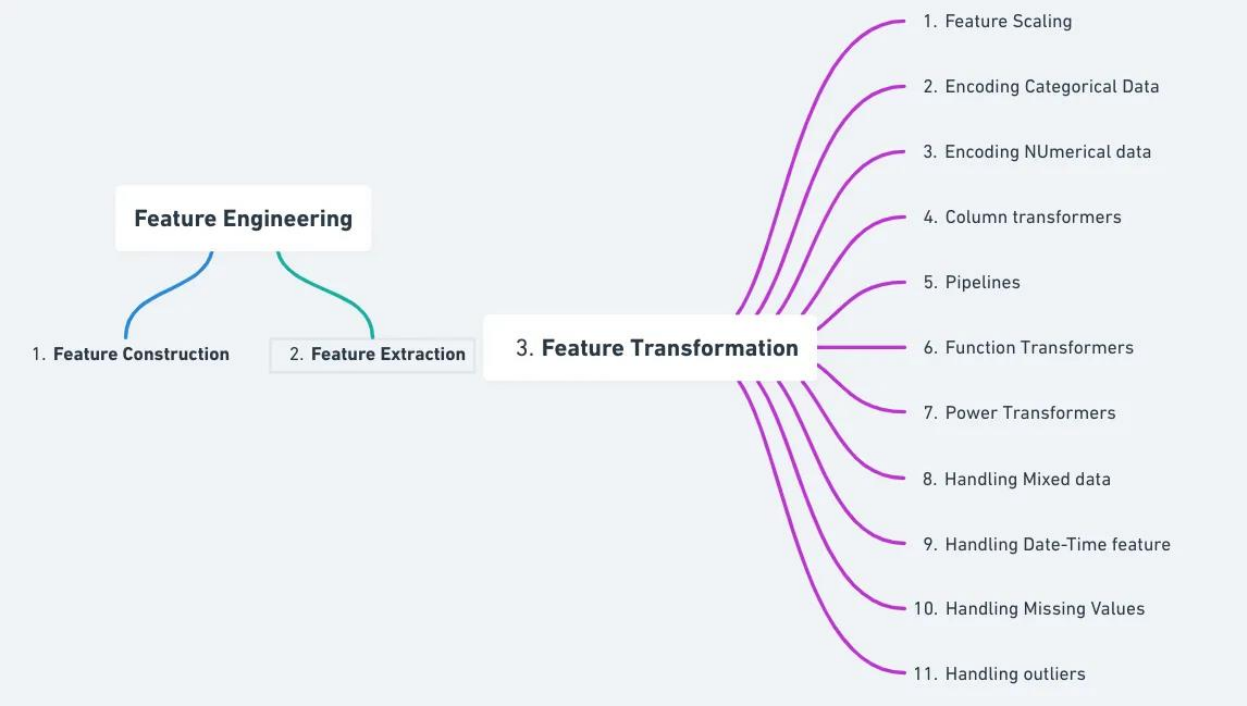
Feature Engineering



What is Feature Transformation?

Feature transformation involves converting, modifying, or restructuring existing features within your dataset to extract more meaningful information or to make them more suitable for machine learning algorithms. The goal is to maximize the predictive power of your models by transforming your data into a more informative and usable format.

Some of the Techniques in Feature Transformation

1. Feature Scaling: One of the fundamental techniques, feature scaling, deals with ensuring that all features are on a similar scale. Common methods include Min-Max scaling and Standardization (Z-score scaling).
2. Feature Construction: This technique involves creating new features by combining or modifying existing ones. For instance, in a house price prediction model, you might construct a new feature that represents the ratio of the number of bathrooms to the number of bedrooms, which can provide valuable information about the house.
3. Encoding Categorical Data: Categorical data, such as product categories or city names, needs to be converted into a numerical format for machine learning models to use. Techniques like one-hot encoding and label encoding are essential here.
4. Column Transformers and Pipelines: Column transformers allow you to apply different preprocessing steps to different subsets of your feature columns. Pipelines streamline the entire data preprocessing and modeling process.

