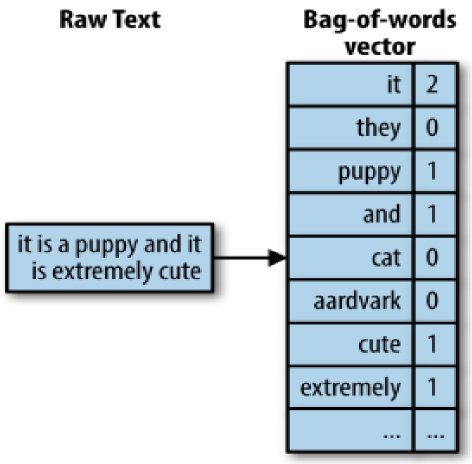
Unit-3: Features of Text and Categorical Data

Bag-of-Words methods to represent text and its drawbacks:

Natural language text can be transformed into a flat vector for use in machine learning models. To build effective models, it's important to use simple, interpretable features that improve accuracy. A common technique is the Bag-of-Words (BoW) model, where text is represented as a list of word counts.

In information retrieval, the goal is to retrieve documents that are most relevant to a given query. For example, in document classification, texts can be categorized into topics like sports (with words like "player," "coach," "match") or politics (with words like "leader," "speech," "media"). The text is converted into a vector where each element corresponds to the frequency of a word from the vocabulary.

A vector is essentially a collection of numbers, with each entry representing the count of a specific word. For instance, if the word "it" appears twice in a document, the corresponding position in the vector holds a value of 2. Words that do not appear in the document are assigned a count of 0. This simple representation captures the frequency of words while discarding grammar and word order, making it widely used for tasks like text classification and document retrieval.



Drawbacks:

- Does not consider the sequence of data
  example: "Cat jumped over rat" and "Rat jumped over cat".
- Does not consider any concept of word hierarchy
  example: 'cat', 'rat' and 'raven' are animal
- Size of vector representation

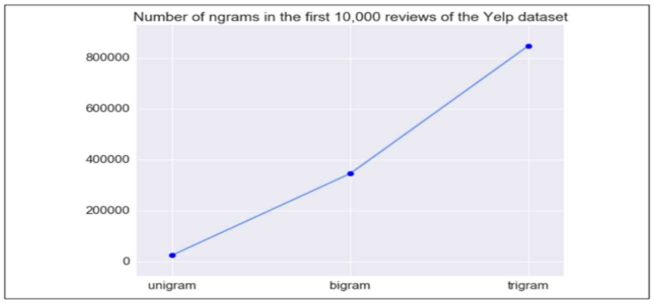
Consider a vocabulary of size 10K words. Each document representation is of size 10K size vector

- No Semantic Understanding

The Bag-of-Words method is a simple, interpretable approach to representing text, making it a good starting point for many text analysis tasks. However, its limitations, such as ignoring word order and semantics, make it less suitable for tasks where deeper linguistic understanding is needed. More advanced techniques, like TF-IDF or word embeddings, are often preferred when context and semantic relationships are crucial.

Bag- bag-of-n-grams and its drawbacks:

Bag-of-n-grams is a natural extension of the Bag-of-Words model that captures sequences of tokens, rather than individual words. An **n-gram** refers to a sequence of n tokens, where a 1-gram (unigram) represents individual words, and higher-order n-grams capture larger word sequences.
After tokenization, the counting mechanism can either count individual words (unigrams) or count overlapping sequences of n tokens as n-grams. For example, the sentence "Emma knocked on the door" generates the following 2-grams: "Emma knocked," "knocked on," "on the," and "the door."
Unlike Bag-of-Words, n-grams retain some of the word order and context from the original text, making the **Bag-of-n-grams** representation more informative by preserving part of the sequence structure.
However, n-grams are computationally more expensive to calculate, store, and model. The higher the value of n, the more context the representation captures, but at the cost of increased complexity. For instance, with k unique words in the vocabulary, there could be up to k^2 unique 2-grams, leading to a much larger and sparser feature space compared to unigrams.



Number of unique n-grams in the first 10,000 reviews of the Yelp dataset

In short, while **Bag-of-n-grams** improves the richness of the text representation by including word sequences, it also introduces a trade-off between capturing more information and managing the associated computational and storage costs.

Drawbacks of Bag-of-n-grams

1. **Exponential Growth of Feature Space**:
   - The number of n-grams grows exponentially as n increases. For example, if a corpus has **k** unique words, the possible number of 2-grams (bigrams) is $\mathbf{k^2}$. This leads to a large and sparse feature space, which increases computational complexity, memory requirements, and the risk of overfitting.
2. **Computational and Storage Costs**:
   - Storing and processing large numbers of n-grams, especially higher-order ones (e.g., 3-grams or 4-grams), requires significant computational resources. This can become impractical for very large corpora or high-dimensional n-gram models.
3. **Data Sparsity**:
   - Many n-grams will not appear in every document, leading to sparse vectors with many zero values. The larger the value of n, the more likely the vectors will be sparse, which can reduce model efficiency and increase the risk of overfitting to the training data.

