Feature Normalizations

1) Encoders : OrdinalEncoder and LabelEncoder

OrdinalEncoder and LabelEncoder are both preprocessing techniques used to encode categorical variables into numerical representations. However, they differ in their approach and use cases.

OrdinalEncoder preserves the ordinal relationship between categories by assigning them ordinal integer values,

while LabelEncoder simply assigns unique integer labels to each category without considering any order.

OrdinalEncoder vs LabelEncoder: Comparison

Aspect	OrdinalEncoder	LabelEncoder
Encoding Method	Encodes categorical features as ordinal integers	Encodes categorical features as unique integers
Handling of Ordinality	Preserves ordinal information	Does not preserve ordinal information
Suitable Data Types	Typically used for ordinal categorical features	Typically used for nominal categorical features
Example	Encoding temperature categories (cold, warm, hot)	Encoding color categories (red, blue, green)
Library	Supported in scikit-learn	Supported in scikit-learn

- **Encoding Method**:
  - OrdinalEncoder encodes categorical features as ordinal integers based on the order of the categories.
  - LabelEncoder assigns a unique integer to each category without considering any order.
- **Handling of Ordinality**:
  - OrdinalEncoder preserves the ordinal information present in the categorical features.
  - LabelEncoder does not consider the ordinality of the categories and treats them as nominal.
- **Suitable Data Types**:
  - OrdinalEncoder is suitable for encoding ordinal categorical features where the order matters, such as temperature categories (cold, warm, hot).
  - LabelEncoder is typically used for encoding nominal categorical features where there is no inherent order, such as color categories (red, blue, green).
- **Example**:
  - OrdinalEncoder can be used to encode temperature categories (cold, warm, hot) as 0, 1, 2, preserving their order.
  - LabelEncoder can be used to encode color categories (red, blue, green) as 0, 1, and 2, without considering any order.
- **Library**:
  - Both OrdinalEncoder and LabelEncoder are supported in scikit-learn, making them readily accessible for data preprocessing tasks.

Additional : terms from statistics

From Quantity to Labels: The Difference Between Cardinal, Ordinal, and Nominal Numbers

Difference Between Cardinal, Ordinal and Nominal Numbers

Parameters	Cardinal	Ordinal	Nominal
Definition	Represents a quantity or numeric value.	Represents a position or order in a sequence.	Represents a name or label for a category.
Operation	Arithmetic operation can be performed.	Arithmetic operations can't be performed but numbers can be compared, ranked and ordered.	Neither arithmetic operations nor comparison can be performed. These values can be grouped or classified.
Scale	Continuous	Discrete	Categorical
Application	Measure weight, volume or length	Ranking performance, rating satisfaction	Categorical data or labelling variable
Example	10 gram, 100-meter	1st place, 5th round, 3rd quarter	BMW car mode, red color

2) StandardScaler from scikit-learn

It guarantees that the features have a mean of 0 and a standard deviation of 1.

