



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to Data Science

Data wrangling and Feature Engineering

Categorical Encoding

SANKARA NAYAKI K

sankaranayaki@wilp.bits-pilani.ac.in

Disclaimer and Acknowledgement

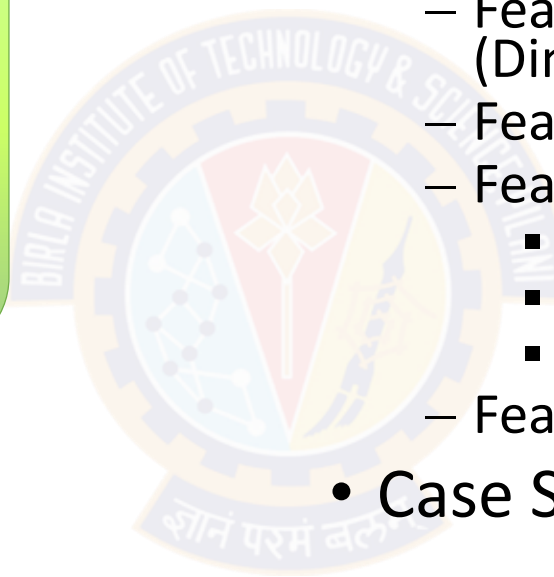


Disclaimer

- The content for these slides has been obtained from books and various other source on the Internet
- I here by acknowledge all the contributors for their material and inputs.
- I have provided source information wherever necessary
- I have added and modified the content to suit the requirements of the course

Data wrangling and Feature Engineering – Part-1

- Data cleaning
- Data Aggregation, Sampling,
- Handling Numeric Data
 - Discretization, Binarization
 - Normalization
 - Data Smoothing
- Dealing with textual Data
- Managing Categorical Attributes
 - Transforming Categorical to Numerical Values
 - Encoding techniques
- Feature Engineering
 - Feature Extraction (Dimensionality Reduction)
 - Feature Construction
 - Feature Subset selection
 - Filter methods
 - Wrapper methods
 - Embedded methods
 - Feature Learning
- Case Study involving FE tasks



Data Transformation

Variable Transformation

- Variable transformation involves changing the values of an attribute
- For each object (tuple), a transformation is applied to the value of the variable for that object
 - For example, if only the magnitude of a variable is important, then the values of the variable can be transformed by taking the absolute value
- We discuss two important types of variable transformations: simple functional transformations and normalization

Transformation using Simple Functions

- For this type of variable transformation, a simple mathematical function is applied to each value individually
- If x is a variable, then examples of such transformations include:
 - x^k , $\log x$, e^x , \sqrt{x} , $1/x$, $\sin x$, or $|x|$
- In statistics, variable transformations, especially sqrt, log, and $1/x$, are often used to transform data that does not have a Gaussian (normal) distribution into data that does
- While this can be important, other reasons often take precedence in data mining

Transformation using Simple Functions

- Suppose the variable of interest is the number of data bytes in a session, and the number of bytes ranges from 1 to 1 billion
- This is a huge range, and it may be advantageous to compress it by using a \log_{10} transformation
- In this case, sessions that transferred 10^8 and 10^9 bytes would be more similar to each other than sessions that transferred 10 and 1000 bytes (9 - 8 : 1 versus 3 - 1 : 2)
- For applications such as network intrusion detection, this may be desired since the first two sessions most likely represent transfers of large files, while the latter two sessions could be two quite distinct types of sessions

Transformation using Simple Functions

- Variable transformations should be applied with caution since they change the nature of the data
- For instance, the transformation $1/x$ reduces the magnitude of values that are 1 or larger, but increases the magnitude of values between 0 and 1
 - For example, the values $\{1, 2, 3\}$ become $\{1, \frac{1}{2}, \frac{1}{3}\}$, but the values $\{1, \frac{1}{2}, \frac{1}{3}\}$ become $\{1, 2, 3\}$
 - Thus, $1/x$ transformation reverses the order
- To understand the effect of a transformation, it is important to ask questions such as:
 - Does the order need to be maintained?
 - Does the transformation apply to all values, especially negative values and 0?
 - What is the effect of the transformation on the values between 0 and 1?

Transformation using Normalization

- Another common type of variable transformation is the standardization or normalization of a variable
 - In the data mining community the terms are often used interchangeably
 - In statistics, the term normalization can be confused with the transformations used for making a variable normal, i.e., Gaussian
- Normalizing the data attempts to give all attributes an equal weight
- The goal of standardization or normalization is to make an entire set of values have a particular property
 - E.g., "standardizing a variable" in statistics
 - If \bar{x} is the mean (average) of the attribute values and s_x is their standard deviation, then the transformation $x' = (x - \bar{x})/s_x$ creates a new variable that has 0 mean and standard deviation of 1

Transformation using Normalization

- Normalization is particularly useful for:
 - classification algorithms involving neural networks or
 - distance measurements such as nearest-neighbor classification and clustering
- In neural network backpropagation algorithm for classification
 - normalizing the input values for each attribute in the training tuples will help speed up the learning phase
- In distance-based methods
 - normalization helps prevent attributes with initially large ranges (e.g., income) from outweighing attributes with initially smaller ranges (e.g., binary attributes)
- It is also useful when given no prior knowledge of the data

Transformation using Normalization

- How normalization helps in distance-based methods
 - Consider comparing people based on two variables: age and income
 - For any two people, the difference in income will likely be much higher in absolute terms (hundreds or thousands of dollars) than the difference in age (less than 150)
 - The comparison between people will be dominated by differences in income
 - If the similarity or dissimilarity of two people is calculated using the similarity or dissimilarity measures (E.g., Euclidean distance), the income values will dominate the calculation
 - The mean and standard deviation are strongly affected by outliers, so data is often normalized

Transformation using Normalization

- Such a transformation is often necessary to avoid having a variable with large values dominate the results of the calculation
 - First, the mean is replaced by the **median**, i.e., the middle value
 - Second, the standard deviation is replaced by the **absolute standard deviation**
 - If x is a variable, then the absolute standard deviation of x is given by

$$\sigma_A = \sum_{i=1}^m |x_i - \mu|$$

Where

- x_i is the i^{th} value of the variable
- m is the number of objects (tuples)
- μ is either mean or median