The **Bag-of-n-grams** method offers a simple and effective way to represent text while preserving some word order and contextual relationships. It is particularly useful in NLP tasks that benefit from capturing short-term dependencies between words, such as text classification, information retrieval, and sentiment analysis. However, the method comes with significant drawbacks. The exponential growth in the number of features, increased computational demands, and sparse data representation can limit its scalability and generalization ability. For large and complex datasets, more advanced techniques, like **TF-IDF weighting**, **word embeddings** (e.g., Word2Vec, GloVe), or **transformer models** (e.g., BERT, GPT), are often better alternatives to handle long-term dependencies and semantic relationships in text.

Text Representation Example:

Text => “I like puppy. I like cat. I like horse.”

Vocabulary

I, like, puppy, cat, horse

bag-of-word vector

	I	Like	Puppy	Cat	horse
I like puppy	1	1	1	0	0
I like cat	1	1	0	1	0
I like horse	1	1	0	0	1

Unigrams (single words):

- "I", "like", "puppy", "cat", "horse"

Bigrams (pairs of words):

- "I like", "like puppy", "I like", "like cat", "I like", "like horse"

Trigrams (three-word sequences):

- "I like puppy", "I like cat", "I like horse"

Definitions:

a. Stemming b. Parsing c. Tokenization.

a. Stemming

Stemming is the process of reducing words to their base or root form, known as the **stem**. This is achieved by removing suffixes and prefixes, which allows different forms of a word to be treated as the same. For example, the words "running," "runner," "ran," and "runs" can all be reduced to the stem "run."

Purpose:

- Stemming helps simplify text analysis by consolidating similar words into a common form, reducing dimensionality in the data.

Example:

- Words like "studies," "studying," and "studied" may all be stemmed to "study."

Common Algorithms:

- **Porter Stemmer**: A widely used algorithm that applies a set of rules to transform words.
- **Snowball Stemmer**: An improved version of the Porter Stemmer that supports multiple languages.

b. Parsing

Parsing refers to the process of analyzing a string of symbols (like text) in order to extract meaningful information from it. In the context of natural language processing, parsing typically involves breaking down sentences into their grammatical components, such as identifying parts of speech (nouns, verbs, adjectives) and the relationships between them.

Purpose:

- Parsing helps understand the grammatical structure of sentences, which is essential for tasks like sentiment analysis, machine translation, and question answering.

Example:

- Given the sentence "The cat sat on the mat," a parser would identify "The" as a determiner, "cat" as a noun, "sat" as a verb, and "on the mat" as a prepositional phrase.

Types of Parsing:

- **Dependency Parsing**: Analyzes the dependencies between words in a sentence.
- **Constituency Parsing**: Breaks down a sentence into sub-phrases (or constituents) based on grammar rules.

c. Tokenization

Tokenization is the process of breaking down a text into smaller units called **tokens**, which can be words, phrases, or symbols. This step is crucial for preparing text for analysis, as it transforms raw text into a structured format that can be processed by algorithms.

Purpose:

- Tokenization facilitates the extraction of meaningful information from text by allowing algorithms to work with smaller, manageable pieces of the text.

Example:

1 For the sentence "I love natural language processing," tokenization would break it down into the following tokens: ["I", "love", "natural", "language", "processing"].

Types of Tokenization:

- **Word Tokenization**: Splits text into words based on spaces and punctuation.
- **Sentence Tokenization**: Divides text into sentences, often using punctuation marks to identify sentence boundaries.

Together, these techniques form an essential part of text cleaning and preprocessing in natural language processing, enabling more accurate and meaningful analysis of textual data.

Handling frequent and rare words in Text Data:

Handling Frequent Words

1. Stop Words Removal:

Definition: Stop words are common words that often carry little meaning and are typically filtered out before processing. Examples include "the," "is," "in," "and," etc.

Treatment: Remove stop words from the text to reduce noise and focus on more informative words. This can help improve model performance by reducing dimensionality and computation.

2. Term Frequency-Inverse Document Frequency (TF-IDF):

Definition: TF-IDF is a statistical measure used to evaluate how important a word is to a document in a collection or corpus.

Treatment: Frequent words across documents might have high term frequency but low inverse document frequency. TF-IDF can help down-weight these common words and prioritize more informative terms that appear less frequently.

3. Feature Selection:

Definition: Identifying and selecting relevant features (words) for the model.

Treatment: Use feature selection techniques to identify and retain only those frequent words that contribute significantly to the outcome. For instance, using techniques like Chi-Square or mutual information to evaluate the importance of frequent words in relation to the target variable.

4. Dimensionality Reduction:

Definition: Techniques used to reduce the number of features in a dataset.

Treatment: Apply dimensionality reduction methods (e.g., Principal Component Analysis, Latent Semantic Analysis) to reduce the impact of frequent words while preserving the overall structure of the data.

