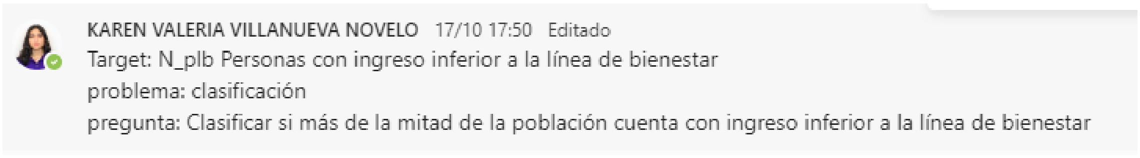
PREDICTOR FROM SCRATCH

MACHINE LEARNING

KAREN VALERIA VILLANUEVA NOVELO
IRC9B
2009146

Screenshot



KAREN VALERIA VILLANUEVA NOVELO 17/10 17:50 Editado
Target: N_plb Personas con ingreso inferior a la línea de bienestar
problema: clasificación
pregunta: Clasificar si más de la mitad de la población cuenta con ingreso inferior a la línea de bienestar

Journal part

The first thing I did was to download the dataset called 'Indicadores de pobreza, pobreza por ingresos, rezago social y gini a nivel municipal,1990, 2000, 2005 y 2010'. The target I chose was 'N_plb Personas con ingreso inferior a la línea de bienestar', which was providing the numerical value. I was solving a classification problem, where it is necessary to have categories, therefore, I needed to transform my target from numerical to categorical. The question I was solving was to classify if the different populations depending on if most of the half of the population was over the wellness income line. With that, I was able to determine two categories: '1' if more than the half of the population would be under the defined wellness income line and '0' if less than the half of the population was on that situation.

Once I defined my approach, I started to make the necessary modifications to the dataset. Because I was going to work with classification algorithms, and non-numerical data can affect the performance of it, I decided to drop the non-numerical columns that I identified, also, the ones that only were mentioning the number of the municipalities, because my problem was only focused on working with the population regardless the origin of them, I was not going to take into account the name of the place or the number of the place, therefore, I decided to drop those columns too.

Also, it was necessary to fill the empty cells on the dataset, this because the missing data can lead to incorrect results, and in order to have the best results possible, I decided to input the missing values with the mean of the same column, in order to avoid the reduction of my data size. Also, it would make it compatible with the knn and perceptron algorithms applied to train the model.

For the conversion of my numerical target into a categorical one, I decided to create a new column called 'Conversion of target' with the labels 1 and 0, as I explained in a previous paragraph, that would be the two categories my model will be trained for. The whole process consisted on: First, it was calculated the percentage that represented the people under the wellness income line for each population taking into account the total population and the numerical values from my original target; once that was calculated, it was assigned a value of 1 to the ones where the percentage of was more than 50% and '0' when tit was less than 50% of the population.

Once that new feature was created, I used it as the target for implementing the algorithms. It was already the last column of my dataset, so I did not need to make any modification related to that part.

Algorithms:

For the application of the algorithms, I selected knn and perceptron, because they were simple to apply and adaptable for my classification model.

The KNN algorithm is a supervised machine learning algorithm that is applied to classification tasks, and the way it works is the following: KNN searches for the k-nearest neighbors from the training dataset. It then assigns the class label to the new data point based on the majority class among its nearest neighbors. KNN doesn't make assumptions about the data distribution, which makes it versatile and effective for both linear and non-linear classification tasks. The hyperparameter that should be considered is the ‘k’ , which is the number of neighbors to consider, that will affect the performance of the algorithm.

In summary, KNN is a simple yet powerful algorithm for classification that leverages the concept of proximity and the majority vote to assign class labels to new data points, making it suitable for a wide range of classification problems.

Perceptron is a fundamental supervised machine learning algorithm used for binary classification tasks. Perceptron takes a set of input features, assigns weights to each feature, and calculates the weighted sum. If the weighted sum exceeds a specified threshold, the perceptron predicts one class; otherwise, it predicts the other class. It's a linear model that can learn a decision boundary that separates the two classes.

