



**UNIVERSIDAD
POLITÉCNICA
DE YUCATÁN**



Ángel Iván Mayo Carrillo


2009093

Computational Robotic Engineering

Machine Learning: Prediction model

23/10/2023

A screenshot of the post in the general channel, where you state your Selected target, your selected problem (regression or classification), and the question that the predictor will answer.



ANGEL IVAN MAYO CARRILLO 13/10, 9:58

1. feature: cpic_rezedu Carenias promedio de personas en rezago educativo

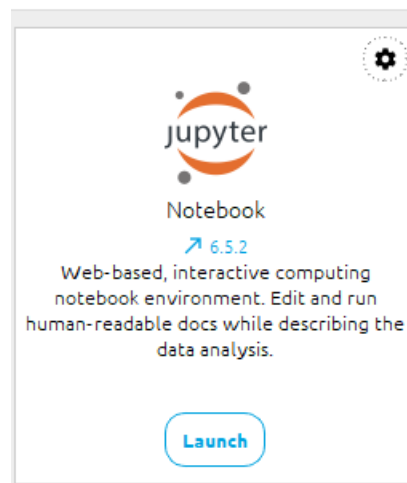
2. tipo de problema: Clasificación

3. pregunta: clasifica en las siguientes categorías las carencias de rezago educativo (Severa, Media, Baja)

The question changes because it is not properly stated, the correct one is Predict in which classification of educational backwardness the states belong.

1. First step is to set the jupyter

This step is easy, once Anaconda is installed you just need to open it and select the jupyter notebooks option and it will immediately open folders where you can create files.



2. Import libraries

For this algorithm we use pandas, numpy and scikit-learn for accuracy scores. Pandas is a very well known, and powerful, library for data manipulation and analysis. It can provide functions to work with data structures like tabular inside dataframes and is necessary for KNN because with pandas you perform the preprocess the dataset by moving features, removing, adding, read CSV files, etc

Numpy is another known library for python

3. Load dataframe

Dataframes are different from each other and not all dataframes can be read the same way. For this one we had to use a specific coding command to read the dataframe.

```
df = pd.read_csv('Indicadores_municipales_sabana_DA.csv', encoding='latin_1')
```

4. Clean dataframe

- a. Remove unnecessary columns

dataframes are usually really large with several features and rows, this one has more than one hundred and for a KNN classification not all of them are necessary or even possible to compute. The first rows that are names (str) and numerical unique values are not necessary. There are columns where they apply a classification to the data values but I removed them because I will use my own metrics, therefore I can drop them.

```
columns_to_remove = ['mun', 'clave_mun', 'ent', 'nom_ent', 'nom_mun', 'gdo_rezsoc00',  
'gdo_rezsoc05', 'gdo_rezsoc10']  
df.drop(columns_to_remove, axis=1, inplace=True)
```

b. Fill empty spaces

It is common to have empty spaces inside dataframes. There are many techniques to solve this problem and for this occasion I use average of the column. But first, we need to find how many empty values are.

```
null_values = df.isnull().sum()  
# this function will find how many empty values exists
```

Once we know that in fact we have empty values, we proceed to fill them with average values of the feature.

```
# Calculate column averages  
column_averages = df.mean()  
  
# Replace null values with column averages  
df.fillna(column_averages, inplace=True)
```

c. Move our desired feature to the last

This step can be performed earlier but for the documentation I put it here. Moving our desired feature to the end of the dataframe is a good practice and also helps you to visualize the feature easily. For this we use a reindex function.

```
df = df.reindex(columns=[col for col in df.columns if col != 'cpic_rezedu'] + ['cpic_rezedu'])
```

d. Create our own metrics for the classification

Now we can create our own metrics. Using our desired feature and a function we can divide into three categories: Baja, Media and Severa. For this we create a new feature at the end and apply labels to it, we also set the new values as str format to avoid complications. Once the new feature is done, it is separated from the main dataframe. By doing that we will preserve the correct order for our dataframe.

```
bin_edges = [0, 2.51, 4.51, float('inf')]  
bin_labels = ['Baja', 'Media', 'Grave']
```

```
df['class'] = pd.cut(df['cpic_rezedu'], bins=bin_edges, labels=bin_labels)
df['class'] = df['class'].astype(str)

df2 = df[['class']]
df.drop('class', axis=1, inplace=True)
```

e. Divide the dataframe into test and train

KNN models needs training and test samples. A normal practice is to divide in eighty percent with training samples and the remaining twenty percent is for testing. To divide we need to know how many features and rows will be inside each cluster of information, and using `iloc` we make them into a numerical value that can be used for mathematical operations.

```
X_train = df.iloc[:1964, 0:130].values
X_test = df.iloc[1964:, 0:130].values

y_train = df2.iloc[:1964, 0].values
y_test = df2.iloc[1964:, 0].values
```

All steps taken to train your predictor. Including an explanation on Why to choose such algorithm and an explanation on its characteristics.

