Encoding of Business Models by Feature Vectors

Yamini Punetha

November 15, 2021
Version: Final

PADERBORN UNIVERSITY
The University for the Information Society

Department of Electrical Engineering,
Computer Science and Mathematics
Warburger Straße 100
33098 Paderborn

INTELLIGENT
SYSTEMS

Intelligent Systems Group (ISG)

Master's Thesis

Encoding of Business Models by Feature Vectors

Yamini Punetha

1. Reviewer	Dr. Gunnar Schomaker Department of Computer Science Paderborn University
2. Reviewer	Prof. Dr. Eyke Hüllermeier Department of Computer Science Paderborn University
Supervisors	Dr. Gunnar Schomaker and Prof. Dr. Eyke Hüllermeier

November 15, 2021

Yamini Punetha

Encoding of Business Models by Feature Vectors

Master’s Thesis, November 15, 2021

Reviewers: Dr. Gunnar Schomaker and Prof. Dr. Eyke Hüllermeier

Supervisors: Dr. Gunnar Schomaker and Prof. Dr. Eyke Hüllermeier

Intelligent Systems Group (ISG)

Department of Computer Science

Pohlweg 51

33098 and Paderborn

Abstract

Business models are often described as a plan for a company to generate revenue by defining the operations and understanding their customers. For decades, these business models have been created by experts based on their knowledge and experience. As the application of machine learning in real-life problems is gaining immense popularity, it makes sense to develop a system to automate this process.

However, designing a machine learning model that learns from a set of business models comes with its challenges. In our setting, the business models are represented using Business model canvas [OPT10; Ost04]. The business team from the project has developed a taxonomy for different attributes of a business model canvas, which structures business models in the form of categorical data with latent hierarchy. From a machine learning point of view, we expect the input for a model to be in the form of a feature vector. In this thesis, we investigated different approaches to deal with categorical data in a machine learning problem. Another challenge when dealing with business models is that business models are often partial, which means that one or more attributes used to describe a business model are not present. To handle the problem of partial data, we investigated different approaches from the literature for dealing with missing or incomplete data in machine learning problems.

We explored and implemented different encoding methods to overcome the challenges stated above. Extensive experimental results on synthetic and open-source datasets are presented to evaluate the performance of different encoding techniques. The results show that feeding machine learning models with input vectors encoded using Entity Embedding improves the performance of these downstream models. Also, we implemented the feature set embedding that naturally handles the problem of partial data.

Contents

Introduction | 1

In the past 15 years, machine learning (ML) has drastically impacted different domains like science, health care, business, government, etc. [LBH15]. It has proven to be very effective in identifying complicated and detailed structures hidden behind high-dimensional data. The development of complex ML models highly depends on the availability of large datasets. Although artificial intelligence (AI) is making its way through various domains for creating more innovative solutions, the application of AI and researches in the context of business is still growing and has been limited so far [Gul07]. Companies like Airbnb, Uber, etc., have been using AI systems to automate the process of business model innovation [Lee+19]. However, for small to mid-size companies, the idea of innovating a business model is usually limited by the knowledge available within the organization.

Pohle and Chapman [PC06] defined business model innovation as a continuous process of developing and improving a company's business model to account for the changing source of value creation. The already existing business model innovation methods lack the ability to suggest its users, suitable business models actively. The SmartGM[1] project aims to develop an assistance system that can recommend appropriate business models compatible with the specifications provided by the user. This project aims to boost the business model innovation process by introducing an advanced assistance system that learns from existing business models and their corresponding ratings. To this end, the assistance system will utilize the concepts of machine learning and crowdsourcing.

Here, the main idea is to start with a random set of business models which are presented to the crowd workers or users. Then, these users are responsible for rating the provided set of business models. These ratings provided by crowd workers along with the set of business models would act as labeled input for the first machine learning model. Again, the process continues, and new business models are generated randomly or using a machine learning model for the crowd to rate. The iteration goes on until the user is satisfied with the quality of the business models. Business models can be rated using various approaches. One such approach

[1]https://wiwi.uni-paderborn.de/en/dep3/winfo2/forschung/projekte/smart-gm

is choice-based rating feedback, where the user selects all the bad business models. Another popular rating approach is star-based rating feedback, which is likely to be seen in movie ratings.

To develop a machine learning model that takes a business model as an input and predicts rating, it is first important to have a suitable representation of the business model. Business models can be represented using business model languages, for example, the business model canvas [Ost04], e3-value [Hue+08], Casual Loop Diagrams [Kia+09]. In the SmartGM project, business models are represented using the business model canvas. The business model canvas is a visual representation of a business model that depicts a company's value, customers, revenue, and architecture. It consists of 9 building blocks, i.e., key activities, key partners, key resources, value propositions, channels, customer segments, customer relationship, cost structure, and revenue streams.

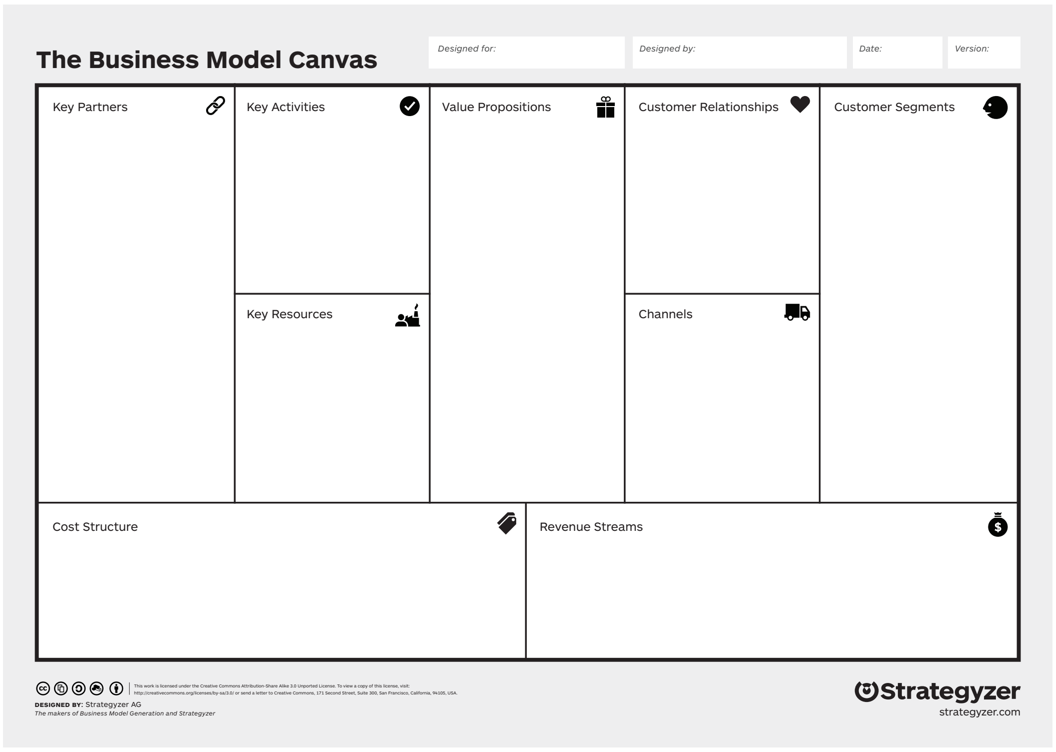


Fig. 1.1.: Business model canvas template by *Strategyzer*[2]

In general, users manually create business models using the business model canvas, which results in unstructured free text describing each building block. It is challenging to train a machine learning model that learns from such unstructured data, which is often difficult to analyze. Hence, the first and foremost step in the SmartGM project was to define a taxonomy for business models. The defined taxonomy provides a hierarchical structure to the business model attributes. By using this taxonomy to create a business model, different attributes of business models will be structured in the form of categorical data with some latent hierarchy. It introduces the two key challenges in learning from business model data in a machine learning problem. The first key challenge is to deal with categorical data as an input for a machine learning model. Also, business models are often described by a subset of all the possible attributes, introducing the problem of partial data. Therefore, the second key challenge here is to deal with partial data in a machine learning problem.

1.1 Problem Statement

In the last section, we identified two main challenges when dealing with business models in a machine learning problem, i.e., dealing with the categorical data and feature sets. This thesis aims to explore and evaluate different methods from literature to address the identified challenges. As mentioned in the above paragraphs, we are currently using a business model canvas and the defined taxonomies to describe a business model. Even with the fully defined taxonomies, each attribute expressing a business model will be structured in the form of categorical data with latent hierarchy. This is problematic since dealing with categorical data, let alone with hierarchical categorical data, is a challenging task for a machine learning model because machine learning models are mathematical models that requires the input to be in numerical format. Hence, there is a need to explore different approaches to encode categorical data into numerical vectors.

On the other hand, when we talk about business models, it is crucial to understand that business models are described by a subset of all the possible attributes. It implies that one or more attributes to define a business model are not present. Note that this situation is different from a typical incomplete data regime because this missingness is often not random but has some hidden meaning. For example, there might be some business model for which some features are incompatible, or some features could be mutually exclusive for some models. We can consider these business models as partial or feature sets. Hence, we want to explore different techniques that are able to handle feature-set-based instances as an input for learning models.

It is important to note that the business model dataset is not available as we conduct the experiments. The availability of business model data depends on the business team responsible for defining the final taxonomy and then creating different business models using the taxonomy. As a workaround, an important task is to identify other suitable datasets with categorical features or generate synthetic data to evaluate different approaches to tackle the critical challenges identified above: to deal with categorical data and feature sets.

To sum up, the main goal of this thesis is to explore and evaluate different methods for encoding categorical data and dealing with feature-set-based instances.

1.2 Objective

Three principal objectives can be derived to achieve the main goal of this thesis, as defined in section 1.1. The first objective is to find suitable datasets for evaluating and testing different approaches from the literature. The project is still at a stage where the actual business model dataset is not available. Hence, it is vital to identify datasets with similar characteristics to perform the experiments. The second objective is to investigate different approaches to deal with categorical data. This research will include implementing various approaches for encoding categorical features in machine learning problems and further comparing them to outline the most suitable approach. To evaluate how different implementations perform, several downstream models will be set up. Here, a downstream model depends on the dataset used, and the problem solved using the corresponding dataset. For example, the encoded vectors will act as an input for various classification models for a classification dataset. Along with implementing various approaches for each dataset, individual preprocessing, feature engineering, model creation, and hyperparameter optimization steps will be implemented.

As discussed above, there is also a need to explore an approach to work with partial data as input for machine learning models. Therefore, the third objective is to explore and implement various approaches to handle partial data.

1.3 Thesis Structure

Chapter 2 — Background

This chapter lays some basic information about business models canvas. This chapter also includes introduction to topic like regression problem, categorical variable and their impact on machine learning problems. Further, this chapter will shed some light on model selection and hyperparameter optimization.

Chapter 3 — Related Work

After discussing the background, related work significant for handling categorical data as well as partial data in machine learning problems is discussed in detail. This chapter is written after a systematic literature survey.

Chapter 4 — Implementation

This chapter explains the implementation process to evaluate different approaches from the literature. This chapter also outlines the experimental setup and datasets used for the experiment.

Chapter 5 — Evaluation

After discussing the experiment setup in the last chapter, this chapter will present the results from the experiments. The results from various experiments are evaluated and conclusions are drawn.

Chapter 6 — Conclusion

The last chapter summarizes the carried out research work in this thesis. Possible directions for future work are indicated.

Background

2

Before moving to related work and the actual implementation carried out in this thesis, preliminary concepts are discussed in this chapter. It includes a detailed description of business models and the business model canvas. Later sections of this chapter cover an overview of supervised learning, the problem of model selection, and hyperparameter optimization. Next section 2.5 describes the problem faced while learning from categorical data along with methods to overcome this issue. The last section 2.6 explains the feature set embedding in detail.

2.1 Business Model and Business Model Canvas

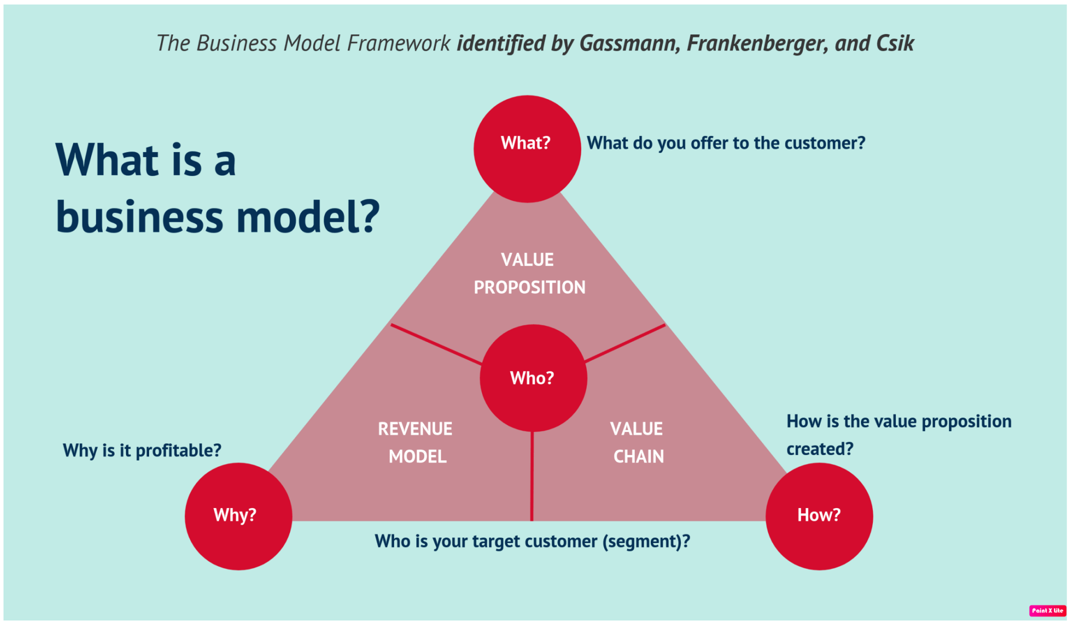

The Business Model Framework identified by Gassmann, Frankenberger, and Csik
What is a business model?
What?
What do you offer to the customer?
VALUE PROPOSITION
Who?
REVENUE MODEL
VALUE CHAIN
Why is it profitable?
How is the value proposition created?
Why?
How?
Who is your target customer (segment)?

Fig. 2.1.: Business model framework inspired by Gassmann et al. [GFc14]

Teece and Linden [TL17] defines a business model as a concept that articulates a logic to demonstrate how a business idea can create and deliver value to its customers. It also explains the architecture of revenue, cost, and profit while delivering the value. A business model is an essential tool required to run a profitable business in a next-generation competitive market. Generally, a business model has four components [AA10], value proposition, value creation architecture, partner network, and revenue system. Figure 2.1 represent different aspect of business model along with the questions answered by each aspect. For example, value proposition defines the product or service offered to the customer to generate revenue.

For any business to run successfully, while meeting the customers need and generating profit simultaneously, it is very important for companies to develop innovative business model. Business models are usually created manually by experts. One uses a business model language to represent a business model. There are multiple business model languages that are used by companies to create their business models. For example, Business model canvas is a template used to design business models. It can also be used to document an existing business model. A business model canvas is a combination of nine building blocks. These nine building blocks can be grouped in 4 major segments.

Infrastructure. This segment includes three key building blocks of a business model.

1. **Key Activities:** This attribute consists of all the important actions that a company requires to operate successfully. For example, for a software company like Microsoft, key activity is software development.

2. **Key Resources:** It defines what are the most important resources for the business model to work. The value is highly dependent on the type of business model. For example, key resources can be humans, financial or intellectual aids.

3. **key Partners:** It includes all the partners associated with the company. Many companies form allies when operating a business to reduce the risk and increase innovation and productivity.

Offering. This segment includes value proposition associated with the company.

1. **Value Proposition:** It defines the products and services that a company provides for each customer segment. It is the attribute that distinguishes a company from its competitors.