Handling frequent and rare words

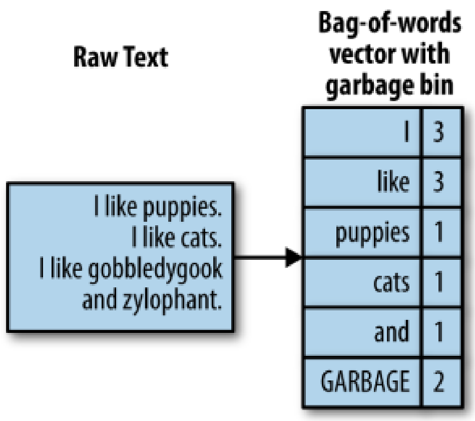
Depending on the task at hand, it may be necessary to filter out rare words. These words can be either obscure terms or misspellings of common words. For statistical models, a word that appears in only one or two documents is often considered noise rather than valuable information.

For instance, in the Yelp dataset, the term "gobbledygook" appears in only a single review. Using such a rare word for business identification would likely lead to errors in machine learning models. Consequently, rare words are generally unreliable as predictors and can introduce unnecessary computational overhead.

In the set of 1.6 million Yelp reviews, there are 357,481 unique words (tokenized based on spaces and punctuation). Notably, 189,915 of these words appear in only one review, while 41,162 occur in two reviews. This means that over 60% of the vocabulary consists of words that are rarely used, which is a common scenario in real-world data.

Moreover, the training time for many statistical machine learning models scales linearly with the number of features, and some models exhibit quadratic or worse scaling behavior. Therefore, it is prudent to trim the vocabulary based on word count statistics to enhance model efficiency and accuracy.

The rare words lose their identity and get grouped into a garbage bin feature



Collocation Extraction using statistical technique:

A sequence of tokens immediately yields the list of words and n-grams. We are more used to understanding phrases, not n-grams. In computational natural language processing (NLP), the concept of a useful phrase is called a collocation. A collocation is an expression consisting of two or more words that correspond to some conventional way of saying things. A collocation a combination of words in a language, that happens very often and more frequently than would happen by chance. Collocation is 'a predictable combination of words' for example we can say heavy rain but not strong rain because it does not sound right' likewise, we can say 'do exercise' but not 'make exercise'. Collocations can be made up of any kinds of words such as verbs, nouns, adverbs and adjectives. Collocations is a word or phrase that is often used with another word or phrase, in a way that sounds correct to people who have spoken the language all their lives, but might not be expected from the meaning

E.g. “Strong Tea” - > “great physical strength” and “tea” => collocation

“New York”, “Times of India” => collocation

“Cute Puppy” => no collocation

Collocations are combinations of words or terms that often appear together in a natural and frequently occurring pattern within a language. These word pairings or groupings have a strong tendency to co-occur due to linguistic convention and native speaker usage. Collocations can involve two or more words and are an essential aspect of language fluency and understanding.

Collocations can be classified into various categories:

Verb-Noun Collocations: Examples include "make a decision," "take a shower," and "do homework." These combinations are considered common and natural in English.

Adjective-Noun Collocations: These are pairings of adjectives with specific nouns, like "strong coffee," "heavy rain," or "fast car." They describe characteristics or qualities associated with the noun.

Noun-Noun Collocations: These are combinations of two nouns, such as "business meeting," "ice cream," and "computer science." They often represent compound nouns or concepts.

Adverb-Adjective Collocations: Examples include "very happy," "extremely tired," and "incredibly beautiful." These combinations emphasize the degree or intensity of the adjective.

Verb-Adverb Collocations: These involve verbs and adverbs, like "run quickly," "sing loudly," and "eat slowly." Adverbs modify the action described by the verb.

Prepositional Collocations: These consist of a verb, adjective, or noun followed by a specific preposition. Examples include "interested in," "famous for," and "dependent on."

Collocations do not have to be consecutive sequences. Because collocations are more than the sum of their parts, their meaning cannot be adequately captured by individual word counts.

Bag-of-words falls short as a representation. Bag-of-n-grams is also problematic because it captures too many meaning-less sequences Collocations are useful as features. But how to identify these from text? Find comprehensive lists of idioms in various languages, and we could look through the text for any matches. It would be very expensive, but it would work. But does not work with domain specific text. Can use the statistical method to identify the collocation?

Hypothesis testing for collocation extraction:

Whether two words appear together more often than they would by chance. For a given pair of words, the method tests two hypotheses on the observed dataset.

Hypothesis 1 (the null hypothesis) says that word 1 appears independently from word 2.

Hypothesis 2 (the alternate hypothesis) says that seeing word 1 changes the likelihood of seeing word 2.

The final statistic is the log of the ratio between the two

$$\text{Log } \lambda = \log\left(\frac{L(\text{data, Hnull})}{L(\text{data, Halternative})}\right)$$

The likelihood function L(Data; H) represents the probability of seeing the word frequencies in the dataset under the independent or the not independent model for the word pair.