Feature Engineering

5. Function Transformers: Function transformers involve applying custom functions to features, often used within pipeline workflows to perform specific data transformations.
6. Power Transformers: Techniques like Box-Cox or Yeo-Johnson transformations are used to make data more normally distributed, which can be beneficial for certain algorithms.
7. Handling Mixed Data: When your dataset contains both numerical and categorical data, techniques are required to manage both types appropriately.
8. Handling Date-Time Data: Date and time data can be challenging to work with, and specialized feature extraction and encoding techniques are necessary.
9. Handling Missing Values: Dealing with missing data is a crucial part of feature engineering. Techniques include imputation, removal, or creating indicators for missing values.
10. Handling Outliers: Outliers can skew model results, so it's important to detect and handle them appropriately through techniques like trimming, transformation, or removal.

Transformations (Simple Label Preserving, Perturbation, Data Synthesis)

Label-preserving Transformations

Before attempting to train a model using some augmentation pipeline, it's a good idea to invest some time in deciding on an appropriate set of transformations to choose from. Some of these transformations also have parameters to tune, and we should also make sure that we settle on a decent set of values for those.

What constitutes as "decent" depends on the dataset. In general we want the augmented images to be fairly dissimilar to the originals. However, we need to be careful that the augmented images still visually represent the same concept (and thus label). If a pipeline only produces output images that have this property we call this pipeline **label-preserving**.

Data Augmentation

These are set of techniques which increase the amount of data for training. They help in making our model robust as well. They are used a lot in deep learning problems, mainly NLP and Computer vision problems.

Few of these techniques are as follows:

A) **Simple Label-Preserving Transformations** - In a computer vision problem, we modify images by cropping, filtering, rotating, inverting it. This keeps the label same. In case of NLP we can simply replace a word with similar in meaning as it won't change the meaning or the sentiment of the sentence.

Feature Engineering

B) **Perturbation** - In this case we add noisy data samples to our training data which helps the model in identifying the weak spots in their decision boundary and improve performance. Noise is added either randomly or by search strategy. This type is called adversarial augmentation.

C) **Data Synthesis** - In this we synthesize the data. For NLP taska we can create templates and keep on filling different values in it. For example - "Find me a [CUISINE] restaurant within [NUMBER] miles of [LOCATION]." We can create different data points by putting different values in cuisines, number and location.

Feature scaling: normalization, standardization.

What is Feature Scaling?