Transformation using Normalization

- Methods of data normalization
 - min-max normalization
 - Performs a linear transformation on the original data
 - z-score normalization (or zero-mean normalization)
 - The values for an attribute, A , are normalized based on the mean (i.e., average) and standard deviation of A
 - normalization by decimal scaling
 - Normalizes by moving the decimal point of values of attribute A

Transformation using Normalization

- Min-Max normalization

- Suppose that \min_A and \max_A are the minimum and maximum values of an attribute, A
- Min-Max normalization maps a value, x_i , of A to x'_i in the range $[\text{new_min}_A, \text{new_max}_A]$ by computing

$$x'_i = \frac{x_i - \min_A}{\max_A - \min_A} (\text{new_max}_A - \text{new_min}_A) + \text{new_min}_A$$

- Min-Max normalization preserves the relationships among the original data values
- It will encounter an "out-of-bounds" error if a future input case for normalization falls outside of the original data range for A

Transformation using Normalization

- Min-Max normalization – Example

- Suppose that the minimum and maximum values for the attribute income are \$12,000 and \$98,000, respectively
- We would like to map income to the range [0.0, 1.0]
- By min-max normalization, a value of \$73,600 for income is transformed to

$$\frac{73600 - 12000}{98000 - 12000} (1.0 - 0.0) + 0.0 = 0.716$$

Transformation using Normalization

- z-Score Normalization

- The values for an attribute, A, are normalized based on the mean (i.e., average) and standard deviation of A

- A value, x_i , of A is normalized to x'_i by computing

$$x'_i = \frac{x_i - \bar{x}}{\sigma_A}$$

- Where \bar{x} and σ_A are the mean and standard deviation, respectively of A

- This method of normalization is useful

- when the actual minimum and maximum of attribute A are unknown, or
 - when there are outliers that dominate the min-max normalization

Transformation using Normalization

- z-score normalization – Example

- Suppose that the mean and standard deviation of the values for the attribute income are \$54,000 and \$16,000, respectively
- With z-score normalization, a value of \$73,600 for income is transformed to

$$\frac{73600 - 54000}{16000} = 1.225$$

- A variation of this z-score normalization replaces the standard deviation by the mean absolute deviation of A

Transformation using Normalization

- z-score normalization – Example

- The mean absolute deviation of A, denoted s_A , is

- $s_A = \frac{1}{n} (|x_1 - \bar{x}| + |x_2 - \bar{x}| + \dots + |x_n - \bar{x}|)$

- Thus, z-score normalization using the mean absolute deviation is

- $x'_i = \frac{x_i - \bar{x}}{s_A}$

- The mean absolute deviation, s_A , is more robust to outliers than the standard deviation, σ_A

- When computing the mean absolute deviation, the deviations from the mean (i.e., $|x_i - \bar{x}|$) are not squared; hence, the effect of outliers is somewhat reduced

Transformation using Normalization

- Normalization by decimal scaling
 - Normalizes by moving the decimal point of values of attribute A
 - The number of decimal points moved depends on the maximum absolute value of A
 - A value, x_i , of attribute A is normalized to x'_i by computing
 - $x'_i = \frac{x_i}{10^j}$
 - Where j is the smallest integer such that $\max(|x'_i|) < 1$

Transformation using Normalization

- Normalization by decimal scaling - Example
 - Suppose that the recorded values of A range from -986 to 917
 - The maximum absolute value of A is 986
 - To normalize by decimal scaling, we divide each value by 1000 (i.e., $j = 3$) so that -986 normalizes to -0.986 and 917 normalizes to 0.917

Data wrangling and Feature Engineering – Part-1

- Data cleaning
- Data Aggregation, Sampling,
- Handling Numeric Data
 - Discretization, Binarization
 - Normalization
 - Data Smoothing
- Dealing with textual Data
- Managing Categorical Attributes
 - Transforming Categorical to Numerical Values
 - Encoding techniques
- Feature Engineering
 - Feature Extraction (Dimensionality Reduction)
 - Feature Construction
 - Feature Subset selection
 - Filter methods
 - Wrapper methods
 - Embedded methods
 - Feature Learning
- Case Study involving FE tasks

Dealing with Textual Data



Introduction

- If all the data in the world was equivalent to the water on earth, then textual data is like the ocean, making up a majority of the volume (Predictive Analytics, Siegel, 2013)
- Typically, a cleaned and organized table consisting of rows and columns of data is fed as input to an algorithm
- The output from the algorithm is a model that could then be used to predict outcomes from a new data set or to find patterns in data
- Are the same techniques applicable to extract patterns and predict outcomes when the input data looks like normal written communication?
- Question is how to extract patterns and discover new knowledge by applying mining techniques, not on ordered data, but on unstructured natural language - text?

Introduction

- Text analytics is driven by the need to process natural human language
- Unlike numeric or categorical data, natural language does not exist in a structured format consisting of rows (of examples) and columns (of attributes)
 - Text mining is, therefore, the domain of unstructured data science
- Traditionally, mining refers to the process of separating dirt from valuable metal
- Text mining refers to the process of separating valuable keywords from a mass of other words (or relevant documents) and use them to identify meaningful patterns or make predictions

Introduction

- Text mining has many applications, such as:
 - Entity identification
 - Plagiarism detection
 - Topic identification
 - Text clustering
 - Translation
 - Automatic text summarization
 - Fraud detection
 - Spam filtering
 - Sentiment analysis



