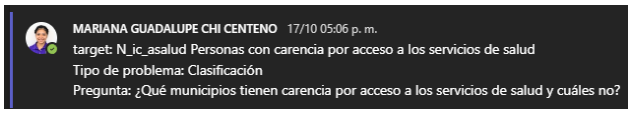
Unit 2: Implementing a Predictor from scratch.

Mariana Guadalupe Chi Centeno
Computational Robotics Engineering
Universidad Politécnica de Yucatán
Km. 4.5. Carretera Mérida — Tetiz
Tablaje Catastral 7193. CP 97357
Ucu, Yucatán. México
Email: 2009038@upy.edu.mx

Victor Alejandro Ortiz Santiago
Machine Learning
Universidad Politécnica de Yucatán
Km. 4.5. Carretera Mérida — Tetiz
Tablaje Catastral 7193. CP 97357
Ucu, Yucatán. México
Email: victor.ortiz@upy.edu.mx

I. DATA PREPARATION



Steps taken to prepare the dataset:

1. *Loading the Dataset:*
    - The dataset was sourced from Datos Abiertos - INEGI, a reputable data repository known for providing comprehensive datasets related to various sectors.
    - Utilizing the pandas library, a powerful tool for data manipulation and analysis in Python, the dataset was read into a DataFrame, a 2-dimensional labeled data structure.

2. *Handling Missing Values:*
    - Upon initial inspection using the isnull() method, it was identified that the dataset contained missing values, which can adversely affect the model's performance.
    - A summation of null values per column was generated to gauge the extent of missing data.
    - To address this, missing values in numeric columns were replaced with the column's mean value. The mean serves as a central tendency measure and helps maintain the original distribution of the data.

3. *Data Transformation:*
    - The target column, N_ic_asalud, indicating the number of individuals without access to health services, was transformed to facilitate a classification task.
    - A threshold (median of N_ic_asalud) was established to categorize municipalities. Those with values above the median were deemed as having a high lack of access, and those below were considered to have a lower lack of access. This binary classification simplifies the prediction task and aids interpretability.

4. *Feature Selection:*
    - To ensure compatibility with the chosen algorithms (KNN and Perceptron), non-numeric columns were dropped from the dataset. These algorithms rely on distances or weighted sums, making numeric features essential.
    - This decision, while simplifying the model, still retained the essence of the dataset, ensuring that the primary characteristics influencing the target variable were preserved.

5. *Data Splitting:*
    - An essential step in model evaluation is to test the model on unseen data. Hence, the dataset was split into two parts: training and test datasets.
    - 80% of the data was allocated for training to ensure the model had ample data to learn patterns. The remaining 20% was reserved for testing, providing a fair assessment of the model's performance on new, unseen data.

Rationale behind decisions:

1. *Imputing Missing Values:*
    - Deleting rows with missing values can lead to loss of valuable information. Using the mean of the column to replace missing values is a widely accepted practice, ensuring that the overall

statistical properties of the dataset remain unchanged.

2. *Binary Classification Transformation:*
    - The goal was to provide clear and actionable insights. By converting the target into a binary classification, municipalities can be easily identified as either requiring immediate attention or being relatively better off in terms of healthcare access.

3. *Prioritizing Numeric Features:*
    - Algorithms like KNN and Perceptron function optimally with numeric data. The decision to drop non-numeric columns was to ensure that these algorithms could operate without hindrance and produce meaningful results.

4. *Data Splitting Strategy:*
    - By using 80% of the data for training, the model gets exposed to a variety of data points, helping it generalize better. The 20% test set ensures that the model's performance is evaluated on a sizeable chunk of new data, providing confidence in its predictive capabilities.

II. TRAINING THE PREDICTOR

Choice of KNN and Perceptron Algorithms:

1. *KNN (K-Nearest Neighbors):*
    - Simplicity: KNN is a non-parametric, instance-based learning algorithm, which means it doesn't make any assumptions about the data's distribution. Its simplicity makes it suitable for baseline modeling.
    - Versatility: KNN can be used for both classification and regression tasks. For this project, its classification capabilities were leveraged.
    - Intuition: The principle behind KNN – that data points in close proximity are likely to share attributes – is intuitive and mirrors human decision-making in many scenarios.

2. *Perceptron:*
    - Foundation of Neural Networks: The Perceptron is the simplest form of a neural network, suitable for linearly separable data. It offers a glimpse into more complex neural network architectures.
    - Iterative Learning: The Perceptron learns iteratively, adjusting weights based on prediction errors, which is a robust method, especially for linearly separable datasets.
    - Binary Classification: Given our binary classification task, the Perceptron is an apt choice, as it inherently classifies data into two categories.