In order to compute this probability, we have to make another assumption about how the data is generated. The simplest data generation model is the binomial model, where for each word in the dataset, we toss a coin, and we insert our special word if the coin comes up heads, and some other word otherwise. The algorithm for detecting common phrases through likelihood ratio test analysis proceeds as follows:

Algorithm

1. Compute occurrence probabilities for all singleton words: P(w).
2. Compute conditional pairwise word occurrence probabilities for all unique bigrams: P(w2|w1).
3. Compute the likelihood ratio log λ for all unique bigrams.
4. Sort the bigrams based on their likelihood ratio.
5. Take the bigrams with the smallest likelihood ratio values as features.

Term frequency-inverse document frequency (TF-IDF) method for text representation:

Term Frequency-Inverse Document Frequency (TF-IDF) is a statistical measure used to evaluate the importance of a word in a document relative to a collection (or corpus) of documents. The TF-IDF method combines two components:

Term Frequency (TF): Measures how frequently a term appears in a document.

Inverse Document Frequency (IDF): Measures how important a term is across the entire corpus, reducing the weight of common terms while increasing the weight of rare terms.

TF is calculated as:

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

IDF is calculated as:

$$\mathrm{IDF}(t,D) = \log\left(\frac{\text{Total number of documents } |D|}{\text{Number of documents containing term } t}\right)$$

TF-IDF is then calculated as:

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

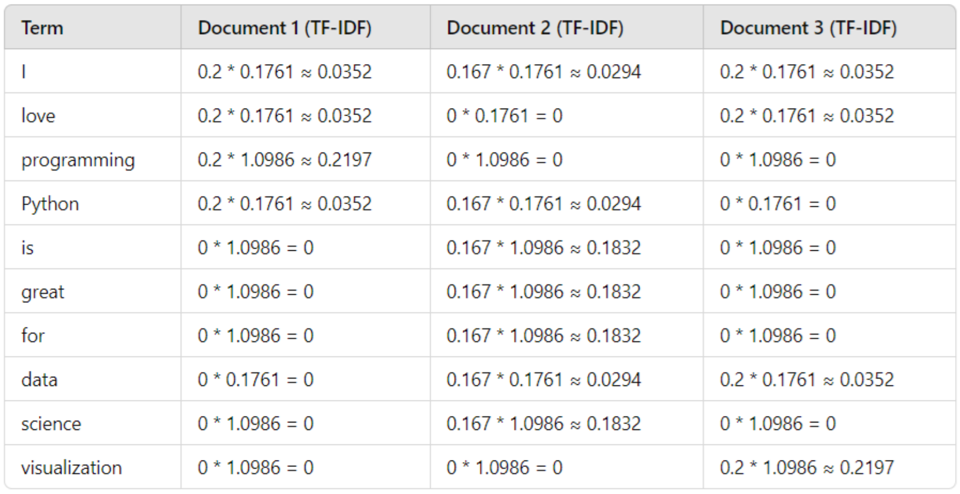
<u>**Example**</u>

Let's consider a small corpus of three documents:

Document 1: "I love programming in Python"

Document 2: "Python is great for data science"

Document 3: "I love data visualization"



Term	Document 1 (TF-IDF)	Document 2 (TF-IDF)	Document 3 (TF-IDF)
I	0.2 * 0.1761 ≈ 0.0352	0.167 * 0.1761 ≈ 0.0294	0.2 * 0.1761 ≈ 0.0352
love	0.2 * 0.1761 ≈ 0.0352	0 * 0.1761 = 0	0.2 * 0.1761 ≈ 0.0352
programming	0.2 * 1.0986 ≈ 0.2197	0 * 1.0986 = 0	0 * 1.0986 = 0
Python	0.2 * 0.1761 ≈ 0.0352	0.167 * 0.1761 ≈ 0.0294	0 * 0.1761 = 0
is	0 * 1.0986 = 0	0.167 * 1.0986 ≈ 0.1832	0 * 1.0986 = 0
great	0 * 1.0986 = 0	0.167 * 1.0986 ≈ 0.1832	0 * 1.0986 = 0
for	0 * 1.0986 = 0	0.167 * 1.0986 ≈ 0.1832	0 * 1.0986 = 0
data	0 * 0.1761 = 0	0.167 * 0.1761 ≈ 0.0294	0.2 * 0.1761 ≈ 0.0352
science	0 * 1.0986 = 0	0.167 * 1.0986 ≈ 0.1832	0 * 1.0986 = 0
visualization	0 * 1.0986 = 0	0 * 1.0986 = 0	0.2 * 1.0986 ≈ 0.2197

Advantages and disadvantages TF-IDF method:

Term Frequency-Inverse Document Frequency (TF-IDF) is a popular method used in natural language processing (NLP) for text representation, particularly in information retrieval and text mining. It reflects the importance of a word in a document relative to its occurrence in a collection of documents (corpus). Here are its key advantages and disadvantages:

Advantages

1. **Weighted Term Importance:** TF-IDF assigns a weight to each term in a document, reflecting its importance within that document and the entire corpus. This allows for more precise representation of document content compared to simple term frequency (TF) or binary presence.
2. **Term Discrimination:** TF-IDF helps discriminate between commonly occurring terms and rare terms. Common words like "the," "and," and "is" have high term frequency but low IDF, making them less important in the overall ranking of documents. Rare and specific terms, on the other hand, tend to have higher importance.
3. **Reduction of Noise:** By reducing the weight of common terms, TF-IDF can help in reducing noise in text data, focusing on the more meaningful and distinctive terms. This is especially useful in information retrieval tasks where you want to find relevant documents.
4. **Language Independence:** TF-IDF is language-independent, meaning it can be applied to texts in any language as long as the corpus is appropriately prepared. This makes it a versatile technique for text analysis and retrieval.
5. **Easy to Implement:** The calculation of TF-IDF scores is relatively straightforward and computationally efficient, making it practical for both small and large text datasets.