Customers. The Customer group consist of all the building blocks that contain all the necessary information about the targeted customer segments.

1. **Customer Segments:** Customer Segments attribute is a set of different customers. To run a successful business, it is very important to identify the customer base and group them specifically based on their needs and characteristics. It is vital to understand that different strategy is needed for different customer segments.

2. **Customer Relationships:** This building block defines the relationship a company establishes with each customer segment. It is an important aspect of a business model which can have a huge impact on customer experience.

3. **Channels:** Channels building blocks describe how a company interacts with each customer segment to deliver its value proposition.

Finances. The finance group includes two major building blocks of a business model that define the cash flow in and out of a company.

1. **Cost Structure:** This building block details all the costs involved to operate a business. It includes all the fixed and variable costs required to deliver the value proposition.

2. **Revenue Stream:** Revenue Stream building block outlines how a company makes money from each customer segment. Cost structure along with revenue stream is used to calculate the profit/loss that a company makes.

The figure 2.2 shows a business model for Google using business model canvas. The figure illustrates all the essential factors of a business model, including the customer information, service offered by the company, and the revenue stream. However, the figure suggests that the values of different segments in a business model are just free text. As mentioned in section 1, a taxonomy is designed by the business team to structure the business model data in the form of categorical features with latent hierarchy. The business model created using the taxonomy makes the business model data more structured, which is much more convenient for a machine learning model. Designed taxonomy solves the problem of unstructured data, but it introduces a new challenge while dealing with business models. Since the data is in the form of categorical features, such features cannot be directly fed to a machine learning model. Categorical features are transformed into a vector

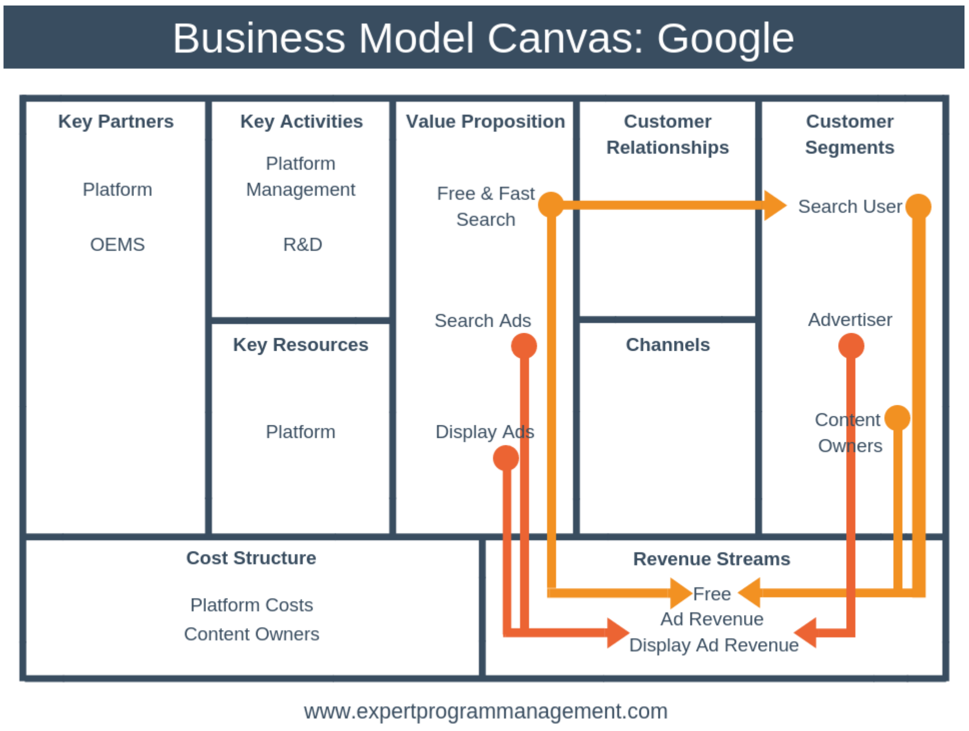


Fig. 2.2.: The Figure illustrates the business model for Google. This representation of a business model makes it easier for the experts to understand the process. It clearly shows that Google builds its revenue from Advertiser customer segment through the ads displayed on search engine.

to overcome this problem. Therefore, one of the key challenges addressed in this thesis is to explore and evaluate different methods to encode categorical features into a vector space. Also, the business model canvas represents a business model as a feature set, where sometimes not all features are necessary to define a business model. It introduces the other challenge that is addressed in this thesis work, to deal with feature sets.

2.2 Supervised Learning

The main goal of this thesis is to identify and evaluate different methods to encode categorical data into feature vectors and deal with feature sets. Different encoding techniques are implemented using five different datasets, four open-source datasets, and one synthetic dataset generated via python code to achieve this goal. All the selected datasets for the experiments are labeled. Therefore, this is a traditional supervised learning problem, also known as learning from examples. In supervised learning, the training dataset is a combination of an input space $\mathcal{X}$ and an output space $\mathcal{Y}$. The aim of supervised learning is to learn a mapping, often called the target function, f from $\mathcal{X}$ to $\mathcal{Y}$.

$$f : \mathcal{X} \to \mathcal{Y}$$

The learning Algorithm $\mathcal{A}$ returns a hypothesis $h \in \mathcal{H}$ that fits the training dataset

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N} \in (\mathcal{X} \times \mathcal{Y})^N \tag{2.1}$$

The resultant hypothesis is represented as $g : \mathcal{X} \to \mathcal{Y}$. It is further used to make

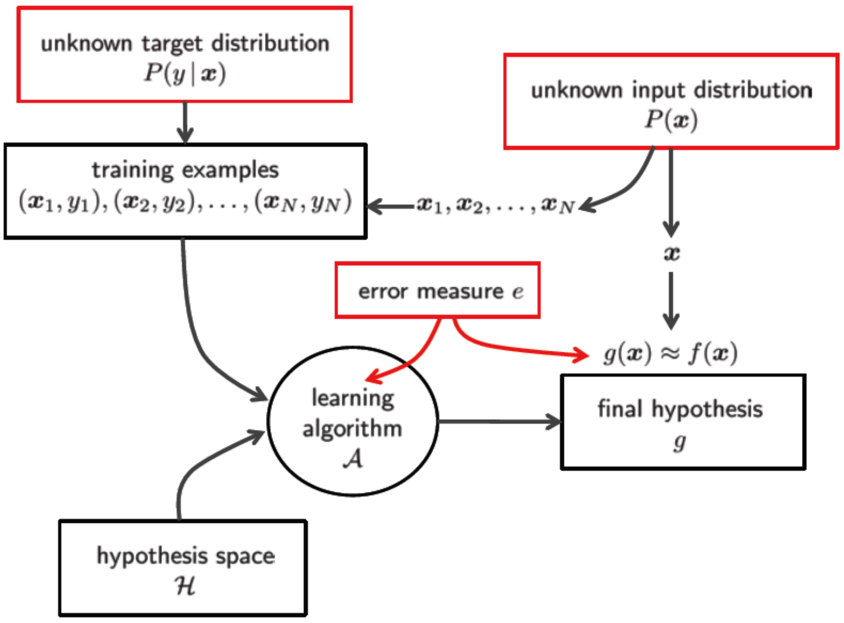

unknown target distribution
$P(y \mid \boldsymbol{x})$
unknown input distribution
$P(\boldsymbol{x})$
training examples
$(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)$
$\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$
$\boldsymbol{x}$
error measure e
$g(\boldsymbol{x}) \approx f(\boldsymbol{x})$
learning algorithm
$\mathcal{A}$
final hypothesis
g
hypothesis space
$\mathcal{H}$

Fig. 2.3.: The general Supervised Learning Problem [AML12]

prediction on test data. Closer the g is to the original target function f, the better are the predictions made. The final hypothesis g is selected from a set of hypothesis, $h \in \mathcal{H}$. To assess the quality of a hypothesis h, risk or out-of-sample error of the hypothesis is calculated. Out-of-sample error is defined as the prediction error of a hypothesis on unseen data. However, in practice, it is not possible to compute the out-of-sample error. Instead, in-sample error, also called empirical risk of h is computed as follows:

$$E_{in}(h) = \frac{1}{N} \sum_{i=1}^{N} e\left(y_i, h(x_i)\right) \tag{2.2}$$

The optimal hypothesis g that estimates the target function f is identified by Empirical risk minimization. It suggests finding a hypothesis with the minimum empirical risk and is represented by the following equation.

$$g \in arg \min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} e\left(y_i, h(x_i)\right) \tag{2.3}$$

2.3 Methods of Model Selection

As a part of implementation in this thesis, various models are fitted with different types of feature vectors. In different experimental settings, the performance of the various supervised learning models, like Random Forest, Logistic regressor, etc., is evaluated. It is essential to evaluate how the fitted model performs on unseen data when training a machine learning model. Out-of-sample error or risk is a measure of a model's capability to perform on previously unseen data. It is defined as follows:

$$E_{out} = \int_{\mathcal{X} \times \mathcal{Y}} e\left(y, h(x)\right) d\, P(x, y) \tag{2.4}$$

where e is the loss function and $P(x, y)$ is the unknown joint probability distribution for x and y.

There are many ways to determine how a trained model performs on the out-of-sample data. One such technique is called Resampling, which is used to estimate the out-of-sample error to analyze the performance of a model. Some of the most commonly used resampling techniques are as described below.

2.3.1 Train-test split

Here, the input data is split into train and test sets. The model is trained using the training set, and the model's performance is evaluated on the test set. It ensures that an unseen set of data is provided during the model evaluation to observe relatively unbiased results. *train_test_split* [1] is a commonly used library for the same.

[1]library provided by Scikit-learn [Ped+11]

2.3.2 Cross-Validation

Another most commonly used resampling technique when evaluating the performance of a learning model is Cross-Validation. Introducing cross-validation into the training process can explain how the model generalizes and computes more reliable expected errors on out-of-sample data. The main idea is to execute the traditional validation technique N/k times. The validation is performed on a different k data points and the average is presented as the final result. Cross-validation proofs to be very advantageous when the provided training dataset is small. Multiple cross-validation techniques can be used to measure estimated error, for example, k-fold cross-validation, Stratified cross-validation, leave one out cross-validation, repeated cross-validation, and nested cross-validation.

1. **K-Fold** is a cross validation variation where, the full training dataset is shuffled and then divided into k sets. The validation process runs through multiple iteration, where k^{th} set is used for validation while the remaining sets are combined to train the learning model. K-fold cross validation splits the complete dataset $\mathcal{D}$ of size N into K folds $\mathcal{D}_1$, $\mathcal{D}_1$,...., $\mathcal{D}_K$. The size of each fold is N/K. The k-fold cross validation trains K hypothesis

   $$g_1^-, g_2^-, ..., g_K^-$$

   Each hypothesis g_k is trained on $\mathcal{D} - \mathcal{D}_k$ data and validation error $E_{val}(g_k^-)$ is calculated on $\mathcal{D}_k$. The average of these k validation error results in the final cross validation error depicted by the following formula.

   $$E_{cv_k} = \frac{1}{K} \sum_{k=1}^{K} E_{val}(g_k^-) \qquad (2.5)$$

2. **Stratified K-Fold** is an ideal cross validation technique for classification task. In this case, the training dataset is grouped in to k folds in such a way that each fold has the same ratio of different classes as in the training set. If the output space $\mathcal{Y}$ consist of l classes such that $y_1, y_2, ..., y_l \in \mathcal{Y}$, the stratified K-fold splits the dataset $\mathcal{D}$ into K folds such that the target classes are distributed evenly for each fold. Stratified K-Fold validation provides more accurate result and the training process is less biased.

3. **Leave one out** is another holdout cross validation technique where only one data point is left out from the training process and used for validation. It is also called LOOCV. LOOCV leaves one data point out for validation from the dataset $\mathcal{D}$ of size N and uses $N-1$ data points for training the model. The Leave one out cross validation error is defined as follows:

$$E_{cv} = \frac{1}{N}\sum_{n=1}^{N} E_{val}(g_n^-) \tag{2.6}$$

4. **Repeated Cross Validation** is similar to general K-fold cross validation. However, in this case, the procedure is further repeated n-times. Before each repetition, the complete data set is shuffled and then new splits are generated. Let us assume, that repeated cross validation is performed with J repetitions. In this case, the K-fold cross validation is performed j times and the final cross validation error for repeated cross validation is defined as:

$$E_{cv_{RCV}} = \frac{1}{J}\sum_{j=1}^{J} E_{cv_k} \tag{2.7}$$

using equation 2.5:

$$E_{cv_{RCV}} = \frac{1}{J}\sum_{j=1}^{J}\frac{1}{K}\sum_{k=1}^{K} E_{val}(g_k^-) \tag{2.8}$$

5. **Nested cross validation** is another well-known technique for cross validation. This technique comes very handy when there is a need to evaluate the predictions made by a learning model. Nested cross validation is often used for hyperparameter optimization.

2.4 Hyperparameter Optimization Techniques

Every machine learning model has some parameters that are adjusted manually when training the model for a particular dataset [FH19]. These parameters are called hyperparameters. Hyperparameters are different from general parameters of a model, which are learned during the training process. Instead, hyperparameters could be considered as a model's configuration that need to be tuned according to the dataset. This is a very important step in a machine learning project so that the performance of a model can be optimized for a dataset. Therefore, hyperparameter optimization or hyperparameter tuning is a technique where different values for model's hyperparameter are provided as input and resultant is a model with best performing hyperparameter for the given dataset.

Let $\mathcal{A}$ be a machine learning algorithm with N hyperparameters. If the n^{th} hyperparameter is denoted by Γ_n then the complete hyperparameter configuration space can be represented as $\Gamma = \Gamma_1, \Gamma_2, ..., \Gamma_N$. Also, γ is a vector of hyperparameters such that $\gamma \in \Gamma$, then a machine learning algorithm $\mathcal{A}$ enabled with γ is denoted by $\mathcal{A}_\gamma$.

The goal of hyperparameter optimization is to find γ^*, vector of hyperparameters that minimizes the loss function of $\mathcal{A}$ that trains on $\mathcal{D}_{train}$ and calculates validation loss on $\mathcal{D}_{val}$

$$\gamma^* = agr \min_{\gamma \in \Gamma} E_{(\mathcal{D}_{train}, \mathcal{D}_{val})}\ V\ (L, \mathcal{A}_\gamma, \mathcal{D}_{train}, \mathcal{D}_{val}) \tag{2.9}$$

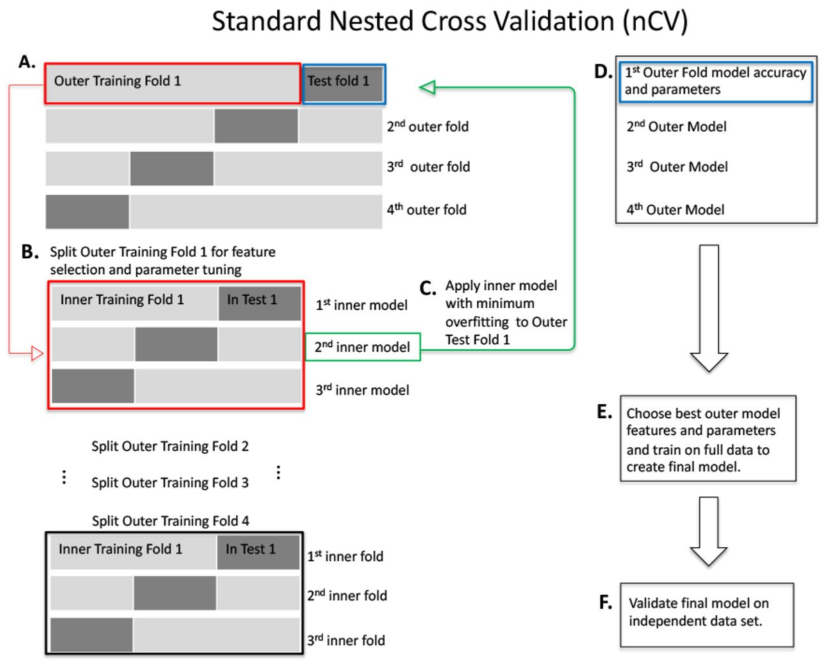
where V is the loss measure of $\mathcal{A}$ with γ initiated as hyperparameter trained on $\mathcal{D}_{train}$ and validated on $\mathcal{D}_{val}$.