Dealing with Textual Data

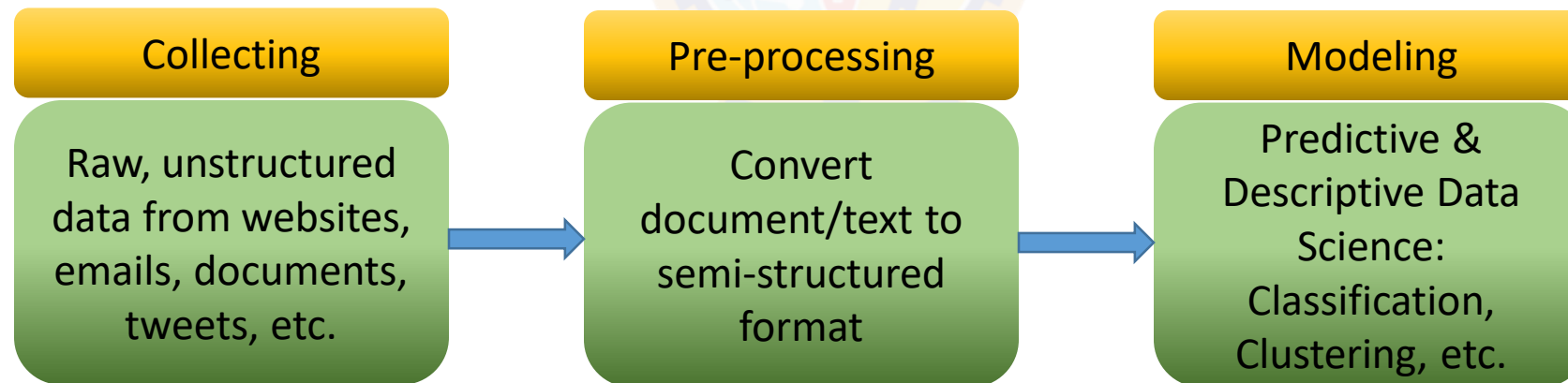
innovate

achieve

lead

How it works?

- The basic step in text mining involves converting unstructured text into semi-structured data
- Models can be trained on the semi-structured data to detect patterns in the unseen text



Term Frequency – Inverse Document Frequency

- Consider a web search problem where the user types in some keywords and the search engine extracts all the documents (essentially, web pages) that contain these keywords
- How does the search engine know which web pages to serve up?
- In addition to using network rank or page rank, the search engine also runs some form of text mining to identify the most relevant web pages
- Suppose for example, that the user types in the following keywords:
"RapidMiner books that describe text mining."
- In this case, the search engines run on the following basic logic:
 - 1) Give a high weightage to those keywords that are relatively rare
 - 2) Give a high weightage to those web pages that contain a large number of instances of the rare keywords

Term Frequency – Inverse Document Frequency

- In this context, what is a rare keyword?
- Clearly, English words like "that," "books," "describe," and "text" possibly appear in a large number of web pages, whereas "RapidMiner" and "mining" may appear in a relatively smaller number of web pages
 - A quick web search returned about 9,66,00,00,000 results (0.58 seconds) for the word "books," whereas "RapidMiner" returned only about 8,40,000 results (0.41 seconds)
- Therefore, only those pages that not only contain the rare keywords (logic 1), but also have a high number of instances of the rare keywords (logic 2) should appear at the top of the search results
- The technique of calculating this weighting is called term TF-IDF, which stands for term frequency-inverse document frequency

Term Frequency – Inverse Document Frequency

- Term Frequency (TF):

- Refers to the ratio of the number of times a keyword appears in a given document, n_k (where k is the keyword), to the total number of terms in the document, n :

$$TF = \frac{n_k}{n}$$

- Inverse Document Frequency (IDF): is defined as follows

$$IDF = \log_2 \left(\frac{N}{N_k} \right)$$

- N is the number of documents under consideration
 - In a search engine context, N is the number of all the indexed web pages
 - For most text mining problems, N is the number of documents that one is trying to mine
- N_k is the number of documents that contain the keyword, k

Term Frequency – Inverse Document Frequency

- Term Frequency: $TF = \frac{n_k}{n}$
 - A common English word such as "this" will have a fairly high TF score and a word such as "RapidMiner" will have a much lower TF score
- Inverse Document Frequency: $IDF = \log_2 \left(\frac{N}{N_k} \right)$
 - Again, a word such as "this" would appear in every document and, thus, the ratio (N/N_k) would be close to 1
 - Thus, the IDF score would be close to zero
 - However, a word like "RapidMiner" would possibly appear in a relatively fewer number of documents and so the ratio (N/N_k) would be much greater than 1
 - Thus, the IDF score would be high for this less common keyword

Term Frequency – Inverse Document Frequency

- TF-IDF is expressed as follows:

$$TF - IDF = \frac{n_k}{n} \times \log_2 \left(\frac{N}{N_k} \right)$$

- When the high TF for "this" is multiplied by its corresponding low IDF, a low (or zero) TF-IDF will be reached,
- Whereas, when the low TF for "RapidMiner" is multiplied by its corresponding fairly high IDF, a relatively higher TF-IDF would be obtained
- Typically, TF-IDF scores for every word in the set of documents is calculated in the preprocessing stage

Terminology – Steps involved in the Textual Data Processing

- Following are the broad steps involved in Text Mining
 - Removing special characters, changing the case (up-casing and down-casing)
 - Tokenization – process of discretizing words within a document
 - Creating Document Vector or Term Document Matrix
 - Filtering Stop Words
 - Term Filtering
 - Stemming/Lemmatization
 - Forming n-grams and storing them in the document vector

Terminology

- Consider the following two sentences:
 - "This is a book on data mining" and
 - "This book describes data mining and text mining using RapidMiner."
- Suppose the objective is to perform a comparison between them, or a similarity mapping
- For this purpose, each sentence is one unit of text that needs to be analyzed
- These two sentences could be embedded:
 - in an email message, in two separate web pages, in two different text files, or else they could be two sentences in the same text file

Terminology - Tokenization

- Document
 - In the text mining context, each sentence is considered a distinct document
- Token
 - Each word is called a token
- Tokenization
 - The process of discretizing words within a document is called tokenization
- For now, a document here is simply a sequential collection of tokens (words)

Document 1 This is a book on data mining

Document 2 This book describes data mining and text mining using RapidMiner

Terminology – Document Vector or TDM

- We can create a matrix where each column consists of a token and the cells show the counts of the number of times a token appears
- Each token is now an attribute in standard data science parlance and each document is an example (record)
- Basically, unstructured raw data is now transformed into a format that is recognized by machine learning algorithms for training
- This table is referred to as Document Vector or Term Document Matrix (TDM)