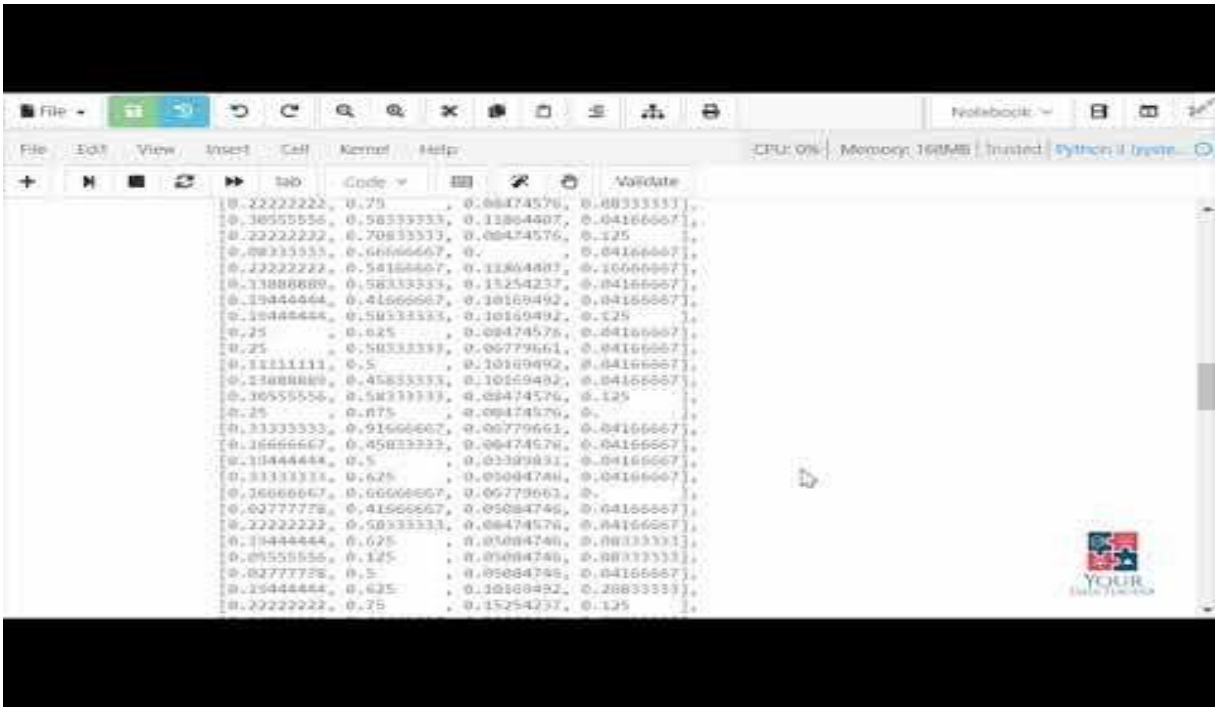
Feature scaling is a preprocessing technique that transforms feature values to a similar scale, ensuring all features contribute equally to the model. It's essential for datasets with features of varying ranges, units, or magnitudes. Common techniques include standardization, normalization, and min-max scaling. This process improves model performance, convergence, and prevents bias from features with larger values.

	Normalization	**Standardization**
1.	Minimum and maximum value of features are used for scaling	Mean and standard deviation is used for scaling.
2.	It is used when features are of different scales.	It is used when we want to ensure zero mean and unit standard deviation.
3.	Scales values between [0, 1] or [-1, 1].	It is not bounded to a certain range.
4.	It is really affected by outliers.	It is much less affected by outliers.
5.	Scikit-Learn provides a transformer called MinMaxScaler for Normalization.	Scikit-Learn provides a transformer called StandardScaler for standardization.
6.	This transformation squishes the n-dimensional data into an n-dimensional unit hypercube.	It translates the data to the mean vector of original data to the origin and squishes or expands.

Feature Engineering

7.	It is useful when we don't know about the distribution	It is useful when the feature distribution is Normal or Gaussian.
8.	It is a often called as Scaling Normalization	It is a often called as Z-Score Normalization.

Reference Link: https://www.youtube.com/watch?v=RyUQT7SqmyI



Link: https://jupyter.org/try-jupyter/tree/

Handling categorical variables: one-hot encoding, ordinal encoding. Word Embeddings vs One-Hot Encoding

Ordinal Encoding:

Feature Engineering

Encoding Categorical Data

There are three common approaches for converting ordinal and categorical variables to numerical values. They are:

- Ordinal Encoding
- One-Hot Encoding
- Dummy Variable Encoding

Let's take a closer look at each in turn.

Ordinal Encoding

In ordinal encoding, each unique category value is assigned an integer value.

For example, "*red*" is 1, "*green*" is 2, and "*blue*" is 3.

This is called an ordinal encoding or an integer encoding and is easily reversible. Often, integer values starting at zero are used.

For some variables, an ordinal encoding may be enough. The integer values have a natural ordered relationship between each other and machine learning algorithms may be able to understand and harness this relationship.

It is a natural encoding for ordinal variables. For categorical variables, it imposes an ordinal relationship where no such relationship may exist. This can cause problems and a one-hot encoding may be used instead.

This ordinal encoding transform is available in the scikit-learn Python machine learning library via the OrdinalEncoder class.

By default, it will assign integers to labels in the order that is observed in the data. If a specific order is desired, it can be specified via the "*categories*" argument as a list with the rank order of all expected labels.

We can demonstrate the usage of this class by converting colors categories "red", "green" and "blue" into integers. First the categories are sorted then numbers are applied. For strings, this means the labels are sorted alphabetically and that blue=0, green=1 and red=2.