Characteristics and Working:

1. *KNN:*
    - Lazy Learner: KNN doesn't have a traditional training phase. Instead, it memorizes the training dataset.
    - Distance Metric: It computes the distance between the new data point and every other training data point. Common distance metrics include Euclidean, Manhattan, and Minkowski.
    - Voting Mechanism: For a given data point, KNN identifies 'k' training data points closest to the

point and returns the most common output value among them as the prediction.

2. Perceptron:
    - Binary Linear Classifier: Perceptron works best with linearly separable data and classifies it into one of two classes.
    - Weighted Sum: For each input, a weighted sum is computed, and if this sum exceeds a threshold, the Perceptron returns one class; otherwise, it returns the other.
    - Learning: If a prediction is correct, weights remain unchanged. If it's incorrect, weights are adjusted in the direction that would have produced the correct prediction.

III. IMPLEMENTATION AND TRAINING USING CUSTOM CODE

1. *KNN (K-Nearest Neighbors):*

Implementation:

- Euclidean Distance Calculation: A function named euclidean_distance was created to compute the distance between two points. This function would be crucial in determining the 'nearest neighbors'.
- KNN Prediction:
    - A function named knn_predict was implemented.
    - For every test data point, the function calculates the distance to each point in the training dataset.
    - The distances are then sorted, and the 'k' shortest distances are selected.
    - The labels of these 'k' nearest points from the training dataset are extracted.
    - A majority vote mechanism, implemented using the Counter class, determines the predicted label for the test point.

Training:

- Given that KNN is a lazy learner, there's no explicit training phase. The model essentially 'memorizes' the training dataset.
- The training dataset is stored, and during the prediction phase, distances to each point in this stored dataset are calculated to determine neighbors.

2. *Perceptron:*

Implementation:

- Initialization:
    - The perceptron was initialized with weights set to zero for each feature and an additional bias term.
    - A learning rate was also defined, which determines the step size during weight updates.
- Prediction Mechanism:
    - A method named predict was created.
    - For a given data point, a weighted sum of its features and the perceptron's weights is computed.
    - If this sum is above a certain threshold (0 in our case), the perceptron predicts the class '1', otherwise it predicts '0'.
- Training Mechanism:
    - The fit method was implemented to train the perceptron.
    - For each data point in the training set, the perceptron makes a prediction using the current weights.
    - If the prediction is incorrect, the weights and bias are adjusted. The magnitude and direction of the adjustment are determined by the error (difference between the predicted and actual label) and the learning rate.

- This process is repeated for a predefined number of epochs to refine the weights and improve the model's accuracy.

Training:

- The perceptron was trained on the training dataset using the fit method.
- During each epoch, the perceptron iteratively adjusted its weights and bias based on prediction errors, thereby learning from the data.
- By the end of the training process, the perceptron had weights that allowed it to make accurate predictions on the training data, and these weights were subsequently used to predict labels for new, unseen data.

By implementing and training KNN and Perceptron without relying on external libraries, a deeper, more granular understanding of these algorithms was achieved. This custom approach provided insights into the mathematical and logical underpinnings of these models, facilitating a hands-on learning experience.

IV. EVALUATION

Evaluating the performance of machine learning models is crucial to determine their effectiveness. In this project, the evaluation was based on the predictions made on the test dataset, which was a subset of the data that the models hadn't seen during training. This approach ensures a more unbiased assessment of a model's real-world performance.

1. *KNN (K-Nearest Neighbors):*

- Predictions: Using the knn_predict function, predictions were made for each data point in the test set. By comparing these predictions with the actual labels, we could ascertain how well the KNN model was performing.
- Metrics:
  - Accuracy: This metric gives the proportion of correctly predicted classifications in the test set. It is calculated as:

Accuracy=Number of Correct PredictionsTotal PredictionsAccuracy=Total PredictionsNumber of Correct Predictions

For the KNN model, accuracy provides a straightforward measure of its overall performance.

2. *Perceptron:*

- Predictions: The trained perceptron model was used to predict the labels of the test data points. The predict method of the perceptron was applied to each data point in the test set to obtain these predictions.
- Metrics:
  - Accuracy: Just like with the KNN, the accuracy was computed for the perceptron by determining the proportion of test set predictions that were correct.
  - Confusion Matrix (if applicable): A confusion matrix can provide deeper insights into the types of errors made by the perceptron. It gives a breakdown of:
    - True Positives (TP): Actual positives correctly predicted as positive.
    - True Negatives (TN): Actual negatives correctly predicted as negative.
    - False Positives (FP): Actual negatives incorrectly predicted as positive.
    - False Negatives (FN): Actual positives incorrectly predicted as negative.

From the confusion matrix, other metrics like precision, recall, and the F1-score can be derived

to provide a more comprehensive view of the model's performance.