Disadvantages

1. **Lack of Context:** TF-IDF doesn't take into account the context or semantic meaning of words. It treats all terms independently, which can result in less accurate representations of documents, especially for languages with complex semantics.
2. **Loss of Word Order:** TF-IDF doesn't consider the order of words in a document, which means it loses information about the syntactic and grammatical structure of the text. For some applications like natural language understanding, this is a significant limitation.
3. **Sensitivity to Document Length:** Documents of different lengths can produce varying TF-IDF scores. Longer documents may have higher overall term frequencies, potentially biasing the importance of terms. This sensitivity can be mitigated by using normalization techniques.
4. **Limited Handling of Synonyms:** TF-IDF doesn't inherently handle synonyms well. Terms with similar meanings may not be recognized as such, leading to less accurate representation and retrieval results. Techniques like word embeddings can address this issue.
5. **Requires a Representative Corpus:** To calculate IDF accurately, you need a representative corpus that covers the domain of your documents. In some cases, obtaining or maintaining such a corpus can be challenging.

Methods for encoding categorical variables:

Encoding categorical variables is an essential preprocessing step in machine learning when dealing with categorical data (i.e., data that contains labels or categories instead of numerical values). Since most machine learning models require numerical inputs, categorical variables need to be transformed into a numerical format.

1. **One-Hot Encoding:**
   - **Description:** Converts each category value into a new binary column (feature). If a categorical variable has 'n' unique categories, one-hot encoding creates 'n' new features.
   - **How It Works:**
     - For each category, create a new binary column.
     - Mark '1' in the column corresponding to the category present in the observation, and '0' in all other columns.
   - **Example:**
     - Categories: "Red", "Green", "Blue".
     - One-Hot Encoded Features:
       - "Red": [1, 0, 0]
       - "Green": [0, 1, 0]
       - "Blue": [0, 0, 1]
   - **Pros:**
     - Simple to implement.
     - Does not assume any order among categories.

- **Cons:**
    - Can create a large number of features if the categorical variable has many unique values (high cardinality).
    - May lead to the "curse of dimensionality," making the model complex and slow.

2. **Label Encoding:**
    - **Description:** Assigns a unique integer to each category.
    - **How It Works:**
        - Categories are mapped to integers starting from 0 or 1.
    - **Example:**
        - Categories: "Red", "Green", "Blue".
        - Label Encoded Values:
            - "Red": 1
            - "Green": 2
            - "Blue": 3
    - **Pros:**
        - Efficient and straightforward.
        - Does not increase the number of features.
    - **Cons:**
        - Imposes an ordinal relationship where none may exist (the model may interpret "Blue" > "Green" > "Red").
        - Not suitable for nominal categories without an inherent order.
3. **Ordinal Encoding:**
    - **Description:** Similar to label encoding but specifically used when categories have a natural order.
    - **How It Works:**
        - Assign integers to categories based on their rank or order.
    - **Example:**
        - Categories: "Low", "Medium", "High".
        - Ordinal Encoded Values:
            - "Low": 1
            - "Medium": 2
            - "High": 3
    - **Pros:**
        - Preserves the ordinal relationship among categories.
    - **Cons:**
        - Not suitable for nominal data without a clear order.
4. **Dummy Encoding:**
    - **Description:** Similar to one-hot encoding but drops one category to avoid redundancy.
    - **How It Works:**
        - For 'n' categories, create 'n-1' binary features.
        - The dropped category is inferred when all other features are '0'.
    - **Example:**
        - Categories: "Red", "Green", "Blue".
        - Dummy Encoded Features (dropping "Blue"):
            - "Red": [1, 0]
            - "Green": [0, 1]
            - "Blue": [0, 0]
    - **Pros:**

- Reduces multicollinearity in linear models.
- Slightly reduces dimensionality compared to one-hot encoding.

- **Cons:**
  - Still increases the number of features, which can be problematic with high-cardinality variables.