The complete example is listed below.

Feature Engineering

```
[1]:
# example of a ordinal encoding
from numpy import asarray
from sklearn.preprocessing import OrdinalEncoder
# define data
data = asarray([['red'], ['green'], ['blue']])
print(data)
# define ordinal encoding
encoder = OrdinalEncoder()
# transform data
result = encoder.fit_transform(data)
print(result)

[['red']
 ['green']
 ['blue']]
[[2.]
 [1.]
 [0.]]

[ ]:
```

One-Hot Encoding

For categorical variables where no ordinal relationship exists, the integer encoding may not be enough, at best, or misleading to the model at worst.

In this case, a one-hot encoding can be applied to the ordinal representation. This is where the integer encoded variable is removed and one new binary variable is added for each unique integer value in the variable.

In the "*color*" variable example, there are three categories, and, therefore, three binary variables are needed. A "1" value is placed in the binary variable for the color and "0" values for the other colors.

This one-hot encoding transform is available in the scikit-learn Python machine learning library via the OneHotEncoder class.

We can demonstrate the usage of the OneHotEncoder on the color categories. First the categories are sorted, in this case alphabetically because they are strings, then binary variables are created for each category in turn. This means blue will be represented as [1, 0, 0] with a "1" in for the first binary variable, then green, then finally red.

The complete example is listed below.

```
# example of a one hot encoding
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder
# define data
data = asarray([['red'], ['green'], ['blue']])
print(data)
# define one hot encoding
encoder = OneHotEncoder(sparse=False)
# transform data
onehot = encoder.fit_transform(data)
```

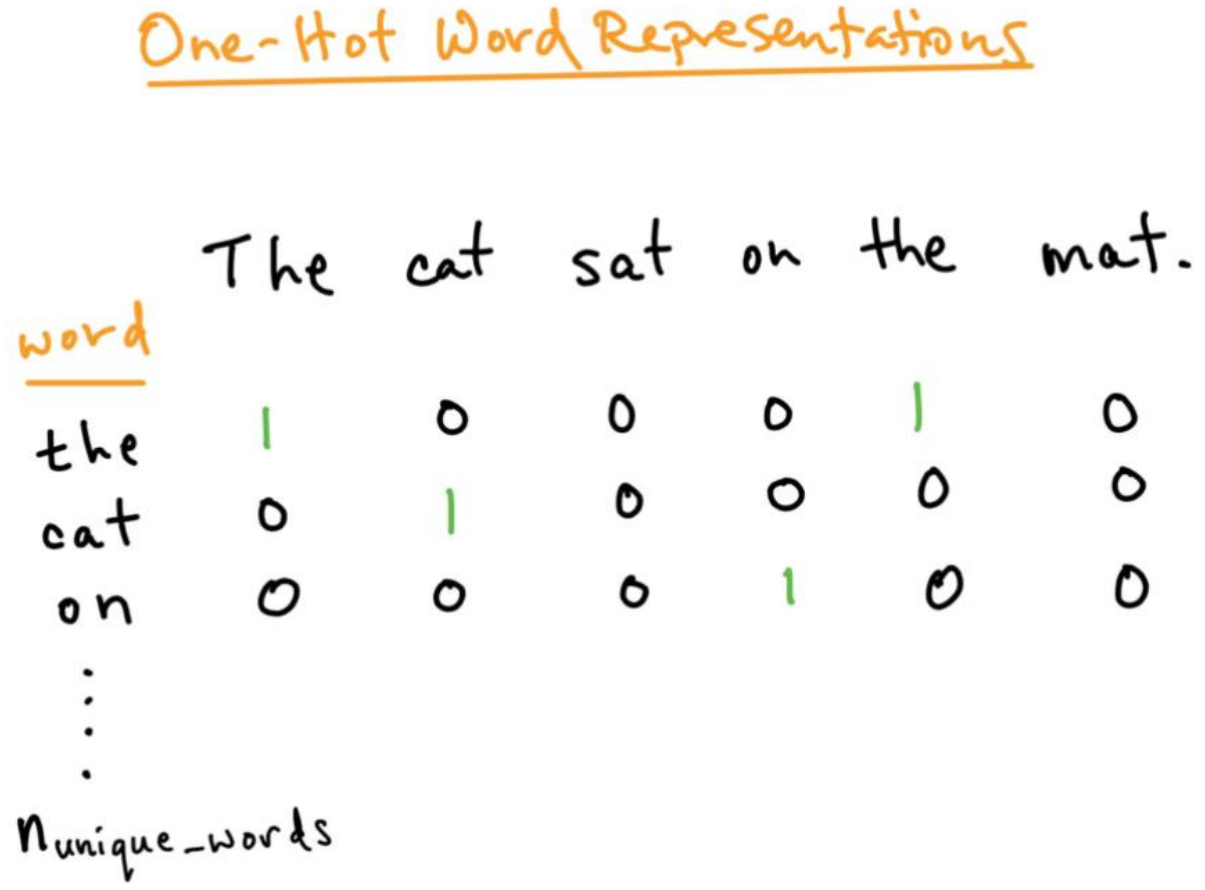
Feature Engineering

```
print(onehot)
```

```
1
0
1
1
```

```
# example of a one hot encoding
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder
# define data
data = asarray([['red'], ['green'], ['blue']])
print(data)
# define one hot encoding
encoder = OneHotEncoder(sparse=False)
# transform data
onehot = encoder.fit_transform(data)
print(onehot)
```

```
[['red']
 ['green']
 ['blue']]
/lib/python3.11/site-packages/sklearn/preprocessing/_encoders.py:975: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be r
emoved in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
[[0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]]
```



