Ayesha Ashraf

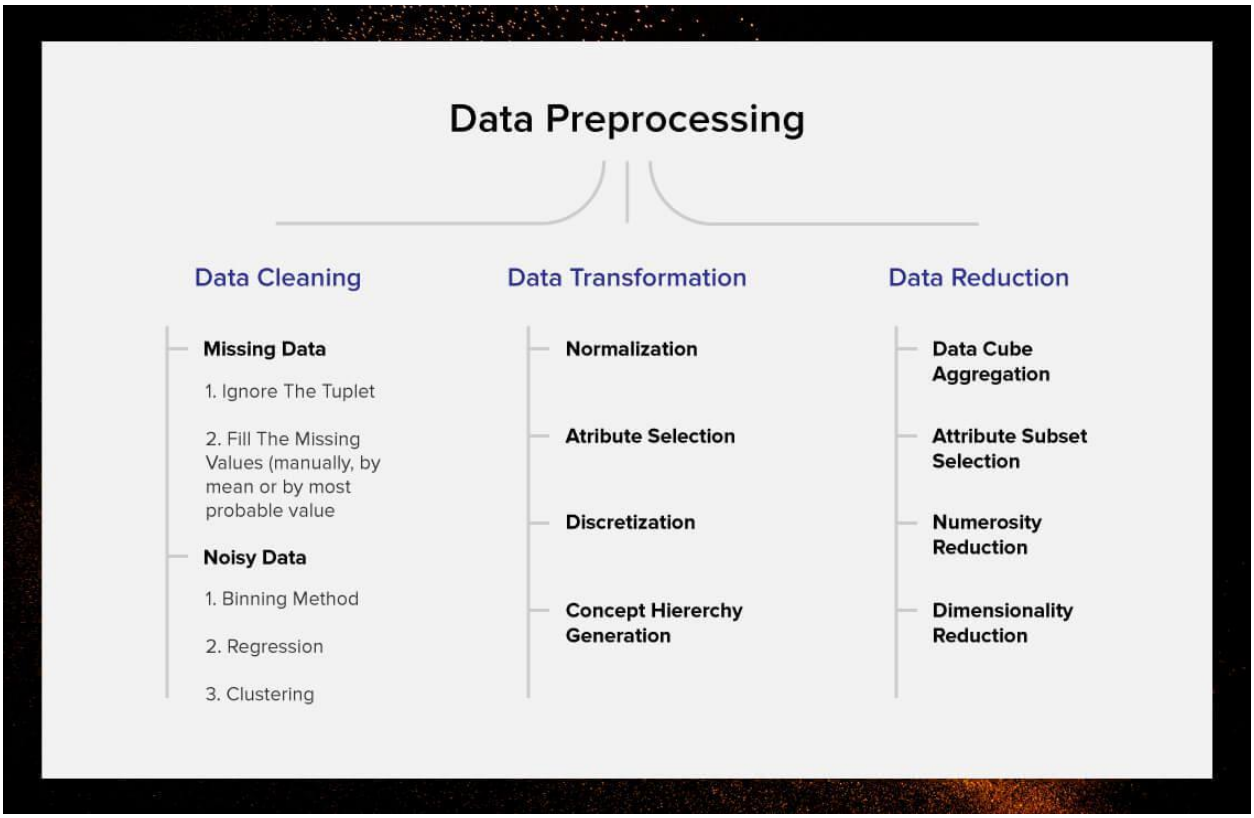
Task 24

Data Preprocessing

What is Data Preprocessing ?

It is the technique of making raw data into more meaningful data or the data which can be understood by the Machine Learning Model. Real-world **data** is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. To tackle this data preprocessing technique is introduced. We will talk about some of the techniques of data preprocessing which are :

- Vectorization
- Normalization
- Handling Missing Values



Vectorization

All input values we give to our machine learning model should be in the form of Integers. Because Neural Networks or commonly machine learning model handles integer representation of data.

- Whether we have dataset which contains different categories of classes then we have to do one-hot encoding to convert that categorical values into integer representations, which can be handled by our model.
- If it is an image we have to convert it into to the pixel values which can be interpreted by the neural networks. We cannot send a image directly to the neural networks, we have to convert it into an array of pixel values.

Whatever data you need to process — sound,image or text you must turn them into integers or tensors which is called data vectorization. All Inputs and targets in a neural network must be tensors of floating point data(or, in specific cases, tensors of integers). We can use **one-hot encoding** to convert categorical values to integer values. And there are many more techniques which can be used to do vectorization .

Normalization

Normalization means transforming features to be on a similar scale.

This helps to improves the performance and training stability of the model.

Example, In digit classification our image data is encoded as integers in the range of 0–255. Before feding this data to the neural network we should normalize our values and cast them to float32 and then divide it by 255, so that they will be the floating point values in the range of 0–1.

We should not feed data into a neural network that takes relatively large values(Ex, multidigit integers, which are much larger than the initial values taken by the weights of the network) or data which is heterogenous (Ex, data where one feature is in the range of 0–1 and other feature in the range of 100–200) and because of this it will trigger large gradient updates and will prevent network from converging. To make learning easier for our network your data should have following characterstics :

- **Take Small Values**- Basically make most values in the range of [0,1] or [-1,1].
- **Be Homogenous**- Features should take values in roughly the same range.