The most commonly used optimization algorithms for hyperparameter tuning are random search and grid search. Random search is an optimization technique where random values for hyperparameters are selected from a grid of possible values for hyperparameters. Finally, the model's performance is evaluated on the selected hyperparameter. The result is the set of hyperparameter values that gives the best performance. The main disadvantage of random search is that it might not return the best performing parameters from the grid since only a specific number of random selections are made for the hyperparameter values. Another commonly used optimization technique is Grid search. In the case of the grid search, the model fits every combination of hyperparameter values and evaluates the model's performance on the selected hyperparameters.

Scikit-learn includes multiple implementations for performing hyperparameter optimization for a model. For example, RandomSearchCV and GridSearchCV.

2.4.1 Nested Cross Validation for Hyperparameter Optimization

Nested cross-validation is an excellent approach for hyperparameter optimization. Generally, a single k-fold cross validation trains a model on all the folds except one, and the hold-out fold is used to evaluate the model's performance. The collection of all the folds used for training is called the training dataset, and the hold-out fold is called the test dataset. Nested cross-validation includes an additional loop where the training data set is again divided into k-folds, which are used for the hyperparameter optimization task. Using nested cross-validation reduces the chances of overfitting and provides less bias as the performance of the tuned model(done in the outer loop) is evaluated on unseen data.



Standard Nested Cross Validation (nCV)
A.
Outer Training Fold 1
Test fold 1
2nd outer fold
3rd outer fold
4th outer fold
B. Split Outer Training Fold 1 for feature selection and parameter tuning
Inner Training Fold 1
In Test 1
1st inner model
2nd inner model
3rd inner model
C. Apply inner model with minimum overfitting to Outer Test Fold 1
D. 1st Outer Fold model accuracy and parameters
2nd Outer Model
3rd Outer Model
4th Outer Model
Split Outer Training Fold 2
Split Outer Training Fold 3
Split Outer Training Fold 4
Inner Training Fold 1
In Test 1
1st inner fold
2nd inner fold
3rd inner fold
E. Choose best outer model features and parameters and train on full data to create final model.
F. Validate final model on independent data set.

Fig. 2.4.: Standard Nested Cross Validation [Par+20]

Nested cross-validation has its advantages as it provides a better estimate of out-of-sample error and reduces the bias. However, it also comes with a considerable increase in computational cost. For example, if we want to train a model with 20 combination of hyperparameter and k= 5 for hyperparameter optimization, the total model to be fitted will be $5 \times 20 = 100$. However, for the outer loop if k=10, then the total number of models to be fitted will be $5 \times 20 \times 10 = 1000$.

2.5 Categorical features in machine learning

One of the key challenges identified in this thesis is to deal with categorical features in a learning problem. In general, a neural network is capable of approximating any continuous function [Cyb89; Nie18]. However, structured data with categorical features lack continuity, making it difficult for a neural network to approximate non-continuous categorical features. There are different techniques to encode categorical features into a numerical vector space so that machine learning algorithms can use these features for learning. This thesis uses three different techniques to encode categorical features into vector space: label encoding, one-hot encoding, and entity embedding. The following sections define these encoding methods in detail.

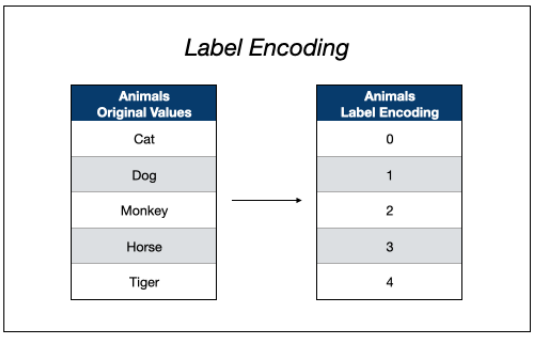


Fig. 2.5.: Figure illustrates a categorical feature encoded by label encoding

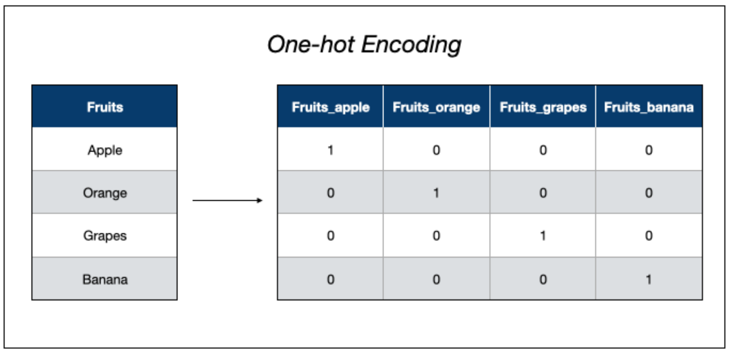
2.5.1 Label Encoding

Label encoding is a deterministic encoding technique where integer values represent different categories. For example, for a feature animal, the possible categorical values are {'cat','dog','horse'} mapped to labels {0,1,2} respectively. Label encoding is the simplest technique to convert categorical variables into numerical values. These numerical values can be fed to various learning models for learning. However, such representation could be misleading for features where categories are independent. For example, in the above example, the learning algorithm would consider the encoded label 0, 1, 2 as a sequence of numbers where the distance between 0 and 2 is greater than 0 and 1. Figure 2.5 shows an example of label encoding where a categorical feature of size five is encoded using a label encoding. The resultant encoded column holds the integer value from 0 to 4 for each categorical value.

2.5.2 One-Hot Encoding

One-hot encoding is a popular encoding technique used to encode categorical features as a preprocessing step. N new binary columns are added to the dataset for each categorical feature with N categorical value using one-hot encoding. Value 1 for the column indicates the presence of the category, while 0 indicates its absence. Although this technique adds certain continuity to the categorical features, it increases the number of columns in the dataset depending on the number of categories for categorical features. If the number of possible categories is very large, then the increased size of the feature vector results in high computation requirement. Also, one-hot encoding ignores any relation between different categorical values and treats each value independently. The above two shortcomings of one-hot encoding inspire to explore more ways to encode categorical features.

Figure 2.6 depicts an example of a categorical feature *Fruits* with 4 different categories encoded using one-hot encoding. One-hot encoding results in 4 new columns are added to the structured data.



Fruits
Apple
Orange
Grapes
Banana

Fruits_apple	Fruits_orange	Fruits_grapes	Fruits_banana
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Fig. 2.6.: Figure illustrates a categorical feature encoded by one-hot encoding

2.5.3 Entity Embedding

Entity embedding is a technique introduced by Guo and Berkhahn [GB16] to map categorical variables into Euclidean spaces using a neural network. The embeddings are learned in a supervised setting. For a given input set, $(x_1, x_2, ..., x_n)$ a learner predicts the target value y.

$$y = f(x_1, x_2, ..., x_n) \tag{2.10}$$

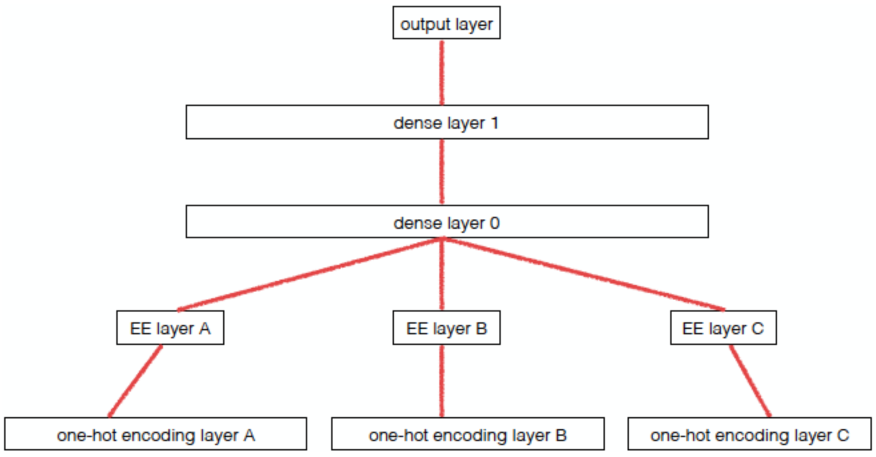


Fig. 2.7.: Implementation of Entity Embedding, where for each categorical feature, an embedding layer is added on top of one-hot layer. [GB16]

To get the approximation of f, each categorical feature x_i is mapped to an embedding space such that,

$$e_i : x_i \mapsto X_i \tag{2.11}$$

For each categorical feature, an extra embedding layer is added on top of a one-hot encoded vector. The one-hot vector of a categorical feature x_i with m_i unique value is represented as:

$$u_i : x_i \mapsto \delta_{x_i \alpha} \tag{2.12}$$

where $\delta_{x_i \alpha}$ is a Kronecker delta for x_i. The length of $\delta_{x_i \alpha}$ is m_i and the element of this vector is non-zero when $\alpha = x_i$. The output of the embedding layer for x_i can be given by:

$$X_i = \sum_{\alpha} w_{\alpha\beta} \delta_{x_i \alpha} = w_{x_i \beta} \tag{2.13}$$

where β is the index of embedding layer and $w_{\alpha\beta}$ is the weight connecting one-hot encoding and entity embedding layer. Therefore, adding an embedding layer introduces another parameter in the neural network setting that can be learned the same as the other parameters. For each categorical feature, an embedding layer can be added to the neural network. Next, all the layers are merged such that the merged layer acts as a typical neural network layer.

2.6 Feature set embedding

Grangier and Melvin [GM10] proposed a method to learn a classifier g that predicts a class for an input set X identified as a set of feature-value pair. For a dataset $D = (X_i, y_i)_{i=1}^{n}$, each instance is represented as (X, y), where X is a set of (feature, value) pair defined as:

$$X = (f_i, v_i)_{i=1}^{|X|}$$

and y denotes the class label such that $y \in \mathcal{Y} = \{1, ..., k\}$. The proposed feature set embedding works for both discrete and continuous feature values, i.e., $v_i \in \mathcal{V}_{f_i}$, where $\mathcal{V}_{f_i} = R$ for continuous features and $\mathcal{V}_{f_i} \in \{1, ..., c_{f_i}\}$ for discrete features. However, the set of features σ_i is discrete, such that $f_i \in \{1, ..., d\}$.

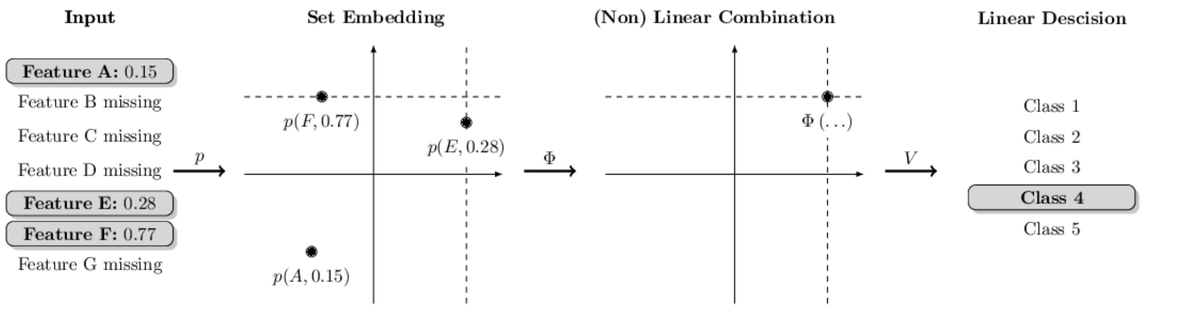


Fig. 2.8.: Feature set embedding for a set of (feature, value) pairs [GM10]

Feature set embedding is a two level architecture. Figure 2.8 illustrates the implementation of feature set embedding for a set of (feature, value) pairs. At the first level, Set Embedding, the feature value pairs are mapped to an embedding space of dimension m. For an instance $X = \{(f_i, v_i)\}_{i=1}^{|X|}$ with feature value pair (f_i, v_i), a function p predicts the embedding vector $p_i = p(f_i, v_i) \in \mathcal{R}^m$. At the second level, the embedding vectors are combined using a linear(mean) or a non-linear(max) function to make class predictions. Here, a function h takes $\{p_i\}_{i=1}^{|X|}$, set of embedded vector, as an input and predicts a vector $h(\{p_i\}_{i=1}^{|X|}) \in \mathcal{R}^k$. In the predicted vector, the correct class for the instance $X = \{(f_i, v_i)\}_{i=1}^{|X|}$ is assigned the highest value. Therefore, the proposed classifier can be described as a two level classifier represented as:

$$g = p \circ h \tag{2.14}$$

Feature set embedding is a promising approach because of two main reasons. First, it treats the input as a set of (feature, value) pairs, allowing it to deal with partial datasets naturally. On the other hand, feature set embedding is a flexible method, as it could be used for both continuous and discrete features simultaneously.

Related Work 3

As mentioned before, the main goal of the thesis is to explore and implement methods for encoding categorical variables and handling feature-sets-based instances in a machine learning problem. To achieve this goal, a systematic literature review was conducted. In the following chapter, related work regarding categorical data and partial data is reviewed. The first section lays out the approaches for processing categorical data for machine learning problems. In the later section, different methods to work with partial or missing data from literature are discussed.

3.1 Encoding of Categorical data

There has been a growing body of research on how to represent categorical data as numerical vectors. Hancock and Khoshgoftaar [HK20] did a survey on different approaches to encode categorical data for neural networks. They divided different techniques to deal with categorical data into three major categories, i.e., the determined, algorithmic, and automatic techniques. Determined techniques involve low-level computation methods to convert the categorical variable into vector form. The encoded values for the categorical data are always the same in the case of the determined technique. An example of a determined technique is one-hot encoding. They argue that one-hot encoding does not work well for categorical data because different categorical variables are treated very differently, even if they are closer in terms of semantics. Another problem with using the one-hot encoding is that with the increase in cardinality, the size of the output vector increases [HK20]. Another example of a determined technique is label encoding. It is a simple technique of assigning an integer value to each possible category group.

Algorithmic techniques usually have more complicated computations and work as pre-processing of data. In such techniques, the researchers develop algorithms to convert categorical data into numerical values. Golinko and Zhu [GZ19] introduced an algorithmic technique called generalized feature embedding, or GEL. The goal here is to achieve a numerical representation of each instance such that it could be used further for supervised and unsupervised tasks. GEL considers the instance of a dataset as a matrix $\mathcal{X}$. The first step is to transform $\mathcal{X}$ into a binary representation $\mathcal{W}$, which is achieved by using traditional one-hot encoding. For a supervised task, GEL used class labels from the training data to create instance partitions. This process is called Class Partitioned Instance feature Representation or CPIR. After a series of matrix operations, the embedding matrix $\mathcal{V}$ is delivered. Finally, $\mathcal{F}$ can be used as an input for various supervised learning tasks.

$$\mathcal{F} = \mathcal{W} \times \mathcal{V}^2$$

Lastly, Hancock and Khoshgoftaar [HK20] define automatic techniques as the approaches where the task of encoding categorical features is included in the neural network's learning process. Entity embedding introduced by Guo and Berkhahn [GB16] is an example of an automatic technique that is used to map categorical variables into a vector space. In their research, the authors have first used one-hot encoding on the input data. Then an additional embedding layer is added for each categorical feature to the neural network. This embedding layer transforms the corresponding categorical variable into real-valued vector space. After adding the embedding layers for all the categorical features, the output is concatenated and fed to dense layers. It can be extended to a neural network for learning or can be used for transfer learning. Ma and Zhang [MZ20] have used an entity embedding inspired by Guo and Berkhahn [GB16] to create a predictive model for travel mode choice. The authors have incorporated the encoding of categorical features using entity embedding in the deep neural network for prediction. The results show that using an entity embedding instead of other encoding methods like one-hot encoding or label encoding with a deep neural network shows better results.