Try yourself:

Word Embeddings vs One-Hot Encoding

Feature Engineering

Feature Extraction with Principal Component Analysis (PCA)

PCA is a dimensionality reduction technique that has four main parts: feature covariance, eigendecomposition, principal component transformation, and choosing components in terms of explained variance.

Linear Discriminant Analysis (LDA) and/or Singular Value Decomposition (SVD) for dimensionality reduction

Nonlinear dimensionality reduction techniques such as kernel PCA and nonlinear manifold learning methods such as UMAP (Uniform Manifold Approximation and Projection) and t-SNE (t-Distributed Stochastic Neighbor Embedding)

Handling text data: text cleaning, tokenization, stemming, lemmatization.

Text preprocessing is an essential step in natural language processing (NLP) that involves cleaning and transforming unstructured text data to prepare it for analysis. It includes tokenization, stemming, lemmatization, stop-word removal, and part-of-speech tagging.

ocessing Task	Reasons
Noise Reduction	Text data often contains noise such as punctuation, special characters, and irrelevant symbols. Preprocessing helps remove these elements, making the text cleaner and easier to analyze.
Normalization	Different forms of words (e.g., “run,” “running,” “ran”) can convey the same meaning but appear in different forms. Preprocessing techniques like stemming and lemmatization help standardize these variations.
Tokenization	Text data needs to be broken down into smaller units, such as words or phrases, for analysis. Tokenization divides text into meaningful units, facilitating subsequent processing steps like feature extraction.
Stopword Removal	Stopwords are common words like “the,” “is,” and “and” that often occur frequently but convey little semantic meaning. Removing stopwords can improve the efficiency of text analysis by reducing noise.
Feature Extraction	Preprocessing can involve extracting features from text, such as word frequencies, n-grams, or word embeddings, which are essential for building machine learning models.

Feature Engineering

Dimensionality Reduction	Text data often has a high dimensionality due to the presence of a large vocabulary. Preprocessing techniques like term frequency-inverse document frequency (TF-IDF) or dimensionality reduction methods can help.

Feature selection techniques: filter methods, wrapper methods, embedded methods.
Handling High-Dimensional Data: Curse of dimensionality: challenges and techniques.

Azure AI Project: https://github.com/Santhoshstark06/Azure-Ai-Project