Preprocessing of Data in Machine Learning

Prof.Dr. Bahadır AKTUĞ
Machine Learning with Python

**Compiled from sources given in the references.*

Machine Learning

- Data preprocessing is an integral step in Machine Learning as the quality of data and the useful information that can be derived from it directly affects the ability of our model to learn; therefore, it is extremely important that we preprocess our data before feeding it into our model.
- Preprocessing covers a variety of processing before the actual machine learning process. The main preprocessing techniques are:
  - Handling Null Values
  - Scaling
  - Handling Categorical Variables
  - One-Hot Encoding
  - Multicollinearity

Handling Null Values

- Neither classification nor regression algorithms allow NULL or NaN values.
- Since data nearly always comes in a Pandas DataFrame, pandas functions can be directly used to deal with them.
- Basic strategies:
  - Dropping NULL values (either rows or columns)
    - df.dropna()
  - Filling NULL values with zero or constant value
    - df.fillna(0)
  - Fill NULL values with some function values (Imputation)
    - from sklearn.impute import SimpleImputer
      imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
      imputer = imputer.fit(df[['Weight']])
      df['Weight'] = imputer.transform(df[['Weight']])

Scaling

- Scaling is another integral preprocessing step.
- Neither classification nor regression algorithms allow NULL or NaN values.
- Since data nearly always comes in a Pandas DataFrame, pandas functions can be directly used to deal with them.
- Basic strategies:
  - Standardization: In Standardization, we transform our values such that the mean of the values is 0 and the standard deviation is 1.
    - from sklearn.preprocessing import StandardScaler
      std = StandardScaler()
      X = std.fit_transform(df[['Age','Weight']])
  - Scaling between an interval: Scaling the data between 0 and 1
    - from skln.preprocessing import MinMaxScaler
    - scaler = MinMaxScaler()
    - scaled = scaler.fit_transform(data)

Handling Categorical Variables

- Categorical variables are basically the variables that are discrete and not continuous. e.g. color of an item is a discrete variable whereas its price is a continuous variable.
- Categorical variables are further divided into 2 types —
  - **Ordinal categorical variables** — These variables can be ordered. e.g. Size of a T-shirt. We can say that M<L<XL.
  - **Nominal categorical variables** — These variables can't be ordered. e.g. Color of a T-shirt. We can't say that Blue<Green as it doesn't make any sense to compare the colors as they don't have any relationship.
- Basic strategies for ordinal categorical variables:
  - Using map() function
    - size_mapping = {'M':1,'L':2}
      df_cat['size'] = df_cat['size'].map(size_mapping)
  - Using Label Encoder
    - from sklearn.preprocessing import LabelEncoder
    - class_le = LabelEncoder()
    - df_cat['classlabel'] = class_le.fit_transform(df_cat['classlabel'].values)

One-Hot Encoding

- We have already seen how to handle ordinal categorical variables
- Since there is priority relation between nominal categorical variables, one-hot encoding is one preferred method to handle them.
- One-Hot Encoding creates 'n' columns where n is the number of unique values that the nominal variable can take. e.g. if color can take Blue, Green and White then we will just create three new columns namely, color_blue, color_green and color_white and if the color is green then the values of color_blue and color_white column will be 0 and value of color_green column will be 1.
- So out of the n columns, only one column can have value = 1 and the rest all will have value = 0.
- One-Hot Encoding is a pretty cool and neat hack but there is only one problem associated with it and that is Multicollinearity.
- Multicollinearity occurs in our dataset when we have features which are strongly dependent on each other.
- In this case we have features color_blue,color_green and color_white which are all dependent on each other and it can impact our model.

Multicollinearity

- Multicollinearity occurs in our dataset when we have features which are strongly dependent on each other.
- The easiest method to identify Multicollinearity is to just plot a pairplot and you can observe the relationships between different features.
- If you get a linear relationship between 2 features then they are strongly correlated with each other and there is multicollinearity in your dataset.
- A simple method do deal with multicollinearity:
  - df_cat = pd.get_dummies(df_cat[['color','size','price']],drop_first=True)
- So here color_blue will be dropped and we will only have color_green and color_white.
- The important thing to note here is that we don't lose any information because if color_green and color_white are both 0 then it implies that the color must have been blue.

References

1 https://scikit-learn.org/
2 https://towardsdatascience.com/
3 McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython 2nd Edition.
4 Albon, C. (2018). Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning
5 Géron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems 1st Edition
6 Müller, A. C., Guido, S. (2016). Introduction to Machine Learning with Python: A Guide for Data Scientists
7 Burkov, A. (2019). The Hundred-Page Machine Learning Book.
8 Burkov, A. (2020). Machine Learning Engineering.
9 Goodrich, M.T., Tamassia, R., Goldwasser, M.H. (2013). Data Structures and Algorithms in Python, Wiley.
10 https://towardsdatascience.com
11 https://docs.python.org/3/tutorial/
12 https://towardsdatascience.com/introduction-to-data-preprocessing-in-machine-learning-a9fa83a5dc9d
13 https://developers.google.com/edu/python/
14 http://learnpythonthehardway.org/book/