Table 9.1 Building a Matrix of Terms From Unstructured Raw Text

	This	is	a	book	on	data	mining	describes	text	rapidminer	and	using
Document 1	1	1	1	1	1	1	1	0	0	0	0	0
Document 2	1	0	0	1	0	1	2	1	1	1	1	1

Terminology – Document Vector or TDM

- Suppose a third statement is added
 - "RapidMiner is offered as an open source software program."
- This new document will increase
 - The number of rows of the matrix by one (Document 3)
 - The number of columns by seven (seven new words or tokens were introduced)
- This results in zeroes being recorded in nine other columns for row 3
- As more new statements are added that have little in common, we end up with a very sparse matrix

Table 9.1 Building a Matrix of Terms From Unstructured Raw Text

	This	is	a	book	on	data	mining	describes	text	rapidminer	and	using
Document 1	1	1	1	1	1	1	1	0	0	0	0	0
Document 2	1	0	0	1	0	1	2	1	1	1	1	1

Terminology – Document Vector or TDM

- We could also choose to use the term frequencies (TF) for each token instead of simply counting the number of occurrences
 - TF can be obtained by dividing number in the table by number of words in the row (document)
- Similarly, TF-IDF scores can also be used for each term to create the document vector

Table 9.2 Using Term Frequencies Instead of Term Counts in a TDM

	This	is	a	book	on	data	mining	describes	text	rapidminer	and	using
Document 1	$1/7 = 0.1428$	0.1428	0.1428	0.1428	0.1428	0.1428	0.1428	0	0	0	0	0
Document 2	$1/10 = 0.1$	0	0	0.1	0	0.1	0.2	0.1	0.1	0.1	0.1	0.1

TDM, *Term document matrix*.

ExampleSet (2 examples, 0 special attributes, 12 regular attributes)

Row No.	RapidMiner	This	a	and	book	data	describes	is	mining	on	text	using
1	0	0	0.577	0	0	0	0	0.577	0	0.577	0	0
2	0.447	0	0	0.447	0	0	0.447	0	0	0	0.447	0.447

Terminology – Stop Words

- Note that in the sample documents, there are common words such as "a," "this," "and," and other similar terms
- In larger documents, there will be a higher number of such terms that do not really convey specific meaning
- Most parts of speech such as articles, conjunctions, prepositions, and pronouns need to be filtered before additional analysis is performed
- Such terms are called stop words
- Stop word filtering is usually the second step that follows immediately after tokenization.
- The document vector gets reduced significantly after applying standard English stop word filtering

Row No.	RapidMiner	book	data	describes	mining	text	using
1	0	1	1	0	1	0	0
2	1	1	1	1	2	1	1

Terminology – Stop Words, Lexical Substitution

- In addition to filtering standard stop words, domain specific terms might also need to be filtered out
 - For example, if we are analyzing text related to the automotive industry, we may want to filter out terms common to this industry such as "car," "automobile," "vehicle," and so on
- This is generally achieved by creating a separate dictionary where these context specific terms can be defined and then term filtering can be applied to remove them from the data
- Lexical substitution
 - It is the process of finding an alternative for a word in the context of a clause
 - It is used to align all the terms to the same term based on the field or subject which is being analyzed
 - This is especially important in areas with specific jargon, e.g., in clinical settings

Terminology – Stemming

- Stemming is usually the next process step following term filtering
- Words such as "recognized," "recognizable," or "recognition" may be encountered in different usages, but contextually they may all imply the same meaning
- For example,
 - "Einstein is a well-recognized name in physics" or
 - "The physicist went by the easily recognizable name of Einstein" or
 - "Few other physicists have the kind of name recognition that Einstein has."
- The root of all these highlighted words is "recognize."
- The conversion of unstructured text to structured data can be simplified by reducing terms in a document to their basic stems
 - Because now only the occurrence of the root terms has to be taken into account.
- This process is called stemming

Terminology – Stemming

- The most common stemming technique for text mining in English is the Porter method (Porter, 1980)
- Porter stemming works on a set of rules where the basic idea is to remove and/or replace the suffix of words
- For example:
 - Replace all terms which end in 'ies' by 'y,' such as replacing the term "anomalies" with "anomaly."
 - Stem all terms ending in "s" by removing the "s," as in "algorithms" to "algorithm."
- While the Porter stemmer is extremely efficient, it can make mistakes that could prove costly
- For example:
 - "arms" and "army" would both be stemmed to "arm," which would result in somewhat different contextual meanings

Terminology – Lemmatization

- Lemmatization has this same goal as Stemming, but does so in a more grammatically sensitive way
- For example
 - While both stemming and lemmatization would reduce "cars" to "car," lemmatization can also bring back conjugated verbs to their unconjugated forms such as "are" to "be."
 - Which one to use depends on our case
- Lemmatization uses POS Tagging (Part of Speech Tagging) heavily
- POS Tagging is the process of attributing a grammatical label to every part of a sentence

Terminology – Lemmatization

- Take the sentence:
 - "Game of Thrones is a television series."
- If we apply POS Tagging on it we get:
 - (`{"game": "NN"}, {"of": "IN"}, {"thrones": "NNS"}, {"is": "VBZ"}, {"a": "DT"}, {"television": "NN"}, {"series": "NN"}`)
- Where:
 - NN = noun, IN = preposition, NNS = noun in its plural form, VBZ = third-person singular verb, and DT = determiner
- POS Tagging is a use case of sentence-tokenization rather than word-tokenization
- After the POS Tagging is complete we can still proceed to word tokenization, but a POS Tagger requires whole sentences
- Combining POS Tagging and lemmatization is likely to give cleaner data than using only a stemmer

Terminology – *n*-grams

- There are families of words in the spoken and written language that typically go together
- For example, the word "Good" is usually followed by either "Morning," "Afternoon," "Evening," "Night," or in Australia, "Day."
- Grouping such terms, called *n*-grams, and analyzing them statistically can present new insights
- The final preprocessing step typically involves forming these n-grams and storing them in the document vector
- Algorithms providing n-grams become computationally expensive and the results become huge so in practice the amount of "n" will vary based on the size of the documents and the corpus
- Table shows a TF-based document vector for bigrams (n = 2) from the examples
- As can be seen, terms like "data mining" and "text mining" and "using RapidMiner" can be quite meaningful in this context