5. **Effect Coding (Deviation Coding):**
   - **Description:** Encodes categories by comparing them to the overall mean.
   - **How It Works:**
     - Assigns values of -1, 0, and 1 to represent the categories.
     - One category is chosen as the reference (baseline) and is represented by -1 across all effect-coded features.
   - **Example:**
     - Categories: "Red", "Green", "Blue" (reference category).
     - Effect Encoded Features:
       - "Red": [1, 0]
       - "Green": [0, 1]
       - "Blue": [-1, -1]
   - **Pros:**
     - Useful in statistical modeling to understand the effect of each category relative to the overall mean.
   - **Cons:**
     - More complex to interpret.
     - Less common in machine learning contexts.
6. **Binary Encoding:**
   - **Description:** Combines the benefits of high-dimensionality reduction and preserving uniqueness of categories.
   - **How It Works:**
     - Categories are first label encoded.
     - Label numbers are converted into binary code.
     - Each digit in the binary code becomes a new feature.
   - **Example:**
     - Categories: "Red"=1, "Green"=2, "Blue"=3.
     - Binary Codes:
       - "Red" (1): [0, 1]
       - "Green" (2): [1, 0]
       - "Blue" (3): [1, 1]
   - **Pros:**
     - Reduces the dimensionality compared to one-hot encoding.
     - Handles high-cardinality variables efficiently.
   - **Cons:**
     - The relationship between categories may become less interpretable.
     - Possible loss of information due to binary representation.
7. **Hash Encoding (Feature Hashing):**
   - **Description:** Uses a hash function to map categories to a fixed number of features.
   - **How It Works:**
     - Categories are hashed into a specified number of bins (features).
     - Multiple categories may map to the same bin (collision).

- **Example:**
    - Categories hashed into 2 features:
        - "Red": Hash("Red") → Feature 1
        - "Green": Hash("Green") → Feature 2
        - "Blue": Hash("Blue") → Feature 1
- **Pros:**
    - Efficient for handling very high-cardinality features.
    - Controls the number of features explicitly.
- **Cons:**
    - Collisions can introduce ambiguity.
    - Not invertible; original categories cannot be retrieved from the hash.

8. **Target Encoding (Mean Encoding):**
    - **Description:** Replaces categories with the average value of the target variable for that category.
    - **How It Works:**
        - For each category, calculate the mean of the target variable.
        - Replace category with this mean value.
    - **Example:**
        - Suppose the target is "Purchase Amount."
        - Categories:
            - "Red": Average Purchase Amount = $100
            - "Green": Average Purchase Amount = $150
            - "Blue": Average Purchase Amount = $200
        - Encoded Values:
            - "Red": 100
            - "Green": 150
            - "Blue": 200
    - **Pros:**
        - Captures the effect of categories on the target variable.
        - Reduces dimensionality.
    - **Cons:**
        - High risk of overfitting, especially with rare categories.
        - Requires careful cross-validation to prevent data leakage.
9. **Frequency Encoding:**
    - **Description:** Replaces categories with their frequency (count) in the dataset.
    - **How It Works:**
        - Count the occurrences of each category.
        - Replace the category with its count or proportion.
    - **Example:**
        - Categories:
            - "Red": Appears 50 times
            - "Green": Appears 30 times
            - "Blue": Appears 20 times
        - Encoded Values:
            - "Red": 50
            - "Green": 30
            - "Blue": 20

- **Pros:**
  - Simple to implement.
  - Preserves information about the rarity or commonness of categories.
- **Cons:**
  - May not capture the relationship with the target variable.
  - Frequencies might not be meaningful if the dataset is unbalanced.

Choosing the right encoding technique depends on the nature of the categorical variable, the machine learning algorithm, and the size of the dataset.

Encoding Examples:

One-Hot Encoding:

- Iris Setosa: [1, 0, 0]

- Iris Versicolor: [0, 1, 0]

- Iris Virginica: [0, 0, 1]

Dummy Coding:

Assume Iris Setosa is the reference category:

- Iris Versicolor: [1, 0]

- Iris Virginica: [0, 1]

Effect Coding:

Assume Setosa is the reference:

- Iris Versicolor: [1, -1]

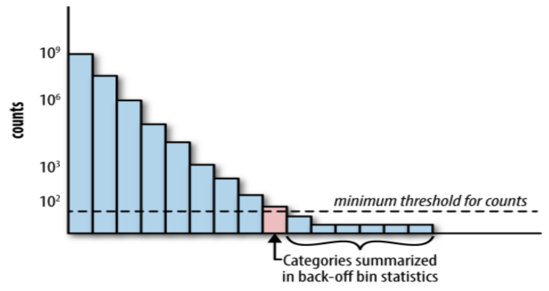
- Iris Virginica: [-1, 1]

Methods used for handling frequent and rare categories from the dataset:

Handling large categorical variables and rare categories is a crucial aspect of data preprocessing, particularly when dealing with high-cardinality data. Poorly managed rare categories or large categorical variables can lead to overfitting, model inefficiencies, and degraded model performance. Below are various strategies for handling these challenges.