Another characteristic about perceptron, is that it supports online learning, where weights can be updated incrementally, making it applicable to real-time or streaming scenarios.

For the development of the algorithms that used libraries, it was mainly used the library scikit-learn, which offers a unified way to work with various machine learning algorithms, making it easy to adapt it to different classification tasks, including KNN and Perceptron.

Scikit-learn provides a *KNeighborsClassifier* class, which is used for KNN-based classification tasks, where it can be created an instance of this class and specified the number of neighbors (k) as a hyperparameter.

The Perceptron algorithm is also available in scikit-learn as the *Perceptron* class. It can be created an instance of this class to build a Perceptron-based classifier.

For the training of the data, it was divided into 80% of the dataset for training and 20% for the testing.

The model uses the training set to adjust its internal parameters or weights to make predictions. It learns to capture the relationships between the input features and the target variable, that in this case is 0 or 1, depending on scenario of the population.

Description of the algorithm of KNN from scratch

The code begins by setting a random seed with np.random.seed(0). This is done to ensure that any random processes within the code are reproducible. After setting the seed, the dataset df is shuffled using the .sample(frac=1) method. Shuffling the data is essential to randomize the order of records, reducing potential bias introduced by the original data order.

The following step consists of splitting the dataset into a training set and a test set. The intention is that 80% of the data is allocated for training (train_set), while the remaining 20% is reserved for testing (test_set).

The next thing is that the features and labels are separated for both the training and test sets. The features are stored in X_train and X_test, while the labels are stored in y_train and y_test.

The mean and standard deviation of the features in the training set are calculated. This information is used for normalizing the data, which is a common preprocessing step in machine learning. The mean and standard deviation are calculated using NumPy.

Now, it is defined a custom KNN function, knn, which takes the training data, training labels, test data, and a parameter k representing the number of nearest neighbors to consider. The KNN function computes distances between test points and all training points, selects the k nearest neighbors, and predicts the label of a test point by majority voting. The KNN function is used with k=3 to make predictions on the test set.

Finally, it is calculated the accuracy of the predictions by comparing the predicted labels (y_pred) to the actual labels in the test set (y_test). The accuracy is reported as a percentage and printed to the console.

Description of the perceptron from scratch

In the first part, the features were stored in the 'X' variable, while labels are stored in 'y'.

Because the algorithm was made from scratch, a step function activation, which acts as the activation function for the perceptron, is defined as step_function(z). It returns 1 if 'z' is greater than or equal to 0, and 0 otherwise. This function is a simplified form of an activation function used in perceptrons.

For the following part, weights and bias are initialized randomly. The weights are initialized using random values with np.random.rand(X.shape[1]), where 'X.shape[1]' represents the number of features. The bias is initialized with a random value using np.random.rand().

The learning rate is defined as 0.1, which determines the step size for weight and bias updates. The number of training epochs is set to 100. The code enters a loop that iterates through the specified number of epochs.

After training, the perceptron is used to make predictions by computing the weighted sum plus bias ('z') and applying the step function to obtain 'predictions.' The code calculates the accuracy of the Perceptron's predictions. It compares 'predictions' to the actual labels ('y') to determine the proportion of correct predictions. The accuracy is expressed as a percentage and printed to the console.

Link

https://github.com/ValeriaVillanueva/PREDICTOR-.git

Reflection

The main challenge of this project, for me, was to try to think about how the best way for me was to prepare my dataset, and how I would transform my target into a variable that could be useful for applying both of my algorithms. Also, to understand how the different algorithms work, how they are conformed, their steps and how the hyperparameters could make the result of the estimations change. . Investigating the role of hyperparameters was an eye-opener, as even slight tweaks could drastically alter the results. The choice of these settings was a crucial decision in the project's success.

In the field of Robotics, this can be applied to the artificial vision area, and the automation field, where tasks related with classification of different products, for example, in robotics, robots often encounter machine learning tasks such as object classification, autonomous navigation, and sensor-based decision-making. Both K-Nearest Neighbors (KNN) and the Perceptron are supervised learning algorithms that can be used for classification tasks, which are crucial in robotics.