Row...	label	RapidMiner	book	book_data	book_descr...	data	data_mining	describes	describes_data	mining	mining_text	mining_usi...	text_0	text_mining	using	using_RapidMiner
1	text1	0	0.447	0.447	0	0.447	0.447	0	0	0.447	0	0	0	0	0	0
2	text2	0.243	0.243	0	0.243	0.243	0.243	0.243	0.243	0.485	0.243	0.243	0.243	0.243	0.243	0.243

Dealing with Textual Data



Summary of Terms and Steps

A Typical Sequence of PreProcessing Steps to Use in Text Mining

Step	Action	Result
1	Tokenize	Convert each word or term in a document into a distinct attribute
2	Stop word removal	Remove highly common grammatical tokens/words
3	Filtering	Remove other very common tokens
4	Stemming	Trim each token to its most essential minimum
5	<i>n</i> -grams	Combine commonly occurring token pairs or tuples (more than 2)

Data wrangling and Feature Engineering – Part-1

- Data cleaning
- Data Aggregation, Sampling,
- Handling Numeric Data
 - Discretization, Binarization
 - Normalization
 - Data Smoothing
- Dealing with textual Data
- Managing Categorical Attributes
 - Transforming Categorical to Numerical Values
 - Encoding techniques
- Feature Engineering
 - Feature Extraction (Dimensionality Reduction)
 - Feature Construction
 - Feature Subset selection
 - Filter methods
 - Wrapper methods
 - Embedded methods
 - Feature Learning
- Case Study involving FE tasks

Categorical Encoding

Categorical Encoding

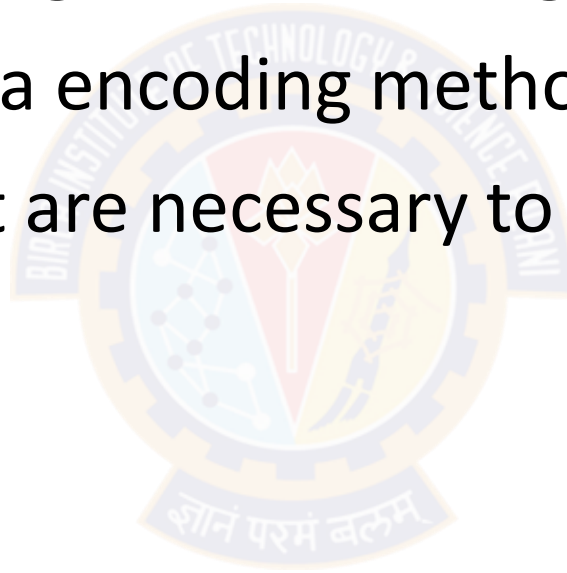
innovate

achieve

lead

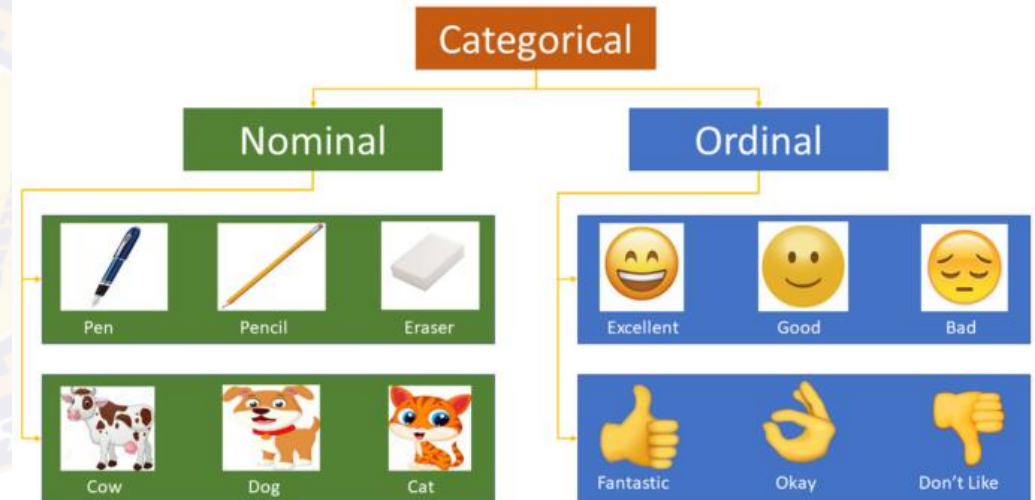
Learning objectives

- Articulate the need for categorical encoding
- List and define various data encoding methods
- Make the calculations that are necessary to get meaningful encodings



Categorical Data

- Categorical variables can be divided into two categories:
 - Nominal (No particular order) and
 - Ordinal (some ordered)
- Nominal Variables
 - Red, Yellow, Pink, Blue
 - Singapore, Japan, USA, India, Korea
 - Cow, Dog, Cat, Snake
- Ordinal Variables
 - High, Medium, Low
 - "Strongly agree", Agree, Neutral, Disagree, and "Strongly Disagree"
 - Excellent, Okay, Bad



Categorical Encoding

innovate

achieve

lead

Overview

- Many machine learning algorithms cannot use non-numeric data
- These non-numeric attributes (features) are represented by strings
- We need to transforming them into numbers before using machine learning algorithms
- The different ways of transforming this are called encodings.
- Categorical encoding refers to replacing the category strings by a numerical representation
- The performance of many algorithm varies based on how the categorical variable is encoded
- The goal of categorical encoding is:
 - To build predictive features from categories
 - To produce variables that can be used to train machine learning models

Different Types of Categorical Encoding

- One Hot Encoding
- Label Encoding
- Ordinal Encoding
- Helmert Encoding
- Binary Encoding
- Frequency Encoding
- Mean Encoding
- Weight of Evidence Encoding
- Probability Ratio Encoding
- Hashing Encoding
- Backward Difference Encoding
- Leave One Out Encoding
- James-Stein Encoding
- M-estimator Encoding