Mikolov et al. [Mik+13] introduced two model architectures to learn distributed word representations to reduce the computational complexity. The first architecture is the continuous bag-of-words model, or CBOW, which works like a feed-forward neural net language model. Nevertheless, instead of using non-linear hidden layers in the network, a log-linear classifier predicts a word based on four past and four future words. Another implemented architecture by Mikolov et al. [Mik+13] is the continuous skip-gram model, which is similar to CBOW architecture. However, instead of predicting a word based on the surrounding context word, it predicts the context words based on the current word.

Another study by Pennington et al. [PSM14] proposed a new log-bilinear regression model called GloVe that transforms the word representation into a vector space.

The model is trained only on nonzero elements of the word-word co-occurrence matrix. The GloVe model is capable of encoding a word into vector space along with a meaningful structure.

Meulemeester and Moor [MM20] studied the impact of using unsupervised natural language processing embeddings for categorical variables on classification tasks. They used natural language processing to transform categorical variables into vector representations. The authors extended the implementation of neural embedding for categorical data and further compared the performance of the CBOW, skip-gram, and GloVe model on classification tasks. This research shows that using CBOW and skip-gram model for encoding categorical data failed to capture the semantic relation of the categorical variables. However, the GloVe model performed well while depicting the relation and the distance between the categorical variable in the embedded space.

In the field of natural language processing, Sutskever et al. [SVL14] provides an insightful approach of using multilayered long-short term memory (LSTM) for implementing sequence learning. Interestingly, the authors have used one LSTM to map the input sequence to a fixed-sized vector and then another LSTM to map the fixed-sized vector to the output sequence. Using LSTM to generate fixed-size vector representation could be a promising approach because of its ability to learn on data with long temporal dependencies.

3.2 Learning from Partial Data

As described in 1.1, business models are often described by a subset of all the possible features, resulting in partial data. Therefore, a systematic literature review was done to collect relevant work to deal with incomplete or partial data in machine learning problems. Imputation is a very popular technique, where the missing values are replaced with alternate values. In some cases, this alternate value can be a constant, zero, or an average of other feature values. Batista and Monard [BM02] and Murti et al. [Mur+19] used k-nearest neighbor to impute the missing data for the machine learning problem. In [BM02], results show that using the k-nearest neighbor as imputation method outperforms the probabilistic approach to treat missing data.

In another research, Hassan et al. [Has+09] have imputed the missing values based on their probability distribution function. By repeating the process, several complete datasets are created. Then all the complete datasets are used to train a neural network, creating an ensemble of networks. Tresp et al. [TNA94] presented an interesting approach to deal with missing data in the training process. The researchers have used the Parzen window to approximate the missing values within the input data. The backward propagation step is used to get a close estimation of missing data from the available known data in the input space. Ghahramani and Jordan [GJ93] have introduced a framework to learn from an incomplete dataset. Here, the authors have used an Expectation-Maximization algorithm to estimate the data distribution over the incomplete dataset.

Apart from imputation-based methods, there are multiple researches done in the past that used different techniques to eliminate the need for imputation to learn from partial data. Śmieja et al. [Śmi+18] have used a neural network to process incomplete data. They have proposed using probability density functions to deal with the uncertainty related to the missing value, which excludes the need for data imputation. A neural network can be trained with partial data using this approach. It can be extended and used for different neural networks. Danel et al. [Dan+20] have used a spatial graph convolutional network to represent an image with missing pixels as a graph and further use it for training.

In a different study, Śmieja et al. [Śmi+19] have used generalized radial basis function or genRBF to model the uncertainty around missing data in input space. In their implementation, firstly, the incomplete data is embedded in a subspace. Next, Gaussian estimation is used to perform probability density distribution on missing data. Finally, classic RBF is extended with a proposed kernel function based on the scalar product between two missing data points. The proposed genRBF is used for classification tasks on different incomplete datasets from UCI library [DG17].

Grangier and Melvin [GM10] have introduced a new method called the feature set embedding to deal with partial data. In this approach, each instance is considered a (feature, value) pair that is then mapped into an embedded space. These embedded pairs are combined into a scalar vector and then linearly classified. This approach is capable of handling partial data without any imputation.

4 Implementation

Chapter 1 discussed the challenges faced when training a supervised machine learning model that takes business models as input and predicts rating. The main goal of this thesis is to evaluate different methods to encode categorical data and feature-set-based instances. To achieve the final goal, a systematic literature review was conducted as described in chapter 3. Different encoding techniques are then selected for implementation to overcome the challenges identified in previous chapters. We implemented three methods to encode categorical data into a vector space, label encoding, one-hot encoding, and entity embedding. However, feature set embedding was implemented to encode instances represented as a set of (feature, value) pairs. The implementation details of the above-mentioned encoding methods are discussed in section 4.1 and 4.2.

As stated in chapter 1.1, the business model dataset is not available yet for evaluating different encoding techniques. Therefore, four different open-source datasets are selected to design experiments to evaluate the above-stated encoding methods. Also, synthetic datasets with different configurations, i.e., number of features and percentage of missing features, are generated to evaluate the implemented feature set embedding. Section 4.3 outlines the selected criteria along with all the details for used datasets.

Once different methods for encoding categorical features and feature-set-based instances are implemented, the next step is to evaluate these encoding. To do so, various experiments are conducted as a part of this thesis work. These experiments are designed on classification and regression datasets for evaluating label encoding, one-hot encoding, and entity embedding. Various experiments are designed such that for each dataset, the categorical features are encoded using different encoding methods, i.e., label encoding, one-hot encoding, and entity embedding. The encoded vectors are then fed as an input to different machine learning models depending upon the type of datasets to evaluate how the implemented methods affect the performance of these learning models.

A different set of experiments are designed to evaluate the implemented feature set embedding. While evaluating feature set embedding, we want to answer the three main questions. First, how accurately does the classifier performs when dealing with the partial feature? Second, does the percentage of missing features in partial data impacts the model's performance? And finally, does the number of features in a dataset impacts the performance of the feature set embedding model? To answer these questions, a series of experiments are performed on programmatically generated datasets with different settings. Section 4.4 gives a detailed overview of all the experiments conducted in this work.

4.1 Approaches to deal with Categorical data

Chapter 3.1 highlights various previous works that either introduced new approaches or evaluated existing approaches to handle categorical data in the input dataset. In this research thesis, three different methods from literature are selected to encode categorical data. Chapter 2.5 describes all the selected approaches: label encoding, one-hot encoding, and entity embedding in detail. The implementation details of these methods are discussed in this section.

4.1.1 Implementation of label encoding

In our experimental setup, *Label Encoding* is implemented using Scikit-learn library in python for categorical variables. Scikit-learn provides LabelEncoder[1] transformer that encodes the variable with n categories into values from 0 to n-1.

4.1.2 Implementation of one-hot encoding

The *one-hot encoding* can be implemented for categorical variables using the Scikit-learn library in python. Scikit-learn provides OneHotEncoder[2] transformer that fits an array to categorical values and returns a binary column for each category. Even pandas [tea20] contains a transformer method for one-hot encoding called

[1]LabelEncoder: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
[2]https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

get_dummies[3]. The advantage of using the pandas' get_dummies method is that one can encode more than one categorical feature at a time. Therefore, we have used pandas' get_dummies method to implement one-hot encoding to transform a categorical feature into a feature vector.

4.1.3 Implementation of entity embedding

Section 2.5.3 explains entity embedding in detail. The implementation of entity embedding is inspired by Guo and Berkhahn [GB16]. A neural network is designed to learn the embedding for categorical features using the deep learning framework Keras [Cho+15]. The embeddings are learned in a supervised setting. For each categorical feature, Keras' embedding layer is added to the neural network. The embedding layer results in the embedding vector representing each unique value for a categorical feature represented by $e_i : x_i \to X_i$. For the continuous features which do not require embedding, kera's dense layer is added to the neural network. Next, a concatenation layer is added to the neural network to merge all the inputs. On top of the already added layers, two fully connected layers with 1000 and 500 neurons are added, followed by an activation layer with ReLU as an activation function. Finally, a dense layer with one neuron as output and sigmoid as activation function is used to complete the neural network. When adding an embedding layer for a categorical feature, the parameters input_shape and output_shape have to be defined. The input_shape represents the number of unique values for a categorical feature, and the output_shape defines the embedding dimensions. This neural network can be trained using the backward propagation method to learn the intrinsic property of categorical features. The advantage of using entity embedding to encode categorical features is that it maps the similar entities closer to each other and reduces the memory usage compared to one-hot encoding.

4.2 Approaches to learn from feature-set-based instances

Learning from incomplete data has been an exciting area of research. Even in the case of business models, there is a need to explore and evaluate approaches to

[3]https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html

handle partial business models. As discussed before, business models are sometimes expected to be represented by partial data. There could be multiple reasons behind the missing feature. For example, some features might not be defined by the expert or even do not make sense for a business model. Some features could be mutually exclusive for an instance of a business model. It is important to handle partial data to ensure that the learning algorithm performs efficiently.

In the 3.2 section, related work underlying different approaches to deal with partial data are listed. Different techniques can be classified as imputation-based techniques and non-imputation-based techniques. Although imputation-based techniques work well in various domains, however, for business models, such techniques are not a good option. Instead, using approaches where the neural network or embedding handles the partial data could be a better option. In this thesis work, we have implemented the Feature Set Embedding introduced by Grangier and Melvin [GM10].

4.2.1 Implementation of feature set embedding

Section 2.6 gives a detailed overview of feature set embedding from the original research paper [GM10]. To sum up, *Feature Set Embedding* is a two-level architecture that learns a classifier g which predicts class from target y from each instance of $\mathcal{X}$. Here, each $X = \{(f_i, v_i)\}_{i=1}^{|X|}$, where f_i is the i^{th} feature and v_i is the corresponding feature value. The target Y is a set of class labels. The feature set embedding can handle continuous and discrete features simultaneously. With the proposed two-level architecture, the feature sets are first mapped to an embedding space, and at the second level, the classifier learns from the embedded vector.

In this thesis, feature set embedding is implemented as a neural network using PyTorch [Pas+19] library. The designed neural network for feature set embedding consists of four neural layers. First, an embedding layer[4] is added to the neural network. This embedding layer is used to generate an embedding for (feature, value) pair of each instance X. The second layer is a pooling layer is added on top of the embedding layer to combine the generated embedding vectors. AvgPool2d[5] from PyTorch is used to introduce a pooling layer in the designed neural network. The above two layers constitute the first level of feature set embedding architecture.

[4]https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html#torch.nn.Embedding
[5]https://pytorch.org/docs/stable/generated/torch.nn.AvgPool2d.html#torch.nn.AvgPool2d

At this level, each instance of x represented by $\{(f_i, v_i)\}_{i=1}^{|X|}$, is encoded into an embedding space. Next, two linear layers are added on top of the pooling layer to implement the architecture's second level. The resultant is a vector of size $|Y|$, where the correct class for the corresponding instance is assigned the highest value.

The implemented feature set embedding takes two parameters as input, depending on the dataset used for the classification task. These parameters are the maximum number of possible features $|X|$ for a dataset and the embedding dimension, m. The value of $|X|$ depends on the used dataset. However, the value of embedding dimension m has to be selected to generate embedding vectors. In this work, embedding dimension $m = |X| - 1$. There is a need to explore methods to select an optimal value for embedding dimension. It remains an open question for this thesis.

4.3 Datasets

This section contains information about all the datasets used for experiments. Since the expected dataset representing business models was unavailable to conduct the experiments, we instead identified other datasets from various resources to bridge this gap. Different datasets are selected based on the following criteria:

1. The dataset should have categorical features or a combination of categorical and continuous features.
2. Number of instances, ranging from {5000, 100000} to evaluate different encoding techniques on small and large datasets.

The following datasets are used to evaluate different methods to encode categorical data into vectors:

1. **Mushroom Dataset**[6] used for different experiments in this thesis is obtained from UCI Machine Learning Repository. The dataset represents hypothetical information about 23 species of mushrooms. Here, each species can be classified as edible or poisonous. The mushroom dataset can be used for classification tasks. It consists of 8124 records where 23 categorical features describe each instance.

[6]https://archive.ics.uci.edu/ml/datasets/Mushroom

2. **Airline Dataset**[7] is obtained from OpenML [Van+13] data repository. This dataset provides information about if a particular flight was delayed or not. It can be used for a prediction task, where a learner predicts whether a flight will be delayed or not. The airline dataset contains 539383 records. Each instance of a flight is represented by eight features, out of which four are categorical, three are numeric features, and the Delay feature is treated as the target.

3. **Census Income Dataset**[8] is also obtained from UCI Machine learning repository. The dataset can be used to predict whether a person's income exceeds $50K per year. It contains 48842 records with 14 different features (eight categorical, six numeric).

4. **Rossmann Sales Dataset**[9] is a historical sales data for 1115 Rossmann stores obtained from the Kaggle platform. This dataset can be used for a regression task, where the model predicts the sales for various stores. The Rossmann dataset contains 1017209 records with 18 columns. Along with the original dataset, three additional datasets are also available that give information about the state where the store is located, state names, and the weather data for different states starting from . As the feature engineering step, the additional datasets are combined with the original dataset to optimize the performance of the learning model.

Synthetic Datasets: For evaluating the implemented feature set embedding, synthetic datasets are generated using the implementation by Caiafa et al. [Cai+21]. Train and test sets are generated with number of instances, $I_{train} = 10000$ and, $I_{test} = 1000$ respectively. The algorithm generates a K-sparse data vector $x \in R^{100}$ using a dictionary $D \in R^{100 \times 200}$. The dictionary D is obtained by the Gaussian distribution with normalized $||D(:, j)|| = 1$ for all j. To generate the target class for each vector, a random hyperplane $\{w, b\}$ is selected, where $w \in R^{I}$ and bR. The target label y_i is defined as:

$$y_i = \langle w, x_i \rangle + b$$

The generated data is masked to introduce partial features. Different datasets are generated by adjusting two main settings: the number of features N and the percentage of partial features introduced by masking. We have considered N =

[7]https://www.openml.org/d/1169
[8]https://archive.ics.uci.edu/ml/datasets/Adult
[9]https://www.kaggle.com/c/rossmann-store-sales/data

{10,20,50} and percentage of partial feature as 20%, 40% and 60% for conducting experiments.

4.4 Experimental Setup

In this thesis, we have conducted a series of experiments to compare different techniques to transform the categorical features into numerical vectors. We have also conducted experiments implementing methods where a learning model learns from partial data. In this section, an overview of implementation details is provided. We implemented four encoding techniques to encode input data(categorical features) into a feature vector. Various python tools and libraries used to create models and evaluate implemented models are explained in this chapter. Further, this chapter also includes the environmental setup and the approach used in this thesis work.

4.4.1 Tools and Framework

Machine learning has gained immense popularity in the modern world. Many open-source tools, libraries, and frameworks are available for developers and researchers to implement their machine learning solutions. Python is considered one of the most popular programming languages for machine learning because of its vast collection of libraries and frameworks. In this thesis work, we have designed and performed various experiments using such tools and libraries.

Scikit-learn is one of the most popular machine learning libraries. It is built on top of two other python libraries, i.e., NumPy and SciPy. Scikit-learn or sklearn contains various algorithms for different machine learning problems, such as classification, regression, preprocessing, model selection, etc. In our experiments, we have used various classification and regression algorithms along with preprocessing algorithms.

PyTorch is an open-source deep learning framework. PyTorch is used for two high-level operations:

- It introduced tensor, which is similar to NumPy array but with better GPU performance.