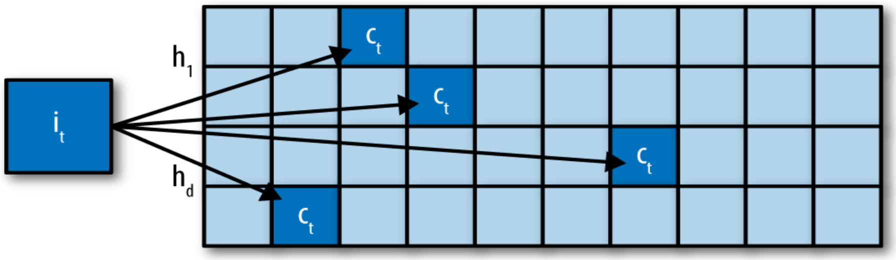
Back-off Technique

One way to deal with this is through *back-off*, a simple technique that accumulates the counts of all rare categories in a special bin. If the count is greater than a certain threshold, then the category gets its own count statistics. Otherwise, we use the statistics from the back-off bin. This essentially reverts the statistics for a single rare category to the statistics computed on all rare categories. When using the back-off method, it helps to also add a binary indicator for whether or not the statistics come from the back-off bin.



Count-min sketch

All the categories, rare or frequent alike, are mapped through multiple hash functions with an output range, m, much smaller than the number of categories, k. When retrieving a statistic, recompute all the hashes of the category and return the smallest statistic. Having multiple hash functions mitigates the probability of collision within a single hash function. The scheme works because the number of hash functions times m, the size of the hash table, can be made smaller than k, the number of categories, and still retain low over-all collision probability.



Selecting the right strategy depends on the model, the size of the dataset, the importance of the categories, and the problem context

Hashing and bin counting methods for handling large categorical variables:

Feature Hashing

A hash function is a deterministic function that maps a potentially unbounded integer to a finite integer range [1, m]. Since the input domain is potentially larger than the output range, multiple numbers may get mapped to the same output. This is called a collision. A uniform hash function ensures that roughly the same number of numbers are mapped into each of the m bins.
This maintains the feature space while reducing the storage and processing time during machine learning training and evaluation cycles. Hash functions can be constructed for any object that can be represented numerically e.g. numbers, strings, complex structures.

Division method :

$h(K) = k \bmod M$

Here,

k is the key value, and

M is the size of the hash table.

Mid-square method

$h(K) = h(k \times k)$

Here, k is the key value.

Steps:

- Square the key.
- Extract the middle digits of the squared value

Digit folding method

$k = k_1, k_2, k_3, k_4, \ldots.., k_n$

$s = k_1 + k_2 + k_3 + k_4 + \ldots. + k_n$

$h(K) = s$

Cryptographic Hash Functions

Cryptographic hash functions are designed to be secure and are used in cryptography.

Examples include MD5, SHA-1, and SHA-256.

Characteristics: Pre-image resistance, Second pre-image resistance, Collision resistance.

Advantages: High security.

Disadvantages: Computationally intensive.

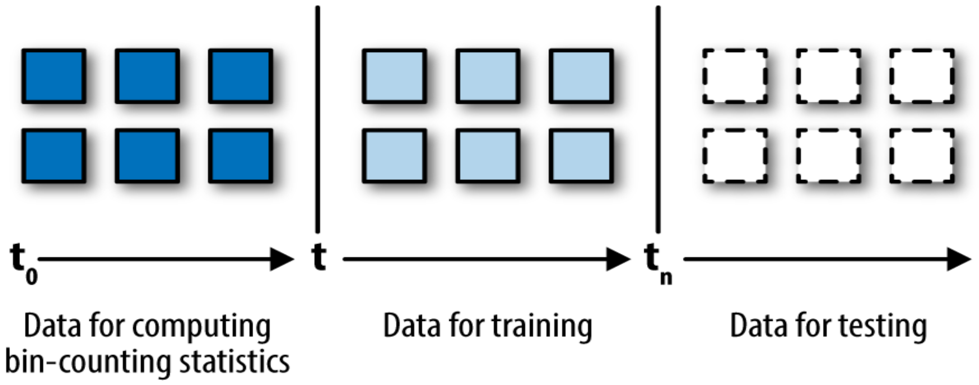
Bin Counting:

Instead of directly using the categorical value as a feature, we replace it with the conditional probability of the target given that value. This method encodes each category based on its statistical association with the target variable, allowing us to capture how strongly each category correlates with the outcome we're trying to predict. Rather than representing the identity of the category itself, we compute summary statistics (e.g., mean or probability) that quantify the relationship between the category and the target.Bin counting converts a categorical variable into statistics about the value.
It turns a large, sparse, binary representation of the categorical variable, such as that produced by one-hot encoding, into a very small, dense, real-valued numeric representation

Data leakage in bin counting method:

Data leakage occurs when information from outside the training dataset is used to create the model, leading to artificially inflated performance during training but poor generalization to unseen data. When using bin counting methods like **target encoding**, where categorical values are replaced by the conditional probability or mean of the target variable, there is a significant risk of data leakage if not handled carefully.

Use an earlier batch of data points for counting, use the current data points for training (mapping categorical variables to historical statistics we just collected), and use future data points for testing This fixes the problem of leakage.



Pros and cons of categorical variable encoding methods:

One-Hot Encoding

Description: Each category is converted into a new binary column, where 1 represents the presence of the category, and 0 represents its absence.