It standardizes the testing data (X_test) using the calculated mean and standard deviation from the training set.

```
# Standardize features

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Tf–idf, term weighting

In a large text corpus, some words will be very present (e.g. “the”, “a”, “is” in English) hence carrying very little meaningful information about the actual contents of the document. If we were to feed the direct count data directly to a classifier those very frequent terms would shadow the frequencies of rarer yet more interesting terms.

In order to re-weight the count features into floating point values suitable for usage by a classifier it is very common to use the tf–idf transform.

Tf means **term-frequency** while tf–idf means term-frequency times **inverse document-frequency**: $\mathbf{tf\text{-}idf(t,d)} = \mathbf{tf(t,d)} \times \mathbf{idf(t)}$.

Using the `TfidfTransformer`'s default settings, `TfidfTransformer(norm='l2', use_idf=True, smooth_idf=True, sublinear_tf=False)` the term frequency, the number of times a term occurs in a given document, is multiplied with idf component, which is computed as

$$\mathbf{idf}(t) = \log \frac{1+n}{1+\mathbf{df}(t)} + 1,$$

where n is the total number of documents in the document set, and $\mathbf{df}(t)$ is the number of documents in the document set that contain term t. The resulting tf-idf vectors are then normalized by the Euclidean norm:

$$v_{norm} = \frac{v}{\|v\|_2} = \frac{v}{\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}}.$$

This was originally a term weighting scheme developed for information retrieval (as a ranking function for search engines results) that has also found good use in document classification and clustering.

The following sections contain further explanations and examples that illustrate how the tf-idfs are computed exactly and how the tf-idfs computed in scikit-learn's `TfidfTransformer` and `TfidfVectorizer` differ slightly from the standard textbook notation that defines the idf as

$$\mathbf{idf}(t) = \log \frac{n}{1+\mathbf{df}(t)}.$$

In the `TfidfTransformer` and `TfidfVectorizer` with `smooth_idf=False`, the “1” count is added to the idf instead of the idf's denominator:

↑ Back to top

$$\mathbf{idf}(t) = \log \frac{n}{\mathbf{df}(t)} + 1$$

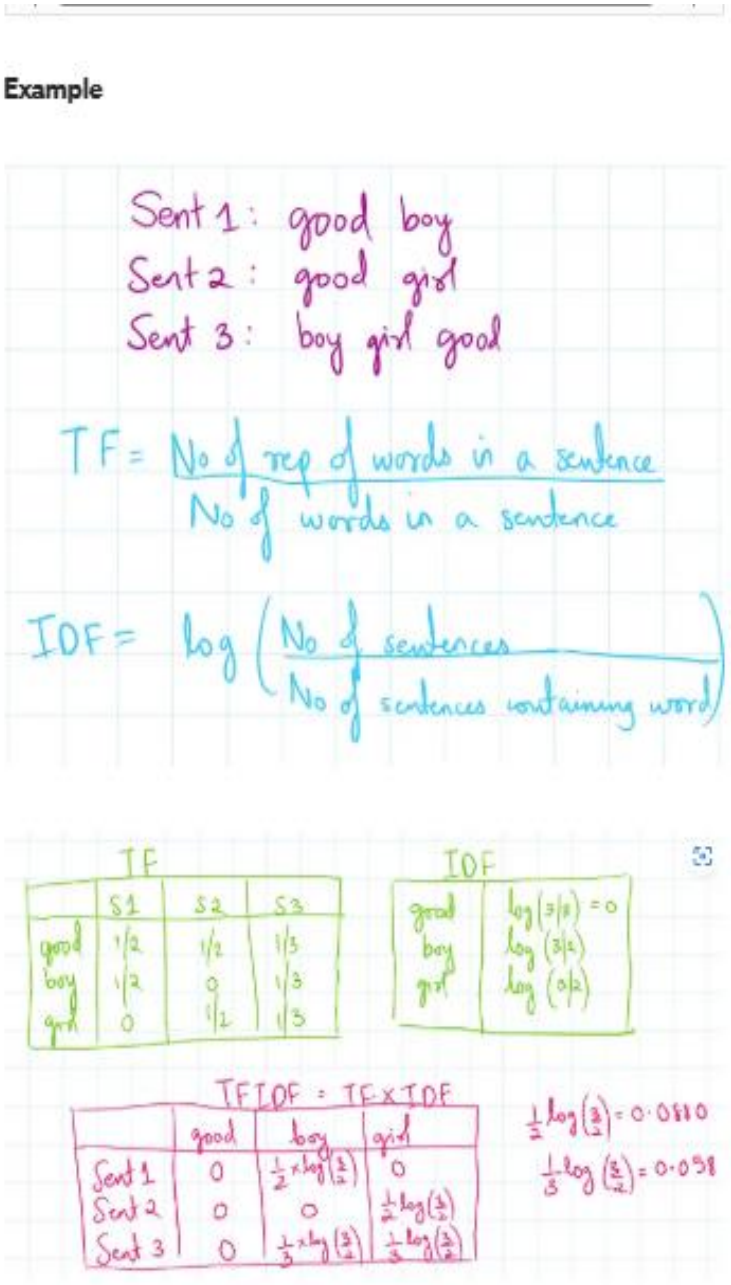
In the realm of Natural Language Processing (NLP), TF-IDF (Term Frequency-Inverse Document Frequency) is a powerful technique used to analyze and understand the importance of words in a document corpus. TF-IDF plays a crucial role in tasks such as text mining, information retrieval, and document classification. Let's delve into the concepts and applications of TF-IDF in NLP.

What is TF-IDF?