- It provides a flexible framework to built complex deep learning models.

We have used PyTorch for both the above operations in our experiments.

Keras is also an open-source API used to create deep learning models. The model and layer API of Keras is used to create flexible neural networks. Keras provides its users already defined and constructed model and layer API and gives its users flexibility to create more complex neural network from scratch. We have used Keras to implement Entity Embedding for our research.

4.4.2 Approach

As defined in section 1.1, the main goal of this thesis is to evaluate different methods to encode categorical data and feature-set-based instance in a machine learning problem. Section 4.1 and 4.2 described the implementation details of different encoding methods for categorical data (label encoding, one-hot encoding and entity embedding) and feature set embedding.

Learning from encoded categorical features

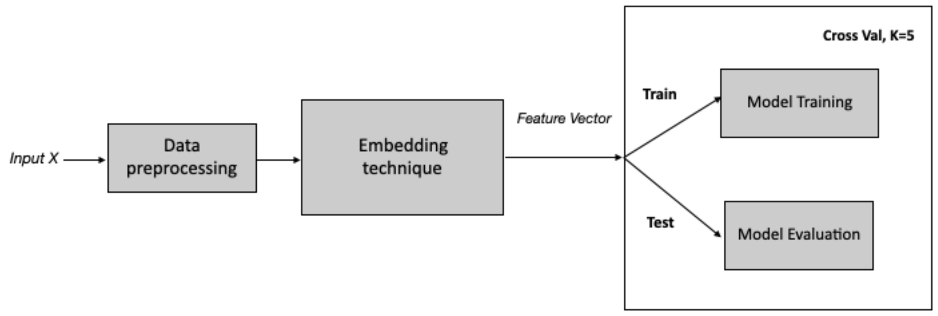
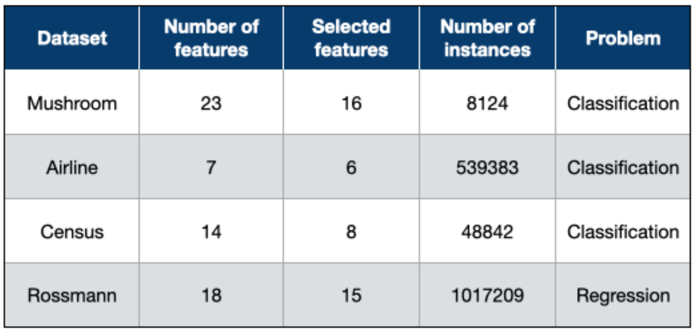


Fig. 4.1.: The figure illustrates the general approach followed for the first experiment for encoding categorical data, denoted as Experiment A1.

Figure 4.1 illustrates different steps performed in the first set of experiments, Experiment A1, to evaluate the implemented encoding methods for categorical features. As the first step of the implementation, feature selection is performed where relevant features are selected for each dataset. In a machine learning problem, feature selection is a crucial step to ensure that the learner is fed with relevant data and reduce the computational complexity. However, since the datasets are pretty simple, feature selection was done manually using the following criteria:

- Columns that contain redundant information are filtered.
- Columns with the same values for each instance.
- Columns with more than 67% missing values.



Dataset	Number of features	Selected features	Number of instances	Problem
Mushroom	23	16	8124	Classification
Airline	7	6	539383	Classification
Census	14	8	48842	Classification
Rossmann	18	15	1017209	Regression

Tab. 4.1.: The table illustrates the number of features present in the original dataset and the number of features after pre-processing.

As represented in the figure 4.1, after pre-processing of the datasets, selected encoding techniques: label encoding, one-hot encoding, and entity embedding are applied. For one-hot encoding, pandas' get_dummies method is used. The selected implementation of one-hot encoding makes it easier to accommodate the encoding of multiple columns with one line of code. Using one-hot encoding drastically impacts the size of the feature vector. For a categorical feature f, m new columns are added, where m is the number of unique categories for the feature f.

On the other hand, sklearn's LabelEncoder transformer is used for implementing label encoding for categorical features. Using label encoding does not change the size of the feature vector. In this case, for each unique value of the categorical feature, a numeric label $\{0, 1, .., N\}$ is assigned.

The implementation of entity embedding model used for experiments is detailed in section 4.1.3. An embedding layer is added to the neural network for each categorical feature, and a dense layer is used for the binary and continuous features. A concatenation layer is added on top of this to merge all the inputs. The input data $\{X, y\}$ is divided into train and test datasets for the cross-validation within the entity embedding model. During the learning process, the neural network optimizes the weights associated with each embedding layer. Finally, the fitted model is used to obtain the weights for each embedding layer. The categorical features are then replaced in the input data with the learned embeddings.

In the next stage of this experiment, we tested out how the above implemented encoding technique affect the performance of different machine learning model. We have used logistic regressor, decision tree classifier, k-nearest neighbor classifier, and gradient boosting classifier for the classification task. However, we have used decision tree regressor, k-nearest neighbors regressor, and random forest regressor for the regression task.

The next step in experiment A1 is to evaluate the performance of different models with the above-mentioned encoding technique. The experiments are designed with k-fold cross-validation to evaluate different models. For all the experiments, we used k=5 for cross-validation. In the case of classification tasks, StratifiedKFold cross-validation is used to overcome the problem of imbalanced class distributions. However, for the regression task, k-fold cross-validation is used.

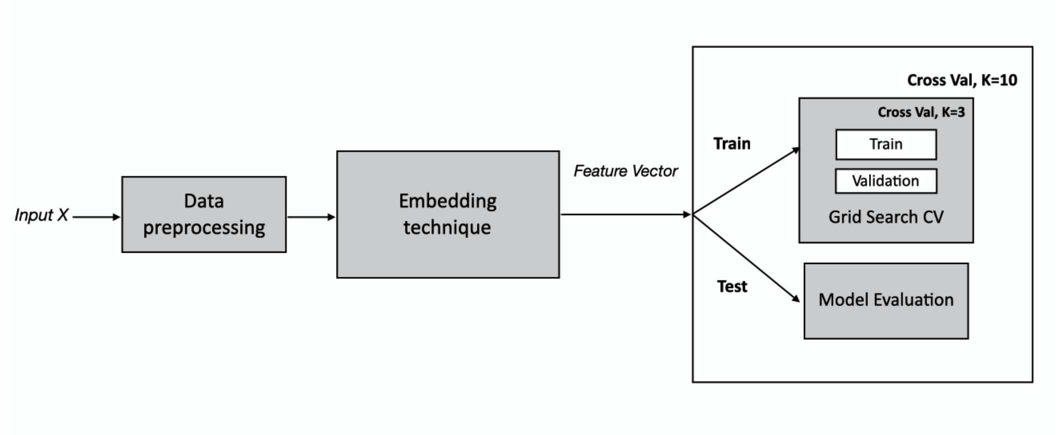


Fig. 4.2.: The figure illustrates the general approach followed for the final experiment for encoding categorical data, denoted as Experiment A2.

After getting results from the first set of experiments A1, another set of experiments, denoted as Experiment A2, are designed. In experiment A2, the first two steps remain the same, i.e., data preprocessing and implementation of encoding techniques. As the last step, hyperparameter optimization is implemented to tune the hyperparameters of the learning algorithm. In a machine learning problem, hyperparameter tuning helps in finding the best hyperparameter for a learner and make the results comparable. In our case, we have used nested-cross validation for hyperparameter optimization. The outer loop of cross-validation takes the complete set as an input and creates k=10 folds for cross-validation. In this loop, the dataset is divided into train and test data. The inner loop takes only the train data as an input and creates k=3 folds for cross-validation. In the inner loop, a grid search is performed to get the best hyperparameters for the model. This implementation ensures that the test data used for prediction is never seen by the model during training and hyperparameter optimization.

Learning from feature-set-based instances

Figure 4.3 shows the step-by-step implementation of Experiment B setup to evaluate feature set embedding. The first step here is to generate the synthetic dataset. The implementation details of data generator are previously discussed in section 4.3. The data generator generates the train and test data based on the number of features, N, and the number of instances for training and testing, Itrain and Itest. For the experiments, all the datasets are generated with Itrain = 10000 and Itest = 1000. However, three different value of N, {10, 20, 50} is used to generate datasets with different number of features. A masking method is designed to introduce partial features in the dataset.

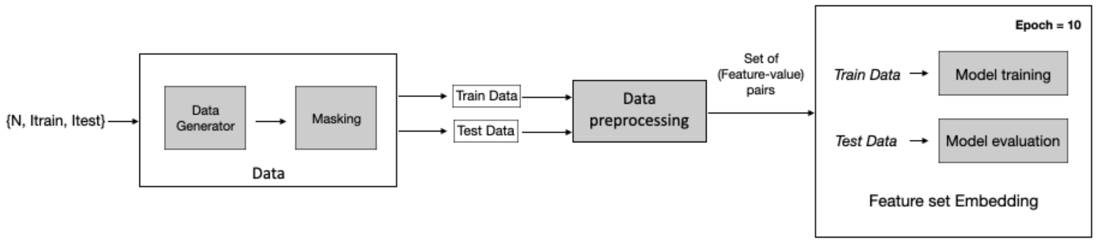


Fig. 4.3.: The figure illustrates the general approach followed for the final experiment for feature set embedding, denoted as Experiment B.

The next step in the implementation is to transform the data vectors into a set of feature-value pairs for each instance. Once the dataset is ready, where each instance is represented as a (feature, value) pair, it can be used to train and evaluate the designed feature set embedding model.

The implementation of feature set embedding model is explained in detail in section 4.2.1. Feature set embedding is implemented as a neural network with four layers. The first layer is the embedding layer that generates embedding for (feature, value) pair for each instance. On top of the embedding layer, a pooling layer is added that combines the generated embedding vectors. The output of the two layers together gives a feature vector for each instance that the classifier can now use to predict the correct class for each instance. The designed neural network is trained using backward propagation. Since the generated dataset introduces the problem of binary classification, cross-entropy is used as a loss function. A stochastic gradient descent optimizer is used for optimizing the models' parameters. In the experiments, the feature set embedding model is trained and evaluated for ten epochs, and the resultant accuracy on test data is presented.

5 Evaluation

This chapter describes an evaluation of previously presented approaches on different datasets. The overall goal of the evaluation is to emphasize the difference in the performance of various machine learning models with the change in encoding techniques. In this thesis, different methods are used to encode categorical data into feature vectors. The main goal of the thesis is to evaluate different encoding techniques to encode business models while addressing the categorical and partial data. But, as mentioned in the previous chapter, the original business model data is not yet available to perform experiments and evaluate the selected techniques. Therefore, we have evaluated different techniques using open-source datasets. In the following section, we define different evaluation metrics used to compare the performance of implemented machine learning models. Later, the results from various experiments are discussed.

5.1 Evaluation Metrics

In a machine learning process, after feature engineering, data preprocessing, and building the model, the next step in the pipeline is to evaluate the model's performance using some performance metrics. There are different performance metrics for different types of machine learning problems. In our work, we have implemented both classification and regression models. Therefore, there is a need to identify various evaluation metrics used in classification and regression tasks.

A model can either predict a correct or an incorrect class for a classification task for test data. Once the prediction is made on the test data X_test, the results y_pred are compared to the original labels y_test. Accuracy is the most common evaluation metrics used for classification problems. It is defined as the number of correct predictions divided by total predictions made for the test dataset. Accuracy is a useful measure for a balanced dataset. A balanced dataset is where the distribution of classes is even. However, for unbalanced datasets, other measures can be considered—for example, recall and precision. Evaluation of the performance of a classification model is based on identifying the correct and the incorrect predictions. A Confusion matrix provides a summary of the model's performance on test data. It is advantageous in identifying the type of errors made by the model during prediction. The confusion matrix determines the four major factors for prediction, which are used to calculate the accuracy, precision, recall, and f1 score. Figure 5.1 illustrates a confusion matrix for a classification problem that classifies an input as positive or negative.

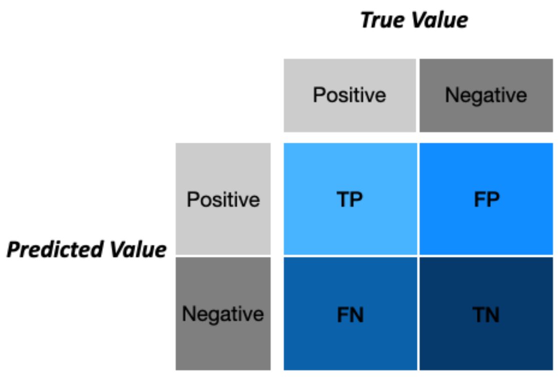


Fig. 5.1.: Confusion matrix for classification

True Positive, TP where the true class is positive and the predicted class is positive

True Negative, TN where the true class is negative and the predicted class is negative

False Positive, FP where the true class is negative and the predicted class is positive

False Negative, FN where the true class is positive and the predicted class is negative

Now, as mentioned above, accuracy is the measure of total correct prediction over the total prediction. Therefore, in terms of the above factors, accuracy can be calculated using the below formula.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

Recall is calculated as the ratio of true positives to all the positives in the test dataset.

$$\text{Recall} = \frac{TP}{TP + FN}$$

However, precision of a model is the ratio of true positives to all the positives predicted by the model.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Choosing an optimal performance matrix depends on the dataset and the domain of the machine learning task. For example, for a model that predicts whether a bank loan should be approved for a client, the impact of having more false-negative could result in losses for the bank. According to the model's prediction, the loan could be approved for a faulty client, resulting in a false negative and a lower recall. Therefore, in this case, there is a need to maximize the recall.

A balance between recall and precision can be obtained using the F1 score. The F1 score is a combination of recall and precision. The maximum value for F1 score is 1 while the minimum is 0.

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

On the other hand, different metrics are used for evaluating the performance of a regression model. The most important metrics are mean absolute error(MAE), mean square error(MSE) or root mean square error(RMSE), R^2 or adjusted R^2. In this thesis, we have implemented a regression model that trains on different types of feature vectors. Here we want to evaluate how different encoding techniques impact the models' performance. Error metrics like MSE, MAE, or RMSE are very useful when optimizing the model by looking at the error. R^2 [CWJ21] is a statistical measure that signifies the proportion of variance in a dependent variable that can be explained by the independent variable of a regression model. For a linear regression model, R^2 value ranges from 0 to 1. However, for the non-linear regression models, the R^2 value could be anywhere between -∞ to 1.

Chicco et al. [CWJ21] defined R^2 using two main terms, mean square error (MSE) and mean total sum of squares (MST). In the below formulas, $\hat{Y}_i$ is the predicted value and Y_i actual value for the i^{th} element and m is the number of instances in test data. The mean of the true values can be defined as

$$\bar{Y} = \frac{1}{m}\sum_{i=1}^{m} Y_i \tag{5.1}$$

Mean total sum of squares

$$MST = \frac{1}{m}\sum_{i=1}^{m} (Y_i - \bar{Y})^2 \tag{5.2}$$

Mean square error

$$MSE = \frac{1}{m}\sum_{i=1}^{m} (\hat{Y}_i - Y_i)^2 \tag{5.3}$$

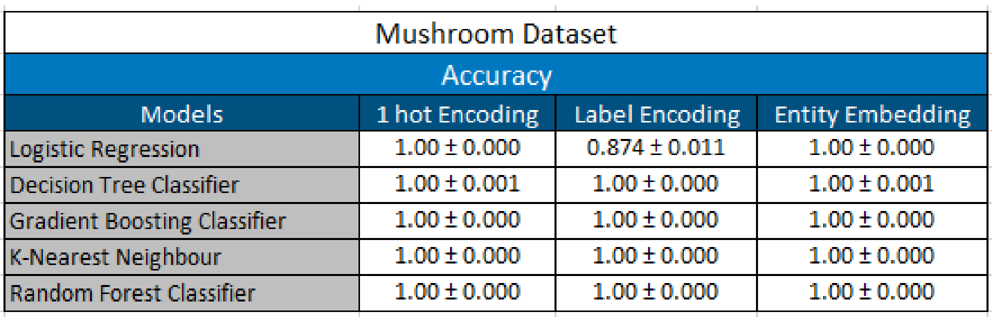
Hence, R^2 is defined as follows:

$$R^2 = 1 - \frac{MSE}{MST} \tag{5.4}$$

or

$$R^2 = 1 - \frac{\sum_{i=1}^{m} (\hat{Y}_i - Y_i)^2}{\sum_{i=1}^{m} (Y_i - \bar{Y})^2} \tag{5.5}$$

Apart from the model's performance, time is another factor that impacts the evaluation of different machine learning models. It turns out that there is a difference in training time for models enabled with different encoding techniques. The computation time required to train a model highly impacts the cost associated with the problem. The longer it takes to train a model, the higher is the computational cost. In this thesis, we evaluate different models on various levels to draw the final results.