One Hot Encoding

- The most common way to represent categorical variables is using the one-hot encoding or one-out-of-N encoding, also known as dummy variables
- One hot encoding consists of encoding each categorical variable with a set of boolean variables, which take values 0 or 1
- One hot encoding indicates if a category is present for each observation

Status		Student	Unemployed	Employee	Retired
Student		1	0	0	0
Unemployed		0	1	0	0
Employee		0	0	1	0
Retired		0	0	0	1

One Hot Encoding

- Using "k-1" fields
 - More generally, a categorical variable should be encoded by creating k-1 binary variables, where k is the number of distinct categories.
 - In the case of binary variables, like gender where $k=2$ (male / female) we need to create only 1 ($k - 1 = 1$) binary variable
 - One hot encoding into k-1 binary variables takes into account that we can use 1 less dimension and still represent the whole information:
 - If the observation is 0 in all the binary variables, then it must be 1 in the final (not present) binary variable.
- Encoding categorical variables into k - 1 binary variables, is better, as it avoids introducing redundant information

Status		Student	Unemployed	Employee
Student		1	0	0
Unemployed		0	1	0
Employee		0	0	1
Retired		0	0	0

One Hot Encoding

- Using "k-1" fields
 - When defining dummy variables, a common mistake is to define too many variables
 - If a categorical variable can take on k values, it is tempting to define k dummy variables
 - A kth dummy variable is redundant; it carries no new information
 - And it creates a severe multicollinearity problem for the analysis
- Using k dummy variables when only k - 1 dummy variables are required is known as the dummy variable trap. Avoid this trap!

Status	Student	Unemployed	Employee
Student	1	0	0
Unemployed	0	1	0
Employee	0	0	1
Retired	0	0	0

One Hot Encoding

- There are a few occasions when it is better to encode variables into k dummy variables:
 - when building tree based algorithms
 - when doing feature selection by recursive algorithms
 - when interested in determine the importance of each single category

workclass	Government Employee	Private Employee	Self Employed	Self Employed Incorporated
Government Employee	1	0	0	0
Private Employee	0	1	0	0
Self Employed	0	0	1	0
Self Employed Incorporated	0	0	0	1

One Hot Encoding

- Advantages
 - The result is binary (rather than ordinal) and everything sits in an orthogonal vector space
 - Makes no assumption about the distribution or categories of the variable
 - Keeps all the information of the categorical variable
 - Suitable for linear models
- Disadvantages
 - High cardinality (many categories), expands the feature space and we will start fighting with the curse of dimensionality
 - Does not add extra information while encoding
 - Many dummy variables may be identical, introducing redundant information

Label / Integer Encoding

- Label encoding consist in replacing the categories by digits from 1 to n (or 0 to n-1, depending the implementation)
 - Where n is the number of distinct categories of the variable.
- The numbers are assigned arbitrarily
- This encoding method allows for quick benchmarking of machine learning models

Status		Status
Student		1
Unemployed		2
Employee		3
Retired		4

Label / Integers Encoding

- One major issue with this approach is there is no relation or order between the classes, but the algorithm might consider them as some order, or there is some relationship
 - For example, it may look like (Cold < Hot < Very Hot < Warm....0 < 1 < 2 < 3)
- Advantages
 - Straightforward to implement
 - Does not expand the feature space
 - Can work well enough with tree based algorithms
- Limitations
 - Does not add extra information while encoding
 - Not suitable for linear models
 - Does not handle new categories in test set automatically

Frequency/Count Encoding

- Categories are replaced by the count or percentage of observations that show that category in the dataset
- Captures the representation of each label in a dataset
- Very popular encoding method in Kaggle competitions
- Assumption:
 - the number observations shown by each category is predictive of the outcome

	Count Encoding	Freq Encoding
Status	Status	Status
Student	3	$3/8 = 0.375$
Unemployed	1	$1/8 = 0.125$
Student	3	$3/8 = 0.375$
Retired	2	$2/8 = 0.25$
Retired	2	$2/8 = 0.25$
Employee	2	$2/8 = 0.25$
Student	3	$3/8 = 0.375$
Employee	2	$2/8 = 0.25$

Frequency/Count Encoding

- Advantages
 - Straightforward to implement
 - Does not expand the feature space
 - Can work well enough with tree based algorithms
- Limitations
 - Not suitable for linear models
 - Does not handle new categories in test set automatically
 - If 2 different categories appear the same number of times in the dataset, they will be replaced by the same number:
 - may lose valuable information.

	Count Encoding	Freq Encoding
Status	Status	Status
Student	3	$3/8 = 0.375$
Unemployed	1	$1/8 = 0.125$
Student	3	$3/8 = 0.375$
Retired	2	$2/8 = 0.25$
Retired	2	$2/8 = 0.25$
Employee	2	$2/8 = 0.25$
Student	3	$3/8 = 0.375$
Employee	2	$2/8 = 0.25$

Ordinal Encoding

- To encode a categorical feature that has a natural order, such as a movie rating (e.g., “bad,” “average,” “good”), the simplest option is to use ordinal encoding:
 - sort the categories in their natural order and map each category to its rank
 - e.g., "bad" maps to 0, "average" maps to 1, and "good" maps to 2