Additionally, the following stricter normalization practice is common and can help, although it isn't always necessary (for example, you didn't do this in the digit-classification example):

- Normalize each feature independently to have a mean of 0
- Normalize each feature independently to have a standard deviation of 1.

```
x -= x.mean(axis=0)
x /= x.std(axis=0)
```

Four common normalization techniques may be useful:

- scaling to a range
- clipping
- log scaling
- z-score

I will write about them separately in different blog.

Handling Missing Values

Sometimes in our data we have missing values.

For example, in the house-price example, the first feature (the column of index 0 in the data) was the per capita crime rate. What if this feature wasn't available for all samples? You'd then have missing values in the training or test data.

Missing values could be: NaN, empty string, ?,-1,-99,-999 and so on. In order to understand if -1 is a missing value or not we could draw a histogram. If this variable has a uniform distribution between 0 and 1 and it has a small peak at -1 then -1 is actually a missing value.

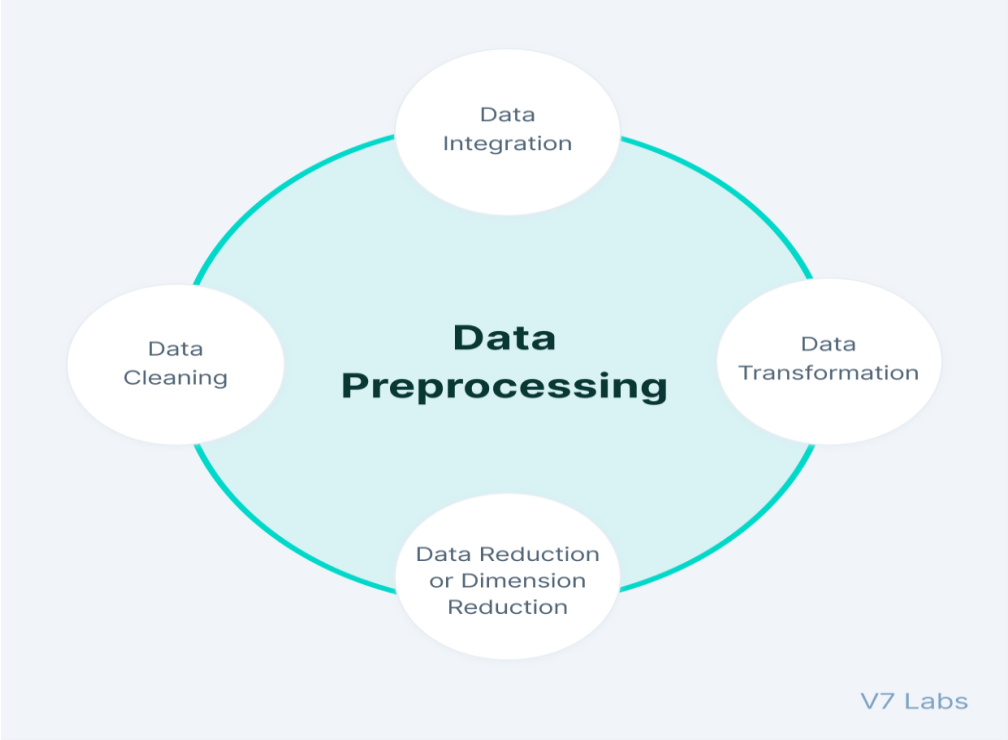
Missing values can be hidden from us and by hidden mean replaced by some other value beside NaN. Therefore, it is always beneficial to plot a **histogram** in order to identify those values.

In general, with neural networks, it's safe to input missing values as 0, with the condition that 0 isn't already a meaningful value. The network will learn from exposure to the data that the value 0 means missing data and will start ignoring the value.

Note that if you're expecting missing values in the test data, but the network was trained on data without any missing values, the network won't have learned to ignore missing values! In this situation, you should artificially generate training samples with missing entries: copy some training samples several times, and drop some of the features that you expect are likely to be missing in the test data.

Methods to handle missing values :

- *Ignore the data row*
- *Back-fill or forward-fill to propagate next or previous values respectively*
- *Replace with some constant value outside fixed value range-999,-1 etc*
- *Replace with mean, median value*
- *Isnull feature*



Feature Engineering

Feature Engineering means transforming raw data into a feature vector

In traditional programming, the focus is on code but in machine learning projects the focus shifts to representation. That is, one way developers hone a model is by adding and improving its features.

Feature engineering is the process of using your own knowledge about the data and about the machine-learning algorithms at hand to make the algorithm work better by applying hardcoded transformations to the data before it goes to the machine learning model.

In many cases it's not good to expect from machine learning model to learn from arbitary data. We need to present the data to the model which helps it to do model's job easier.

Before Deep Learning, feature engineering used to be difficult or critical because classical shallow algorithms in machine learning didn't have hypothesis spaces rich enough to learn useful features by themselves.

So, basically you have to remove the features from your data which are not relevant or contributing very much to your predictions and which helps you to reduce your hypothesis space and make your model's job easier to learn faster.

For instance, before convolutional neural networks became successful on the MNIST digit-classification problem, solutions were typically based on hardcoded features such as the number of loops in a digit image, the height of each digit in an image, a histogram of pixel values, and so on.

But, Modern Deep Learning removes the need for most feature engineering, because neural networks are capable of automatically extracting useful features from raw data.

Does this mean you don't have to worry about feature engineering as long as you're using deep neural networks? No, for two reasons:

- Good Features will help you to solve problem more better and easily using fewer computational resources.

- Good Features can help you solve the problem with far less data. The ability of deeplearning models to learn features on their own relies on having lots of training data available; if you have only a few samples, then the information value in their features becomes critical.