Mushroom Dataset			
Accuracy			
Models	1 hot Encoding	Label Encoding	Entity Embedding
Logistic Regression	1.00 ± 0.000	0.874 ± 0.011	1.00 ± 0.000
Decision Tree Classifier	1.00 ± 0.001	1.00 ± 0.000	1.00 ± 0.001
Gradient Boosting Classifier	1.00 ± 0.000	1.00 ± 0.000	1.00 ± 0.000
K-Nearest Neighbour	1.00 ± 0.000	1.00 ± 0.000	1.00 ± 0.000
Random Forest Classifier	1.00 ± 0.000	1.00 ± 0.000	1.00 ± 0.000

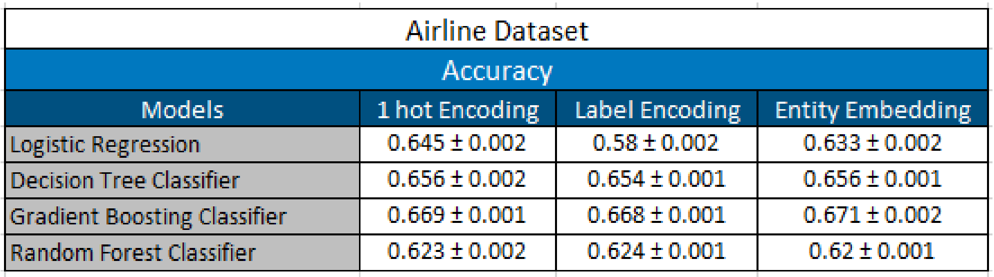
Tab. 5.1.: Comparison between label encoding, one-hot encoding and Entity Embedding for Mushroom Dataset. The table shows the mean accuracy with standard deviation obtained for five different classifiers feed by different encoding.

5.2 Results

The overall goal of this thesis is to evaluate different methods to encode categorical data and feature-set-based instances. To achieve this goal, multiple experiments are conducted as described in 4. In the following section, the experimental results are discussed in detail. In the first section, the results from the experiments regarding the encoding of categorical features are listed. However, the second section outlines the results for the feature set embedding experiments.

5.2.1 Encoding of categorical data

One of the main goals of this thesis is to explore and evaluate different methods to encode categorical data in a machine learning problem. To accomplish this goal, three different encoding techniques are implemented. As described in section 4.4.2, different experiments are conducted to evaluate the implemented encoding methods. Four different datasets (details in 4.3) are used to test the performance of selected encoding methods, i.e., label encoding, one-hot encoding, and entity embedding. This section summarizes the results of all the implemented models for different datasets. Out of the four selected datasets, three are classification datasets and one regression dataset. The encoded vectors are used as an input for various machine learning models to evaluate how the implemented encoding techniques impact the model's performance. For the classification task, the model's performance based on accuracy and time is discussed. However, for the regression task, R^2 score and time are discussed.



Airline Dataset			
Accuracy			
Models	1 hot Encoding	Label Encoding	Entity Embedding
Logistic Regression	0.645 ± 0.002	0.58 ± 0.002	0.633 ± 0.002
Decision Tree Classifier	0.656 ± 0.002	0.654 ± 0.001	0.656 ± 0.001
Gradient Boosting Classifier	0.669 ± 0.001	0.668 ± 0.001	0.671 ± 0.002
Random Forest Classifier	0.623 ± 0.002	0.624 ± 0.001	0.62 ± 0.001

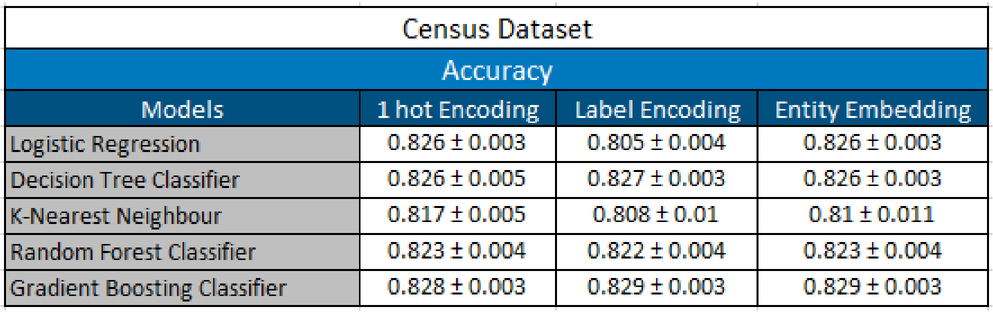
Tab. 5.2.: Comparison between label encoding, one-hot encoding and Entity Embedding for Airline Dataset. The table shows the mean accuracy with standard deviation obtained for four different classifiers feed by different encoding.

The mushroom dataset was the smallest dataset used for experiments. Table 5.1 shows that, except for Logistic regression, no significant difference could be observed for the three different encoding techniques. In the case of logistic regression, the use of feature vectors encoded by label encoding produces the least accuracy of 87.4%. For the other classification models like decision tree, k-nearest neighbor, random forest, and gradient boosting classifier, all the models show 100% accuracy for three different techniques.

Apart from the mushroom dataset, experiments were conducted on two other classification datasets: the Airline and the Census datasets. In the case of the airline dataset, label encoding gives the best accuracy of 62.4% for a random forest classifier. However, in the case of gradient boosting classifier, entity embedding work best by delivering an accuracy of 67.1%. The results in table 5.2 show that one-hot encoding surpasses the other two techniques in terms of accuracy for logistic regressor. At last, for the decision tree classifier, both one-hot encoding and entity embedding resulted in the same accuracy of 65.6%.

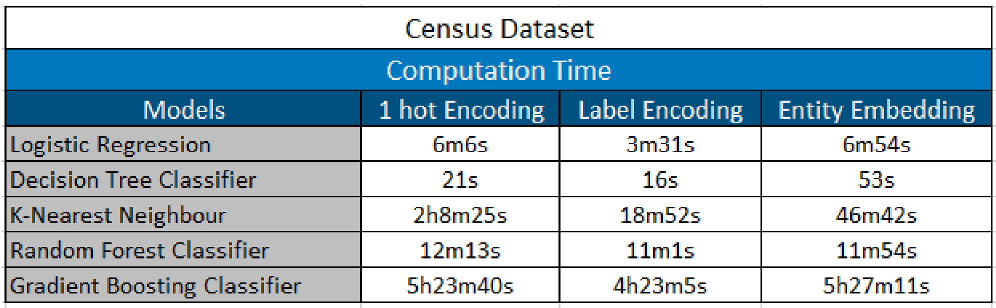
Table 5.3 shows the results obtained for the Census dataset. Here, three different encoded vectors are fed to five different classifiers: logistic regressor, decision tree, k-nearest neighbor, random forest, and gradient boosting classifier. In the case of the decision tree, random forest, and gradient boosting classifier, no significant difference is observed in terms of accuracy for different encoding techniques. Like mushroom and airline datasets, label encoding gives the least accuracy for logistic regressor, even for the census dataset. One-hot encoding and entity embedding gave the same accuracy of 82.6% for logistic regressor.

Next, table 5.4 depicts the CPU time for different settings using the Census dataset. For all five models, label encoding took the least time because the size of the feature



Census Dataset			
Accuracy			
Models	1 hot Encoding	Label Encoding	Entity Embedding
Logistic Regression	0.826 ± 0.003	0.805 ± 0.004	0.826 ± 0.003
Decision Tree Classifier	0.826 ± 0.005	0.827 ± 0.003	0.826 ± 0.003
K-Nearest Neighbour	0.817 ± 0.005	0.808 ± 0.01	0.81 ± 0.011
Random Forest Classifier	0.823 ± 0.004	0.822 ± 0.004	0.823 ± 0.004
Gradient Boosting Classifier	0.828 ± 0.003	0.829 ± 0.003	0.829 ± 0.003

Tab. 5.3.: Comparison between label encoding, one-hot encoding and Entity Embedding for Census Dataset. The table shows the mean accuracy with standard deviation obtained for five different classifiers feed by different encoding.

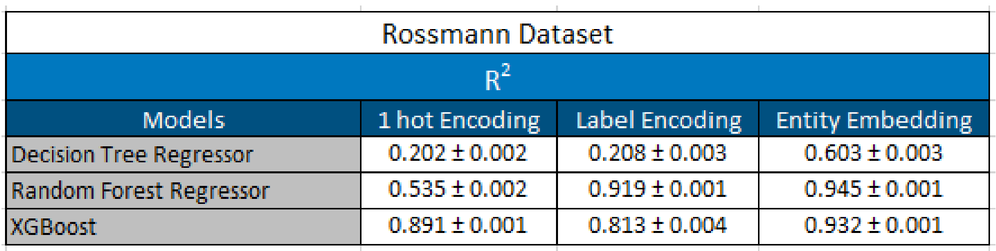


Census Dataset			
Computation Time			
Models	1 hot Encoding	Label Encoding	Entity Embedding
Logistic Regression	6m6s	3m31s	6m54s
Decision Tree Classifier	21s	16s	53s
K-Nearest Neighbour	2h8m25s	18m52s	46m42s
Random Forest Classifier	12m13s	11m1s	11m54s
Gradient Boosting Classifier	5h23m40s	4h23m5s	5h27m11s

Tab. 5.4.: Comparison between label encoding, one-hot encoding and Entity Embedding for Census Dataset. The table shows the computation time for five different classifiers feed by different encoding.

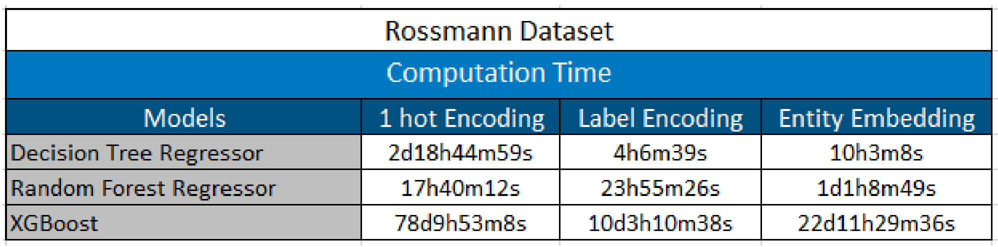
vector is the same as the number of features in the input set. However, for one-hot encoding, the size of the feature vector depends on the number of categorical values for each feature. Except for random forest classifiers, using entity embedding took the most time for processing.

As mentioned before, the Rossmann dataset is used for the regression task. Table 5.5 shows the R^2 score noted for three different regression models with the selected encoding techniques as an input. For the Rossmann dataset, entity embedding gives the best performance in terms of the R^2 score. For a decision tree regressor, R^2 for label encoding and one-hot encoding are 20.8% and 20.2% respectively, which suggests that the variance in the predicted dependent variable is not explained by the independent variables. For a random forest regressor, one-hot encoding resulted in the lowest R^2 score. For xgboost, the difference in R^2 is not much for label encoding and one-hot encoding. Although, the entity embedding surpasses the two other encoding techniques for xgboost with the maximum R^2 score of 93.2%.



Rossmann Dataset			
R^2			
Models	1 hot Encoding	Label Encoding	Entity Embedding
Decision Tree Regressor	0.202 ± 0.002	0.208 ± 0.003	0.603 ± 0.003
Random Forest Regressor	0.535 ± 0.002	0.919 ± 0.001	0.945 ± 0.001
XGBoost	0.891 ± 0.001	0.813 ± 0.004	0.932 ± 0.001

Tab. 5.5.: Comparison between label encoding, one-hot encoding and Entity Embedding for Rossmann Dataset. The table shows the mean R^2 with standard deviation obtained for three different regressor feed by different encoding.



Rossmann Dataset			
Computation Time			
Models	1 hot Encoding	Label Encoding	Entity Embedding
Decision Tree Regressor	2d18h44m59s	4h6m39s	10h3m8s
Random Forest Regressor	17h40m12s	23h55m26s	1d1h8m49s
XGBoost	78d9h53m8s	10d3h10m38s	22d11h29m36s

Tab. 5.6.: Comparison between label encoding, one-hot encoding and Entity Embedding for Rossmann Dataset. The table shows the computation time for three different regressor feed by different encoding.

The result for computation time for the Rossmann dataset is listed in table 5.6. The results show that all the models enabled with label encoding took the least time for training. For random forest regressor, the entity embedding took more than 25 hours to complete the training process. However, for the one-hot encoding, the training process is completed in less than 18 hours. For the other two regressors, decision tree and xgboost, using feature vectors encoded by one-hot encoding resulted in exceptionally high CPU time. For example, table 5.6 shows that the computational time for xgboost using a one-hot feature vector is noted to be more than 78 days.

5.2.2 Feature set embedding

The experiments were conducted to evaluate the performance of implemented feature set embedding while noting the impact of the quantity of partial data and the number of features in the dataset. In this thesis, the feature set embedding is implemented to overcome the problem faced while training machine learning with partial data. The experiments are performed in three different setups, partial data only at training time, partial data only at testing time, partial data in both training and testing. In different setups, the accuracy of the classification model is noted for different values of N, representing the total number of features for each instance. As mentioned before, partial data implies that the corresponding values are missing for one or more features. The following section lays down the results in different experimental setups, showing how the number of features and the amount of partial data impacts the model's performance.

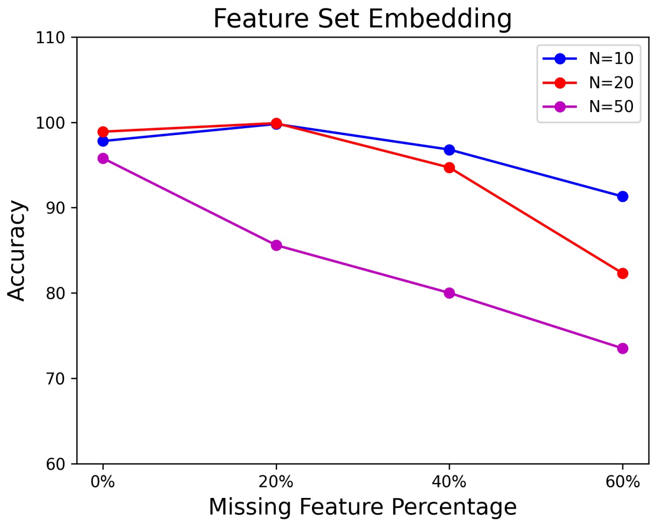


Fig. 5.2.: Accuracy of implemented feature set embedding in different setting. The visualization depicts the model accuracy for three different values of N, the number of features for each instance and corresponds to partial data at training time only.

Partial features at train time

Figure 5.2 depicts the accuracy of the feature set embedding model such that training data is partial. The results show that for $N = 20$, the model performed with the highest accuracy for complete data, where no features were missing. While dealing with partial train data, the model's accuracy declines as the percentage of the missing features increases in the input data. For example, for $N = 10$, as the percentage of missing features in partial data increases, the model accuracy simultaneously decreases.