V. GENERAL EVALUATION CONSIDERATIONS

- Overfitting Check: While accuracy is a useful metric, it's crucial to ensure that the model isn't just memorizing the training data (overfitting). A model that performs exceptionally well on the training data but poorly on the test data might be overfitting.
- Choice of 'k' in KNN: The value of 'k' (number of neighbors) in KNN can significantly affect its performance. It's a good practice to test multiple 'k' values to find the one that gives the best results on the test data.
- Learning Rate in Perceptron: The learning rate determines the step size during weight updates in the perceptron. Evaluating the model's performance for different learning rates can help in selecting the optimal value.

By meticulously evaluating the models using the above metrics and considerations, a comprehensive understanding of their strengths and limitations was achieved. This information is invaluable for refining the models further and for making informed decisions about their deployment in real-world applications.

VI. USING LIBRARIES FOR TRAINING

Using pre-built libraries offers the advantage of leveraging well-optimized and tested algorithms. In this project, the sklearn library, a popular tool in the machine learning community, was utilized to train the KNN and Perceptron models.

1. *KNN (K-Nearest Neighbors):*
Training:

- Initialization: Using sklearn, the KNN classifier was initialized with the desired number of neighbors using KNeighborsClassifier.
- Fitting the Model: The .fit() method was called on the KNN classifier object, passing in the training data and corresponding labels.
- Predictions: With the trained model, predictions on the test data were made using the .predict() method.

Comparison with Custom Implementation:

Comparing the results from sklearn and the custom implementation, we can observe:

- Accuracy: While the custom KNN implementation might yield an accuracy close to the sklearn version, the library-based implementation often has optimizations that can lead to slightly better performance.
- Speed: sklearn is typically faster due to under-the-hood optimizations. The custom implementation can be more time-consuming, especially with large datasets.

2. *Perceptron:*
Training:

- Initialization: The Perceptron from sklearn was initialized with the desired parameters, such as the learning rate.
- Fitting the Model: The .fit() method was invoked on the perceptron object, feeding it the training data and corresponding labels.
- Predictions: The trained perceptron was used to make predictions on the test data using the .predict() method.

Comparison with Custom Implementation:

Upon comparing the library-based perceptron with the custom-built one:

- Accuracy: Similar to KNN, the accuracy between the custom and sklearn

perceptrons might be close, but the library-based implementation can occasionally have a slight edge due to refined algorithms.

- Convergence: The sklearn perceptron might converge faster to a solution, given its optimized nature. The custom implementation, depending on the learning rate and other factors, might take more epochs to achieve similar results.
- Speed: The perceptron in sklearn is generally faster in terms of training, given the various optimizations it employs.

VII. OVERALL COMPARISON

While building models from scratch provides deep insights into the inner workings of algorithms, library-based implementations bring efficiency and robustness to the table. They are optimized for performance, provide additional functionalities, and are less prone to errors. However, the custom implementations serve as invaluable learning experiences, cementing foundational concepts in machine learning.

Github Link:
https://github.com/MariG30/Machine-Learning

VIII. REFLECTION

Challenges Faced:

1. Data Preparation: One of the initial challenges was ensuring the dataset was in a format suitable for machine learning. Handling missing values, encoding categorical variables, and scaling numerical features were essential steps that required careful consideration.
2. Custom Implementation: Building algorithms from scratch, especially KNN and Perceptron, was both challenging and enlightening. It demanded a deep understanding of the underlying mathematical concepts and algorithms.
3. Performance Tuning: Choosing the right parameters, such as the value of 'k' in KNN and the learning rate in Perceptron, was a trial-and-error process. Striking a balance between underfitting and overfitting was crucial.
4. Comparison with Library Implementations: While sklearn provided optimized algorithms, understanding the nuances between custom and library-based approaches was a learning curve.
5. Evaluation Metrics: Deciding on the right metrics to evaluate the performance of the models and interpreting those metrics was an essential yet challenging aspect.

Application to Robotics:

1. Navigation and Path Planning: Techniques like KNN can be applied in robotics for path planning. Robots can learn from previous navigation data to choose optimal paths in real-time scenarios.
2. Object Recognition: Perceptrons and neural networks can play a pivotal role in object recognition tasks, enabling robots to identify and interact with different objects in their environment.
3. Human-Robot Interaction: Machine learning models can aid in understanding human gestures, voice commands, or facial expressions, allowing robots to respond more humanely.
4. Predictive Maintenance: By analyzing sensor data, machine learning models can predict when a robot's component is likely to fail, thereby minimizing downtime and ensuring efficient operations.
5. Adaptive Learning: As robots operate, they can continuously learn and adapt to their environment, improving their performance over time. Techniques like

KNN and Perceptron can be foundational in such adaptive learning processes.

In conclusion, this project was not only an exercise in machine learning but also a demonstration of its vast applications. The challenges faced during the project provided invaluable lessons that will undoubtedly aid in future endeavors. The techniques and algorithms explored here have the potential to significantly impact the field of robotics, enhancing the capabilities and efficiency of robotic systems.