Temperature		Status
Very hot		0
Hot		1
Warm		2
Cold		3
Freezing		4

Categorical Encoding

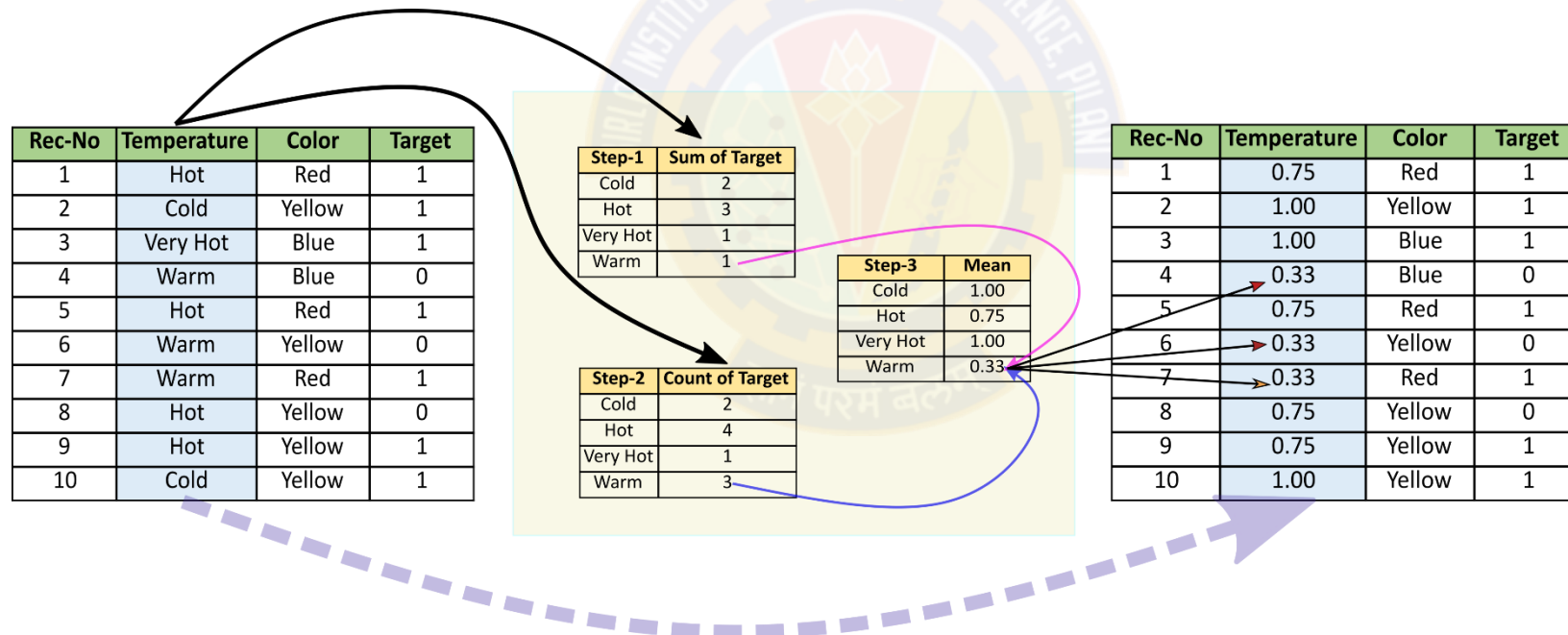
innovate

achieve

lead

Mean/Target Encoding

- Mean Encoding takes the number of labels into account along with the target variable to encode the labels



Categorical Encoding

innovate

achieve

lead

Mean/Target Encoding

- Mean Encoding takes the number of labels into account along with the target variable to encode the labels

Job	Age	Target
Data Scientist	45	1
Data Scientist	40	0
Data Analyst	32	1
Data Engineer	35	1
Data Scientist	42	1
Data Analyst	33	1
Data Architect	44	1
Data Architect	50	0
Data Scientist	50	1
Data Engineer	36	1
Data Analyst	36	0

Job	Target Count
Data Scientist	4
Data Analyst	3
Data Engineer	2
Data Architect	2

Job	Target Sum
Data Scientist	3
Data Analyst	2
Data Engineer	2
Data Architect	1

Job	Mean
Data Scientist	$3/4 = 0.75$
Data Analyst	$2/3 = 0.67$
Data Engineer	$2/2 = 1.00$
Data Architect	$1/2 = 0.50$

Categorical Encoding

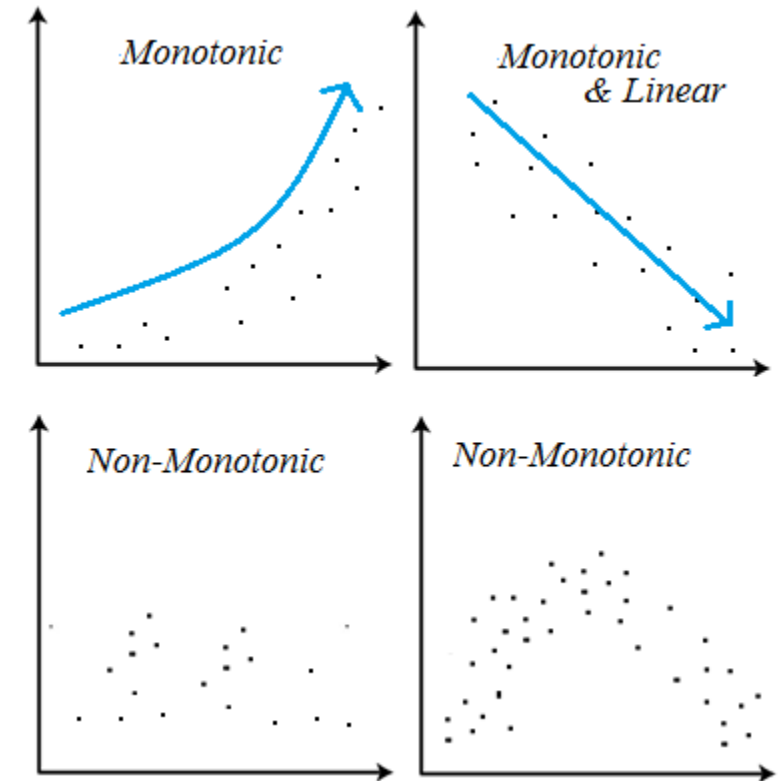
innovate

achieve

lead

Mean/Target Encoding

- Advantages
 - Increases the quality of a classification
 - Straightforward to implement
 - Does not expand the feature space
 - Creates monotonic relationship between categories and target
- Disadvantages
 - May lead to over-fitting
 - Difficult to implement together with cross-validation with current libraries
 - If 2 categories show the same mean of target, they will be replaced by the same number => potential loss of value
 - The fact that we are encoding the feature based on target classes may lead to data leakage, rendering the feature biased
- Note:
 - Monotonic variables increase (or decrease) in the same direction, but not always at the same rate
 - Linear variables increase (or decrease) in the same direction at the same rate