Partial features at test time

Figure 5.3 shows the evaluation of the feature set embedding model in the experimental setup where the training data has no partial features. However, while evaluating the model, test data consist of partial features. In this case, the model's performance based on accuracy is observed for different values of N and the percentage of missing features in the test data. The results show that the model's

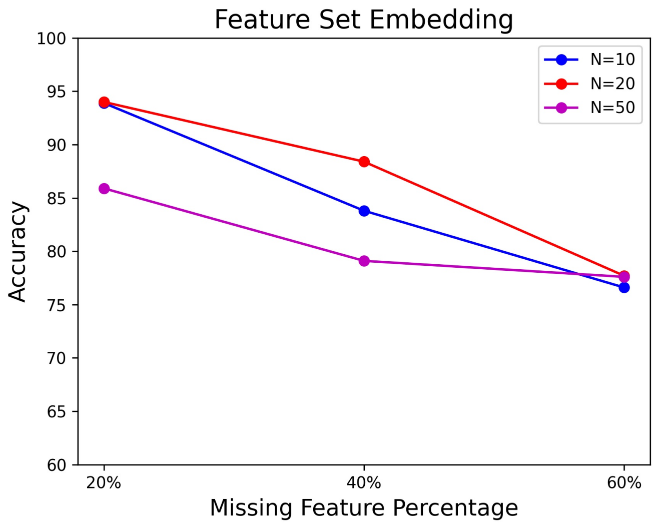


Fig. 5.3.: Accuracy of implemented feature set embedding in different setting. The visualization depicts the model accuracy for three different values of N, the number of features for each instance. These experimental result corresponds to partial data at testing time only.

accuracy is 94% for $N = 20$ for 20% missing features in the test data. Yet again, as the percentage of missing features in the partial dataset increases, the model's performance declines drastically. Similar observations can be made for $N = 10$ and $N = 50$.

Partial features at train and test time

Figure 5.4 visualizes the result for the feature set embedding in the third experimental setup, where partial features are observed at both train and test time. A similar scenario is expected to be for the business model data. There is a possibility that the partial data would be seen in the train and the test dataset. However, the same pattern is observed for this setup as well. The model's accuracy on test data decreases as the percentage of the partial features increases in both train and test data.

In all the three setups, the model's performance is impacted by the number of features in the train and test data, but there is no significant difference observed between $N = 10$ and $N = 20$. However, increasing the number of features by 5 times, such that $N = 50$, the accuracy of the model declines by 5-8% in every case.

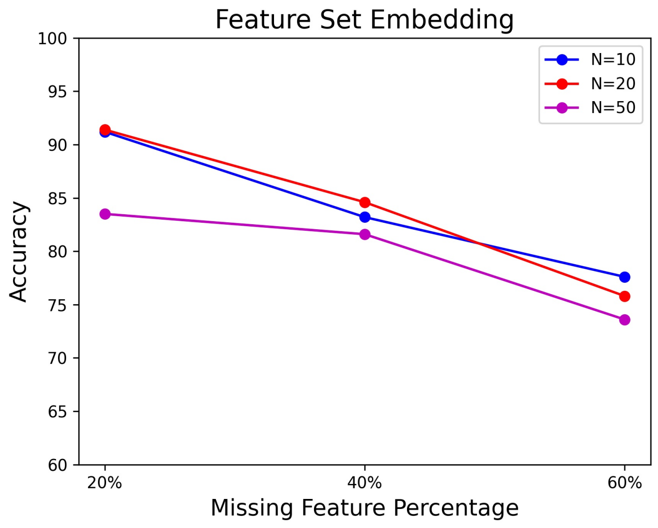


Fig. 5.4.: Accuracy of implemented feature set embedding in different setting. The visualization depicts the model accuracy for three different values of N, the number of features for each instance. These experimental result corresponds to partial data at both training and testing time.

Next, the figure 5.5 visualizes the performance accuracy of feature set embedding, highlighting the difference in model's accuracy in the previously mentioned three different setups for $N = 10$. Irrespective of the percentage of the missing data in the partial dataset, the maximum accuracy is seen with the first setup, i.e., partial features at train time. For example, when the feature set embedding model is fed with 20% missing features in train data, the model predicts test data with an accuracy of 99.8%. However, in the second setup, where partial features are seen in both training and test datasets, the model's accuracy reduces to 91.20%. In the final setup, where the model is trained on the complete dataset, but the prediction is made on the partial dataset, the model's accuracy is noted to be 93.90%. The graph in the figure 5.5 shows the same trend when the percentage of partial features in the dataset is 40% and 60%.

To summarize, this thesis aims to evaluate different techniques to encode categorical features and handle feature-set-based instances. Three different encoding methods, label encoding, one-hot encoding, and entity embedding, are implemented to encode categorical features. The evaluation of these encoding techniques shows that using label encoding to encode categorical data for most datasets results in a worse performance of the machine learning models. However, one-hot encoding and entity embedding performed better than label encoding. For the classification task, although the performance of one-hot encoded vectors and entity embedding

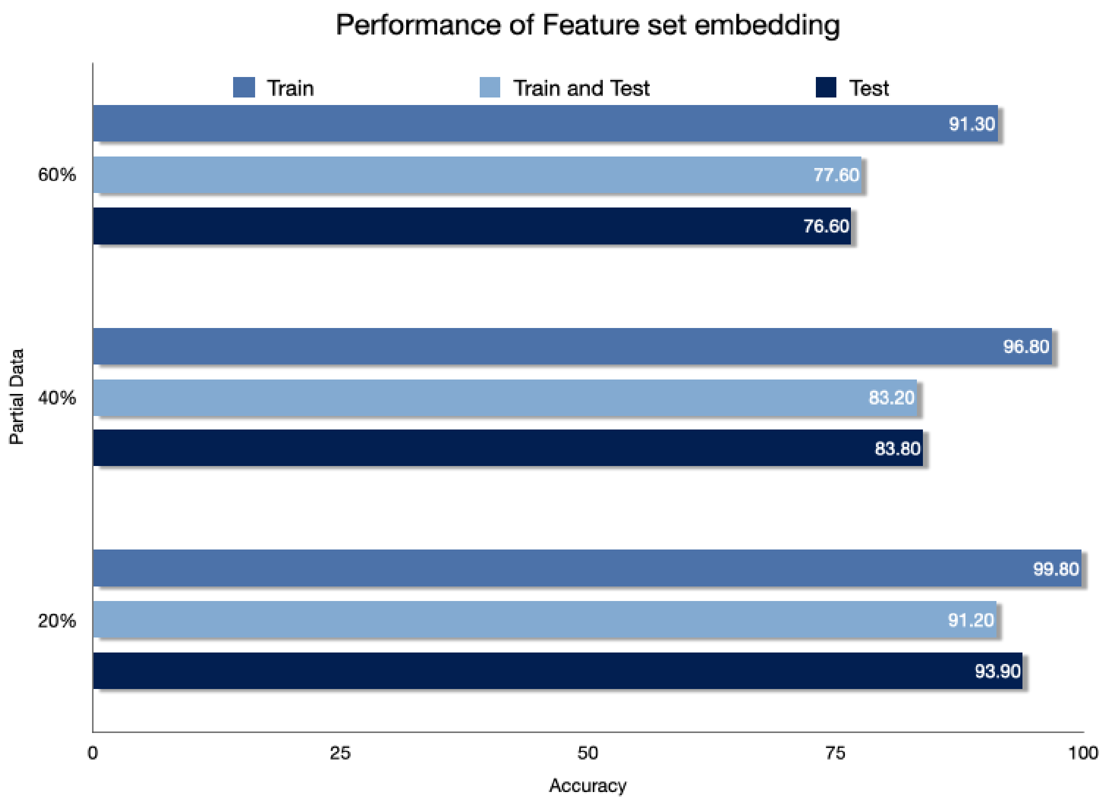


Fig. 5.5.: Performance of feature set embedding for three different setup for $N = 10$, partial features at train time, partial features at train and test time and partial features at test time.

was almost similar in terms of model accuracy, it is important to note that the computational time for one-hot encoding was longer compared to entity embedding. Computation time and resources are important aspects for deciding the suitable encoding method. Therefore, considering the performance of different machine learning models and the computation time for implemented encoded vectors, entity embedding looks like a promising approach that can be further implemented to handle categorical features in the business model datasets.

Also, feature set embedding is implemented and evaluated to address the second goal of the thesis, i.e., to handle the feature-set-based instances. Synthetic datasets are generated programmatically for assessing the performance of feature set embedding. The experiments are designed to evaluate how feature set embedding handles partial data. The experiments are performed in three different settings, partial features at train time, partial feature only at test time, and partial feature at train and test time. The overall results show that as the percentage of partial data increases, the model's performance declines.

6 Conclusion

This thesis identifies the challenges faced while transforming a business model data into a feature vector. Two central problems when dealing with a business model in a machine learning problem are categorical data and partial data. This thesis explored different techniques to deal with both categorical and partial data in a machine learning problem. For categorical data, three different approaches from the literature are implemented, label encoding, one-hot encoding, and entity embedding. The approaches like label encoding and one-hot encoding have relatively more straightforward implementation. However, entity embedding was implemented as a neural network that learns embeddings for all the categorical features using Keras. The feature set embedding was implemented to address the problem of partial data in machine learning problems using PyTorch modules.

In order to test how the encoded vector generated by different encoding methods impact the learning process, different machine learning models were designed, implemented, and evaluated depending on the type of problem, i.e., classification or regression.

For the problem of categorical data, various experiments are performed on four different datasets selected based on fixed criteria. This work demonstrated and compared the accuracy of machine learning models applied to categorical data encoded using different encoding techniques. For the classification dataset, the model's accuracy is compared to carry out the evaluation. For the mushroom dataset, no significant difference in accuracy was observed for five out of four classifiers. Interestingly, the mushroom dataset is the smallest dataset used for experiments. For the airline dataset, using label encoding to encode the categorical variable resulted in the worst performance, and One-hot encoding outperformed the other two techniques 75% of the time. Even for the Census dataset, similar observations were made. Entity embedding resulted in better model accuracy than label encoding 80% of times, but one-hot encoding performed equally or better than entity embedding for four out of five classifiers. It is important to note that, even though there is a difference in the model's accuracy for different encoding techniques, the accuracy is always between 80-83%.

For the regression task, the categorical features in the Rossmann dataset were encoded with label encoding, one-hot encoding, and entity embedding. In this case, the entity embedding outperformed the other two encoding techniques for three different regressors.

Even though there is not much difference in the model's performance for one-hot encoded vectors and feature vectors encoded by entity embedding, the computational time taken for different encoding techniques varies a lot. Except for the decision tree classifier, the total CPU time for one-hot encoding is the highest. Entity embedding took 53 seconds for the decision tree classifier because of the training involved in learning the embedding using neural networks. The main reason for higher computational time for training on a one-hot encoded vector is the size of the feature vector. For each categorical feature, n number of new columns are added to the dataset, where n is the number of unique values for the categorical feature. When the value of n is significantly large, the size of the feature vector drastically increases.

Considering the model accuracy and the computational time for each encoding technique, entity embedding is a better choice when compared to the other two techniques.

For the problem of partial data, feature set embedding was implemented and evaluated under different experimental setups. The feature set embedding considers the inputs as a set of feature value pairs instead of vectors, making handling missing features easier. For evaluating the implementation of feature set embedding, a synthetic dataset is created, and some features are removed at random. Initially, the model was tested on a complete dataset, and later missing features were introduced. In different experimental settings, the model performance decrease with the increase in partial data.

In conclusion, while some interesting results could be shown in this experiment, it is still necessary to conduct further evaluations for more complex datasets with real-world business model data.

6.1 Limitation

In this thesis, we implemented different methods to deal with categorical data and feature-set-based instances in a machine learning problem. The following section lists the limitation of the conducted experiments and the approach used in this thesis.

Two embedding techniques are implemented in this thesis: entity embedding and feature set embedding. The embedding dimension d, is an important hyperparameter for an embedding layer representing the dimension of the embedding space on which categorical features are encoded. Our experiments did not use any hyperparameter optimization to get the best possible value for embedding dimension. In the case of entity embedding, the value of d is selected at random for embedding layers. However, for regularization, d was limited to be less than $n/2$, where n is the number of unique values for a categorical feature. For feature set embedding, the embedding dimension is bounded between 1 to $m-1$, where m is the maximum number of features for an instance.

Secondly, due to time constraints, the unsupervised embeddings are not implemented and evaluated as a part of this thesis. In section 3.1, many interesting approaches were highlighted from previous work. For example, the GloVe model performed well while depicting the relation and the distance between the categorical variable in the embedded space [MM20]. Once the business model dataset is available, the GLoVe model can also be considered for encoding categorical features in business models.

6.2 Future Work

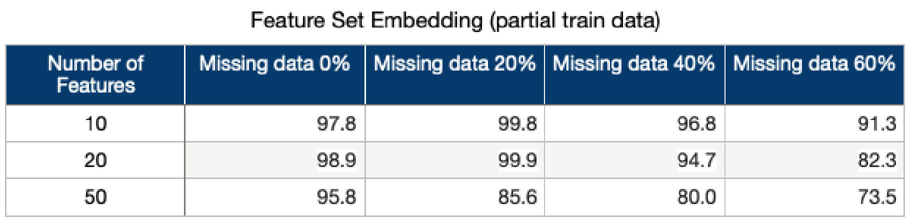
While this thesis describes and evaluates different approaches for handling the challenges faced when encoding business models using feature vectors, there are still some unanswered questions. The future work can be divided into the dataset, hyperparameter for the implemented models, and other approaches for encoding categorical data.

First, in this thesis, all the implemented approaches are evaluated using open-source datasets or the synthetically generated dataset. To understand how these techniques perform with real-world business model data, it is essential to conduct further experiments once the business model data is available.

Second, there is no fixed approach to select the dimension of the embedding space for entity embedding and feature set embedding. For feature set embedding, the embedding dimension is bounded between 1 to $m - 1$, where m is the maximum number of features for an instance. Defining a theoretical guideline to select the value of embedding dimension would be very useful. It will enable the researchers to follow the guideline to choose an optimal embedding dimension without worrying about the model's performance.

Lastly, in this thesis, we have compared and evaluated three different techniques for encoding categorical data. Label encoding and one-hot encoding are deterministic techniques. However, entity embedding is a deep learning technique implemented in a supervised setting. Other unsupervised embeddings for categorical data like BERT, GLoVe, word2vec were not explored in this work because of the time constraint. Once the business model data is available, such techniques could also be implemented to encode categorical data. The advantage of these models is that they are pre-trained models and can be implemented quickly.

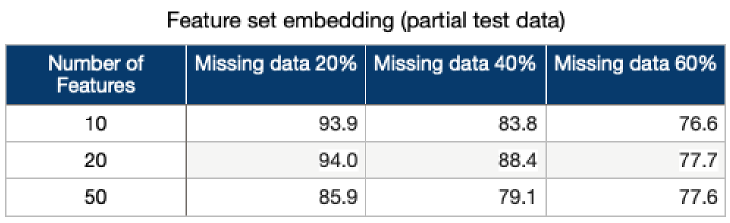
A Appendix



Feature Set Embedding (partial train data)

Number of Features	Missing data 0%	Missing data 20%	Missing data 40%	Missing data 60%
10	97.8	99.8	96.8	91.3
20	98.9	99.9	94.7	82.3
50	95.8	85.6	80.0	73.5

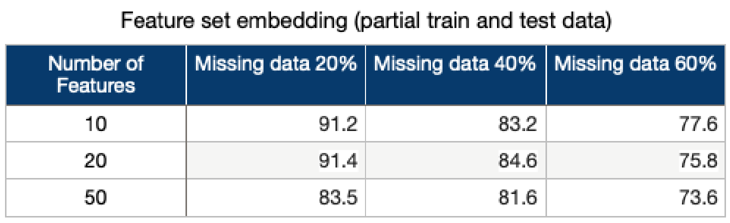
Tab. A.1.: Accuracy of implemented feature set embedding in different setting. The results are listed for three different values of N, the number of features for each instance. These experimental result corresponds to partial data at training time only.