5. Create the KNN model

a. Euclidian function

This function is important because the Euclidean distance is based on the Pythagorean theorem and measures the straight-line distance between two points in Euclidean space. This makes it an intuitive choice for calculating the similarity or dissimilarity between data points. It is also very computational efficient and this saves energy and resources and is a function that can be adapted to several scenarios without changing much of its core. The euclidian function we use is the following

```
def euclidian(p1,p2):
    dist = np.sqrt(np.sum((p1-p2)**2))
    return dist
```

b. Prediction

The prediction function will continue the prediction for the model and also storages the labels depending on the index.

```
def predict(x_train, y , x_input, k):
    op_labels = []
```

c. Main K Nearest Neighbor function

In a summary, the KNN model is basic machine learning model that makes predictions based on the memory and position of the values, neighbors, where the classification of a point is determined by the majority of the neighbors it has inside a certain K which says the amount of neighbors it considers. The first step is setting a for loop for every value, then the distances are storage and another for loop is used to store the distances and their calculations and determine them. Then a function that sorts the distance is applied, the sorting is from the smallest to the largest, and store the labels for distances.

Voting is the final section. Here the labels are count and a function is used to detect the one that has majority, the label will be assign to the point we want to predict.

```
for item in x_input:

    #distances storage
    point_dist = []

    #data to be processed
    for j in range(len(x_train)):
        distances = euclidian(np.array(x_train[j,:]) , item)
        #Calculating the distance
        point_dist.append(distances)
    point_dist = np.array(point_dist)

    #preserve the index
    dist = np.argsort(point_dist)[:k]

    labels = y[dist]

    #voting
    unique_labels, label_counts = np.unique(labels, return_counts=True)
    most_common_label = unique_labels[np.argmax(label_counts)]

    op_labels.append(most_common_label)

return op_labels
```

Apply the model

```
predictions = predict(X_train, y_train, X_test, k=3)
```

```
In [16]: predictions
```

```
Out[16]: ['Media',  
          'Media',  
          'Media',  
          'Media',  
          'Media',  
          'Media',  
          'Media',  
          'Media',  
          'Media',  
          'Media',  
          'Media',  
          'Media',  
          'Media',  
          'Media',  
          'Media',  
          'Media',  
          'Media',  
          'Media',  
          'Media']
```

d. Steps to measure the accuracy

For my model an accuracy library is used. This is because is way easier and precise than making more functions. I use the following library.

```
from sklearn.metrics import accuracy_score  
  
# TO APPLY THE FUNCTION WE JUST NEED THE FOLLOWING  
  
accuracy_score(y_test, predictions)
```

e. KNN with libraries

The preprocessing is the same, the difference comes when using the model and loading the data in the sections. Sklearn has its own KNN and classification that is present in the following code. Here the K (amount of neighbors to consider), train and test sets are defined easily and the main code is behind the library and perform the same as the previous.

```
# Import necessary libraries  
from sklearn import datasets  
from sklearn.model_selection import train_test_split  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import accuracy_score  
  
# Load dataset  
X = df.iloc[:1964, 0:130].values # Features  
y = df2.iloc[:1964, 0].values # Target variable
```

```

# Split the dataset into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a KNN classifier with a specified number of neighbors (k)
k = 3
knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Fit the classifier on the training data
knn_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn_classifier.predict(X_test)

# Print the predictions
print("Predictions:")
for i in range(len(X_test)):
    print(f"Predicted: {y_pred[i]}, Actual: {y_test[i]}")

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

```

f. Comparison of results and accuracy

Both codes compile and perform the operations without problems. There are only small differences in the accuracy.

With sklearn	Without sklearn
Accuracy: 90.17%	0.9065040650406504

6. Link for github

<https://github.com/IvanMayo0/MachileLearning.git>

7. Opinion

I had very few knowledge about KNN and having to interpret information and create my own code supposed a heavy challenge, I don't have the necessary programming knowledge to understand everything and specially when variables use "__". Another challenge I faced was the concept of having more than two features for a model, in this case I had more than a hundred in one variable. It was also the first time that I used Pandas so having to read the most common commands required time. I didn't had much troubles with the preprocessing of the dataframe because there is plenty of documentation for that. In general, it was a demanding task that if not having the possibility to deliver it on Monday, I wouldn't have finished on the Friday.