Categorical Encoding

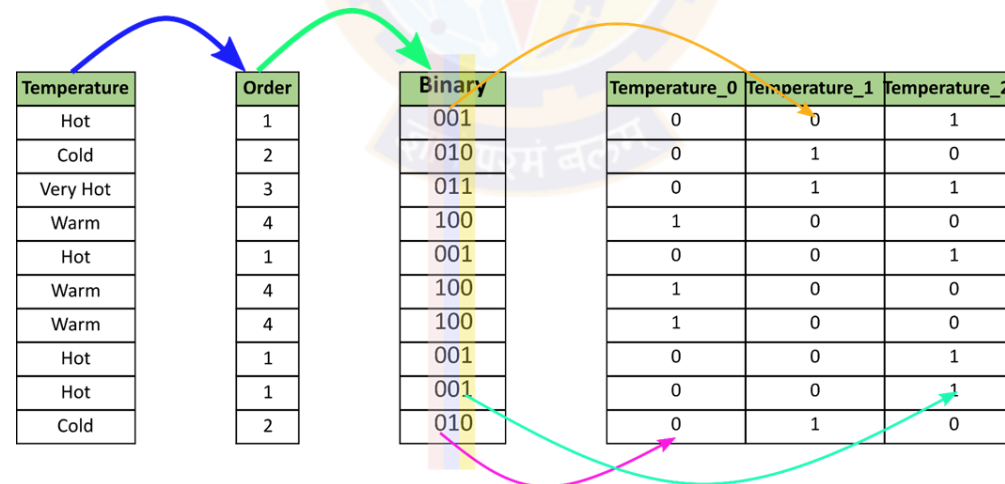
innovate

achieve

lead

Binary Encoding

- Binary encoding can be thought of as a hybrid of one-hot and hashing encoders
- Here's how it works:
 - The categories are encoded as per Ordinal Encoding
 - Then those integers are converted into binary code, so for example 5 becomes 101 and 10 becomes 1010
 - Then the digits from that binary string are split into separate columns
 - So if there are 4–7 values in an ordinal column then 3 new columns are created:
 - one for the first bit, one for the second, and one for the third.



Binary Encoding

- Advantages

- Straightforward to implement.
- Does not expand the feature space too much.
- Binary creates fewer features than one-hot
 - It is more memory efficient
- Preserves some uniqueness of values in the column
- Binary really shines when the cardinality of the column is higher
 - E.g., 50 US states

- Disadvantages

- It exposes the loss of information during encoding.
- It lacks the human-readable sense
- With only three levels, the information embedded becomes muddled
- There are many collisions and the model can't glean much information from the features

Feature Hashing

- The central part of the hashing encoder is the hash function
- Feature hashing, also known as the hashing trick, is a fast and space-efficient way of vectorizing features
 - i.e. turning arbitrary features into indices in a vector or matrix
- It works by applying a hash function to the features and using their hash values as indices directly
- Feature hashing maps each category in a categorical feature to an integer within a pre-determined range

Feature Hashing

- This technique is applied typically in text mining
- The technique involves converting data into a vector of features
- When this is done using hashing, we call the method "feature hashing" or "the hashing trick"
- Let's say our text is: "Taking my dog for a walk is fun."
- We would like to represent this as a vector
- The first thing we need is to fix the length of the vector (the number of dimensions) we are going to use, let's say we would like to use 5 dimensions.
- Once we fix the number of dimensions we need a hash function that will take a string as input and returns a number between 0 and $n-1$, in our case between 0 and 4
- Any good hash function can be used and you just use $h(\text{string}) \bmod n$ to make it return a number between 0 and $n-1$

Feature Hashing

- Once we apply the hash function, we can simply construct our vector as: (4,3,1,1,2,0,1,4)

Feature	h	Remainder
Taking	$h(\text{Taking}) \bmod 5$	4
my	$h(\text{my}) \bmod 5$	3
dog	$h(\text{dog}) \bmod 5$	1
for	$h(\text{for}) \bmod 5$	1
a	$h(\text{a}) \bmod 5$	2
walk	$h(\text{walk}) \bmod 5$	0
is	$h(\text{is}) \bmod 5$	1
fun	$h(\text{fun}) \bmod 5$	4


	0	1	2	3	4
Taking	0	0	0	0	1
my	0	0	0	1	0
dog	0	1	0	0	0
for	0	1	0	0	0
a	0	0	1	0	0
walk	1	0	0	0	0
is	0	1	0	0	0
fun	0	0	0	0	1

Feature Hashing

- In document classification task, the input to the machine learning algorithm (both during learning and classification) is free text
- From this text, a bag of words (BOW) representation is constructed
 - The individual tokens are extracted and counted, and each distinct token in the training set defines a feature (independent variable) of each of the documents in both the training and test sets.
- Therefore, the bags of words for a set of documents is regarded as a term-document matrix where each row is a single document, and each column is a single feature/word
 - The entry i, j in such a matrix captures the frequency (or weight) of the j 'th term of the vocabulary in document i
- Typically, these vectors are extremely sparse
- The common approach is to construct a dictionary representation of the vocabulary of the training set, and use that to map words to indices

Feature Hashing

- Hash tables and tries are common candidates for dictionary implementation.
- E.g., the three documents
 - John likes to watch movies.
 - Mary likes movies too.
 - John also likes football.
- can be converted, using the dictionary



Feature	h
John	1
likes	0
to	1
watch	1
movies	1
Mary	1
too	1
also	0
football	1

Feature Hashing

- Advantages

- The number of dimensions will be far less than the number of dimensions with encoding like One Hot Encoding.
 - Even if we have over 1000 distinct categories in a feature and we set $b=10$ as the final feature vector size (a pre-determined range), the output feature set will still have only 10 features as compared to 1000 binary features if we used a one-hot encoding scheme

- Disadvantages

- Hash functions can hash different keys to the same integer value (this is known as 'collision')
 - It will certainly happen in this case as we had represented 1000 distinct categories as 10 columns

- Note

- You have to do a trade-off between the No. of categories getting mapped to the same integer value (% of collision) and the final feature vector size (b i.e. a pre-determined range)
- Here, b is nothing but $n_components$

innovate

achieve

lead



Thank You!