
Pandas: https://pandas.pydata.org/docs/
MatPlotLib: https://matplotlib.org/
Jupyter Notebooks: https://jupyter.org/