Feature set embedding (partial test data)

Number of Features	Missing data 20%	Missing data 40%	Missing data 60%
10	93.9	83.8	76.6
20	94.0	88.4	77.7
50	85.9	79.1	77.6

Tab. A.2.: Accuracy of implemented feature set embedding in different setting. The results are listed for three different values of N, the number of features for each instance. These experimental result corresponds to partial data at testing time only.



Feature set embedding (partial train and test data)

Number of Features	Missing data 20%	Missing data 40%	Missing data 60%
10	91.2	83.2	77.6
20	91.4	84.6	75.8
50	83.5	81.6	73.6

Tab. A.3.: Accuracy of implemented feature set embedding in different setting. The results are listed for three different values of N, the number of features for each instance. These experimental result corresponds to partial data at both training and testing time.

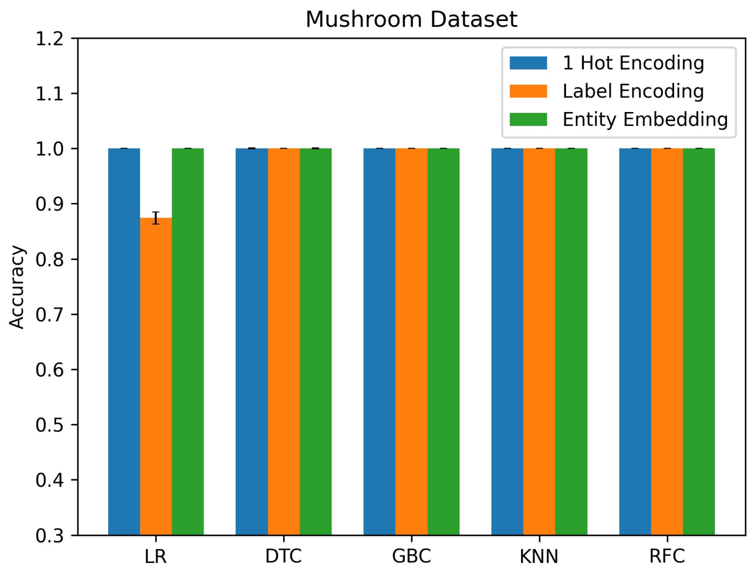


Fig. A.1.: The figure illustrates the performance accuracy of five different classification model trained on three different encoded vectors generated by label encoding, one-hot encoding and entity embedding for Mushroom dataset. The figure visualizes the mean accuracy of the models, along with the error bars representing the standard deviation from the mean value.

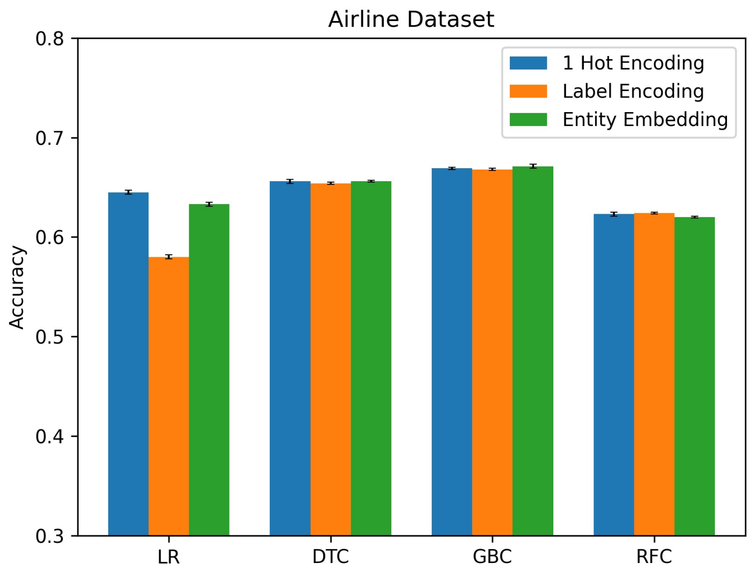


Fig. A.2.: The figure illustrates the performance accuracy of four different classification model trained on three different encoded vectors generated by label encoding, one-hot encoding and entity embedding for Airline dataset. The figure visualizes the mean accuracy of the models, along with the error bars representing the standard deviation from the mean value.

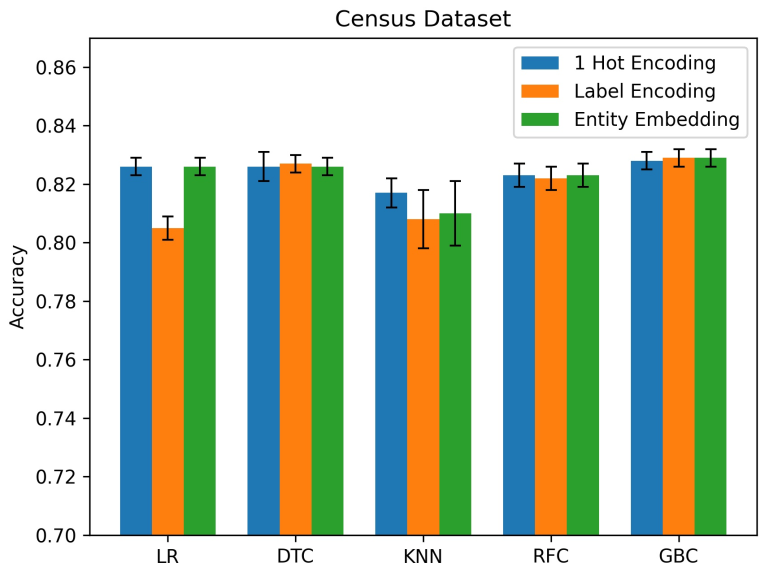


Fig. A.3.: The figure illustrates the performance accuracy of five different classification model trained on three different encoded vectors generated by label encoding, one-hot encoding and entity embedding for Census dataset. The figure visualizes the mean accuracy of the models, along with the error bars representing the standard deviation from the mean value.

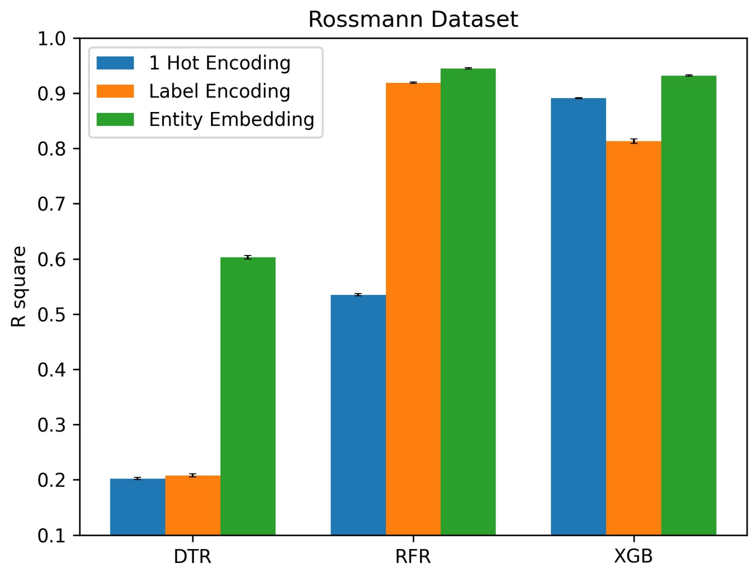


Fig. A.4.: The figure illustrates the R^2 value of three different regression model trained on three different encoded vectors generated by label encoding, one-hot encoding and entity embedding for Rossmann dataset. The figure visualizes the mean R^2 value of the models, along with the error bars representing the standard deviation from the mean value.

Bibliography

[AML12] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012 (cit. on p. 11).

[BM02] Gustavo E. A. P. A. Batista and Maria Carolina Monard. "A Study of K-Nearest Neighbour as an Imputation Method". In: *HIS*. 2002 (cit. on p. 25).

[Cai+21] Cesar F. Caiafa, Ziyao Wang, Jordi Solé-Casals, and Qibin Zhao. "Learning from Incomplete Features by Simultaneous Training of Neural Networks and Sparse Coding". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2021), pp. 2621–2630 (cit. on p. 32).

[CWJ21] Davide Chicco, Matthijs J. Warrens, and Giuseppe Jurman. "The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation". In: *PeerJ Computer Science* 7 (2021) (cit. on p. 41).

[Cho+15] François Chollet et al. *Keras*. `https://github.com/fchollet/keras`. 2015 (cit. on p. 29).

[Cyb89] George V. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2 (1989), pp. 303–314 (cit. on p. 17).

[Dan+20] Tomasz Danel, Marek Śmieja, Lukasz Struski, Przemysław Spurek, and Lukasz Maziarka. "Processing of incomplete images by (graph) convolutional neural networks". In: *ICONIP*. 2020 (cit. on p. 26).

[AA10] Mutaz Al-Debei and David Avison. "Developing a Unified Framework of the Business Model Concept". In: *European Journal of Information Systems* 19 (May 2010), pp. 359–376 (cit. on p. 8).

[DG17] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017 (cit. on p. 26).

[FH19] Matthias Feurer and Frank Hutter. "Hyperparameter Optimization". In: May 2019, pp. 3–33 (cit. on p. 15).

[GFc14] Oliver Gassmann, Karolin Frankenberger, and M.cSamil cSik. "The Business Model Navigator: 55 Models That Will Revolutionise Your Business". In: 2014 (cit. on p. 7).

[GJ93] Zoubin Ghahramani and Michael I. Jordan. "Supervised learning from incomplete data via an EM approach". In: *NIPS*. 1993 (cit. on p. 26).

[GZ19] Eric Golinko and Xingquan Zhu. “Generalized Feature Embedding for Supervised, Unsupervised, and Online Learning Tasks”. In: *Information Systems Frontiers* 21 (2019), pp. 125–142 (cit. on p. 23).

[GM10] David Grangier and Iain Melvin. “Feature Set Embedding for Incomplete Data”. In: *NIPS*. 2010 (cit. on pp. 20, 26, 30).

[Gul07] Ranjay Gulati. “Tent Poles, Tribalism, and Boundary Spanning: The Rigor-Relevance Debate in Management Research”. In: *Academy of Management Journal* 50, No. 4 (Aug. 2007) (cit. on p. 1).

[GB16] Cheng Guo and Felix Berkhahn. “Entity Embeddings of Categorical Variables”. In: *ArXiv* abs/1604.06737 (2016) (cit. on pp. 19, 24, 29).

[HK20] John T. Hancock and Taghi M. Khoshgoftaar. “Survey on categorical data for neural networks”. In: *Journal of Big Data* 7 (2020), pp. 1–41 (cit. on pp. 23, 24).

[Has+09] Mostafa M. Hassan, Amir F. Atiya, Neamat El Gayar, and Raafat S. Elfouly. “NOVEL ENSEMBLE TECHNIQUES FOR REGRESSION WITH MISSING DATA”. In: *New Mathematics and Natural Computation* 05 (2009), pp. 635–652 (cit. on p. 26).

[Hue+08] Christian Huemer, Alexander Schmidt, H. Werthner, and Marco Zapletal. “A UML Profile for the e3-Value e-Business Modeling Ontology”. In: *http://www.alexandria.unisg.ch/Publikatio* (Jan. 2008) (cit. on p. 2).

[Kia+09] Behdad Kiani, Mohammad Gholamian, Aso Hamzehei, and Seyed Hossein Hosseini. “USING CAUSAL LOOP DIAGRAM TO ACHIEVE A BETTER UNDERSTANDING OF E-BUSINESS MODELS”. In: 7 (Jan. 2009), pp. 159–167 (cit. on p. 2).

[LBH15] Yann LeCun, Y. Bengio, and Geoffrey Hinton. “Deep Learning”. In: *Nature* 521 (May 2015), pp. 436–44 (cit. on p. 1).

[Lee+19] Jaehun Lee, Taewon Suh, Daniel Roy, and Melissa Baucus. “Emerging Technology and Business Model Innovation: The Case of Artificial Intelligence”. In: *Journal of Open Innovation: Technology, Market, and Complexity* 5 (July 2019), p. 44 (cit. on p. 1).

[MZ20] Yixuan Ma and Zhenji Zhang. “Travel Mode Choice Prediction Using Deep Neural Networks With Entity Embeddings”. In: *IEEE Access* 8 (2020), pp. 64959–64970 (cit. on p. 24).

[MM20] Hannes De Meulemeester and Bart De Moor. “Unsupervised Embeddings for Categorical Variables”. In: *2020 International Joint Conference on Neural Networks (IJCNN)* (2020), pp. 1–8 (cit. on pp. 25, 53).

[Mik+13] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. “Efficient Estimation of Word Representations in Vector Space”. In: *ICLR*. 2013 (cit. on p. 24).

[Mur+19] Della Murbarani Prawidya Murti, Utomo Pujianto, Aji Prasetya Wibawa, and Muhammad Iqbal Akbar. “K-Nearest Neighbor (K-NN) based Missing Data Imputation”. In: *2019 5th International Conference on Science in Information Technology (ICSITech)* (2019), pp. 83–88 (cit. on p. 25).

[Nie18] Michael A. Nielsen. *Neural Networks and Deep Learning*. misc. 2018 (cit. on p. 17).

[Ost04] Alexander Osterwalder. “The business model ontology a proposition in a design science approach”. In: 2004 (cit. on pp. v, 2).

[OPT10] Alexander Osterwalder, Yves Pigneur, and Christopher Tucci. “Clarifying Business Models: Origins, Present, and Future of the Concept”. In: *Communications of AIS* 16 (June 2010) (cit. on p. v).

[Par+20] Saeid Parvandeh, Hung-wen Yeh, Martin P. Paulus, and Brett A. McKinney. “Consensus features nested cross-validation”. In: *Bioinformatics* (2020) (cit. on p. 16).

[Pas+19] Adam Paszke, Sam Gross, Francisco Massa, et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, et al. Curran Associates, Inc., 2019, pp. 8024–8035 (cit. on p. 30).

[Ped+11] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 12).

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “GloVe: Global Vectors for Word Representation”. In: *EMNLP*. 2014 (cit. on p. 24).

[PC06] George Pohle and Marc Chapman. “IBM’s global CEO report 2006: Business model innovation matters”. In: *Strategy Leadership* 34 (Sept. 2006) (cit. on p. 1).

[Śmi+19] Marek Śmieja, Lukasz Struski, Jacek Tabor, and Mateusz M. Marzec. “Generalized RBF kernel for incomplete data”. In: *Knowl. Based Syst.* 173 (2019), pp. 150–162 (cit. on p. 26).

[Śmi+18] Marek Śmieja, Lukasz Struski, Jacek Tabor, Bartosz Zieliński, and Przemysław Spurek. “Processing of missing data by neural networks”. In: *NeurIPS*. 2018 (cit. on p. 26).

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *NIPS*. 2014 (cit. on p. 25).

[tea20] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020 (cit. on p. 28).

[TL17] David J. Teece and Greg Linden. “Business models, value capture, and the digital enterprise”. In: *Journal of Organization Design* 6 (2017), pp. 1–14 (cit. on p. 7).

[TNA94] Volker Tresp, Ralph Neuneier, and Subutai Ahmad. “Efficient Methods for Dealing with Missing Data in Supervised Learning”. In: *NIPS*. 1994 (cit. on p. 26).

[Van+13] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. “OpenML: Networked Science in Machine Learning”. In: *SIGKDD Explorations* 15.2 (2013), pp. 49–60 (cit. on p. 32).

List of Figures

List of Tables

Colophon

This thesis was typeset with $\LaTeX\,2_\varepsilon$. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.

Declaration

You can put your declaration here, to declare that you have completed your work solely and only with the help of the references you mentioned.

Paderborn, November 15, 2021

Yamini Punetha