Pros:

- **Simple and Easy to Implement**: Straightforward encoding that doesn't assume any ordinal relationship between categories.
- **Does Not Impose Ordinal Bias**: Suitable for models that can't handle categorical variables (e.g., logistic regression, linear models).
- **Works Well with Tree-Based Models**: Decision trees and random forests naturally handle one-hot encoded features.

Cons:

- **High Dimensionality**: For variables with many categories (high cardinality), the number of columns grows significantly, leading to increased memory usage.

- **Sparse Data**: Creates sparse matrices, which can be inefficient to work with.
- **Not Efficient for High Cardinality**: May lead to overfitting and long training times for large datasets.

Dummy Coding

Description: Similar to one-hot encoding, but one category is chosen as the baseline (reference) and not encoded. The rest are encoded as binary columns.

Pros:

- **Reduces Collinearity**: By dropping one category, it avoids the multicollinearity problem that occurs when including all categories.
- **Efficient**: Slightly reduces dimensionality compared to one-hot encoding.
- **Common in Statistical Models**: Useful in linear models and regressions, where dropping a category helps with interpretation.

Cons:

- **Arbitrary Reference Choice**: The category you choose as the baseline/reference can influence how you interpret the coefficients.
- **Still Creates Many Columns**: Although slightly reduced in size compared to one-hot encoding, high-cardinality variables still result in high dimensionality.
- **Loses Information About the Dropped Category**: The reference category is not explicitly represented in the encoding.

Effect Coding (or Deviation Coding)

Description: Like dummy coding, but instead of using a baseline, each category is compared to the overall mean of the target variable. The baseline is encoded as -1, and other categories as 0 or 1.

Pros:

- **Meaningful Interpretations**: Each category is compared to the overall effect, rather than to a specific reference category.
- **Centering Effect**: Helps with centering the data, which can be useful in certain statistical models.
- **Useful in ANOVA/Linear Models**: Effect coding is often used in regression and ANOVA models, where comparisons to the overall mean are valuable.

Cons:

- **Not Intuitive**: Harder to interpret for beginners as the baseline is represented by -1, and the meaning of coefficients is less direct.
- **Still Requires Multiple Columns**: Like dummy coding, it still results in an increase in dimensionality, although with slight reductions compared to one-hot encoding.

- **Complexity**: Can introduce more complexity than is necessary for many machine learning models.

Feature Hashing (Hashing Encoding)

Description: A hash function maps categories to a fixed number of dimensions, with possible collisions (i.e., multiple categories mapped to the same hash value).

Pros:

- **Memory Efficient**: The number of output dimensions is fixed, making it scalable for high-cardinality variables.
- **Fast to Compute**: Hash functions are fast and require minimal preprocessing.
- **Reduces Dimensionality**: Helps avoid the memory and computational overhead of one-hot encoding or other high-dimensional encodings.

Cons:

- **Hash Collisions**: Different categories can map to the same column (collision), leading to information loss.
- **Non-Interpretable**: The encoded features are not human-readable, making model interpretability challenging.
- **No Control Over Representation**: The mapping is random, and relationships between categories are not maintained in the encoding.

Bin Counting (Target Encoding)

Description: Each category is replaced with the mean (or some other statistic) of the target variable for that category. For instance, in a classification task, each category can be encoded with the probability of the target being 1.

Pros:

- **Captures Target Relationships**: Captures the relationship between the categorical variable and the target, which can improve model performance.
- **Compact Representation**: Reduces dimensionality compared to one-hot encoding, as a single numerical value replaces each category.
- **Works Well for Tree-Based Models**: Provides a powerful representation for models that benefit from target-specific statistics.

Cons:

- **Data Leakage Risk**: Calculating the target statistic using the entire dataset can introduce data leakage, causing overfitting. Requires careful implementation (e.g., using cross-validation).

- **Overfitting**: Especially for small datasets or rare categories, the model may overfit to the target mean, leading to poor generalization.
- **Requires Regularization**: Target encoding often needs smoothing or regularization to avoid overfitting for rare categories.
- **Not Always Interpretable**: The encoded values are not as interpretable as other methods because they depend on the target.

Summary Table of Pros and Cons:

Encoding Method	Pros	Cons
One-Hot Encoding	Simple, works with most models, no ordinal assumptions	High dimensionality, inefficient for high-cardinality features, sparse matrices
Dummy Coding	Reduces collinearity, efficient, suitable for statistical models	Arbitrary baseline choice, still increases dimensionality, reference category information is lost
Effect Coding	Compares to overall mean, useful for statistical models like ANOVA	Hard to interpret, adds complexity, still results in increased dimensionality
Feature Hashing	Memory efficient, handles high-cardinality, fixed size	Hash collisions cause information loss, non-interpretable, random mappings
Bin Counting	Captures target relationships, reduces dimensionality, works well with tree models	Risk of data leakage, overfitting for rare categories, requires regularization, not always interpretable

Each method has its strengths and weaknesses, and the choice of encoding depends on the dataset, the model, and the nature of the categorical variables. For high-cardinality data, feature hashing or bin counting may be the best approach, while one-hot encoding and dummy coding are more suitable for low to medium-cardinality categorical variables.