TF-IDF is a numerical statistic that reflects the significance of a word within a document relative to a collection of documents, known as a corpus. The idea behind TF-IDF is to quantify the importance of a term in a document with respect to its frequency in the document and its rarity across multiple documents.

Term Frequency (TF)

TF measures the frequency of a term within a document. It is calculated as the ratio of the number of times a term occurs in a document to the total number of terms in that document. The goal is to emphasize words that are frequent within a document.



$$TF(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

Inverse Document Frequency (IDF)

IDF measures the rarity of a term across a collection of documents. It is calculated as the logarithm of the ratio of the total number of documents to the number of documents containing the term. The goal is to penalize words that are common across all documents.

$$IDF(t,D) = \log\left(\frac{\text{Total number of documents in the corpus } N}{\text{Number of documents containing term } t}\right)$$

Combining TF and IDF: TF-IDF

The TF-IDF score for a term in a document is obtained by multiplying its TF and IDF scores.

$$\text{TF-IDF}(t,d,D) = TF(t,d) \times IDF(t,D)$$

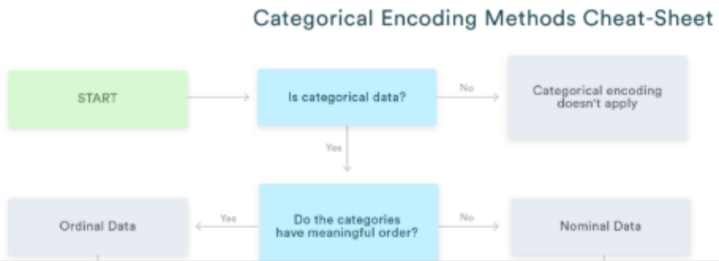
Sample code at LAB 3

NOTE

- The theoretical value and the actual value is being different by a significant amount.
- It is because inside of the scikit library the formula that is being used for IDF is

- For each term, calculate the inverse document frequency across the entire corpus.
- $$IDF(t, D) = \log\left(\frac{\text{Total number of documents in the corpus } N}{\text{Number of documents containing term } t}\right) + 1$$

Categorical Encoding Methods Cheat-Sheet



https://machinelearningmastery.com/tips-for-effective-feature-engineering-in-machine-learning/?utm_source=drip&utm_medium=email&utm_campai...

Create More Representative Features with Binning

Binning features into buckets involves grouping continuous variables into discrete intervals. Sometimes grouping features like age and income into bins can help find patterns that are hard to identify within continuous data.

For the example housing dataset, let's bin the age of the house and income of the household into different bins with descriptive labels. You can use the cut() function in pandas to bin features into equal-width intervals like so:

```
# Creating income bins
X_train['age_bin'] = pd.cut(X_train['age'], bins=3, labels=['new', 'moderate', 'old'])
X_test['age_bin'] = pd.cut(X_test['age'], bins=3, labels=['new', 'moderate', 'old'])

# Creating income bins
X_train['income_bin'] = pd.cut(X_train['income'], q=4, labels=['low', 'medium', 'high', 'very
X_test['income_bin'] = pd.cut(X_test['income'], q=4, labels=['low', 'medium', 'high', 'very_h
```

Binning continuous features into discrete intervals can simplify the representation of continuous variables as features with more predictive power.