Application of Natural Language Processing and Machine Learning in E-Commerce

Predicting Product Taxonomy

Master Thesis

by

Shoney Arickathil

Student no: 11017678

29th September 2023

Masters in Applied Computer Science

SRH University Heidelberg
School of Information, Media and Design

Primary Supervisor:	**Prof. Dr. Gerd Moeckel**
Secondary Supervisor:	**Prof. Dr. Andreas Jäger**

Affidavit

I, **Shoney Arickathil**, solemnly affirm and declare:

- that I have independently composed the chapters of the master thesis for which I am named as the author.
- that I did not use any other sources and additives but the once specified.
- that I did not submit this work for any other examination procedure.

Heidelberg, 29th September 2023 ______________________________

Abstract

Shoney Arickathil, School of Information, Media and Design, SRH University Heidelberg

Master Thesis

Application of Natural Language Processing and Machine Learning in E-Commerce

Predicting Product Taxonomy

Business-to-business or business-to-consumer transactions conducted online is increasing substantially. Manufacturers and suppliers rely on various sales channels such as an E-commerce website to gain a market share. E-commerce platforms fetches manufacture's or supplier's version of multilingual product information including the products' taxonomy from the Product Information Management (PIM) systems. Vocabulary used in defining a product taxonomy may differ from one source to another. In E-commerce platform, mediation of machine learning model to predict the product taxonomy eases the process of product information deployment.

In this thesis, the author developed a supervised text based classification model to predict the product taxonomy using open source deep learning framework PyTorch. Author described the data pre-processing methods, architecture of Recurrent Neural Network (RNN), parameter tuning, model evaluation and deployment. The concepts of neural networks, classification problem and machine learning process were analyzed from a mathematical point of view.

Initially, research on impact of well-defined product taxonomy on a recommendation system, virtual agents, Search Engine Optimization (SEO), in an E-commerce was conducted. The concept of supervised learning and classification methods was presented. Recent developments and research related to E-Commerce and its reliance on Natural Language Processing (NLP) and machine learning algorithm were shown. In the following chapters, the step by step implementation of an RNN based classification model were described. Finally, the model was evaluated using the confusion matrix.

Comparing the falsely predicted category along with the actual category, indicated some records were false negative. Meaning the model predicted the category correctly. However, due to prior human error the actual category itself was incorrectly saved. Author concluded that leveraging this thesis study with the image based machine learning algorithm to image captioning, image translation, image classification can generate better product taxonomies.

Contents

Chapter 1

Introduction

1.1 Background and context

While working at an e-commerce industry, specializing in sales of automobile parts, author realized that manufactures and suppliers of similar automobile parts often utilize different product taxonomies. Product taxonomy is hierarchical multi level categorization of the product. Suppliers rely on online platforms to reach out to the customers. However, due to inconsistencies in the product taxonomies among various suppliers, directly importing these product taxonomies into the online platform is not feasible. The inconsistency could be due to use of different vocabulary for the similar product. For example, manufacturers of car oil may use the term "Motor oil" as well as "Engine oil" for the same product.

Defining the product taxonomy is a prior and crucial task of product information deployment. A well-defined product taxonomy enables customers to search the product with the fewest possible clicks. Manually defining the product taxonomies requires a significant amount of time and are prone to human error.

The e-commerce industry is growing substantially, and wide range of products are being offered online. Utilization of machine learning model for recommending the products to the customer, auto completing the search text, automatically filtering of products based on the product description provided by the customer are few examples of function facilitated by artificial intelligence. It is important to note that successful execution of all these functionalities primary requirement is to have a well-defined product taxonomy.

Customer review monitoring system uses natural language processing to understand the semantic meaning of the comment. Determining whether a customer is satisfied or dissatisfied based on their review can be a complex task. By understanding the semantic meaning helps to identify the negatively commented review regarding a product or service. The reviews are classified as either positive or negative, similar to the classification problem of predicting the product category.

Understanding the mathematical foundations of neural network is important to develop a robust supervised text classification model for predicting the product taxonomy. Analyzing the research path intersecting between mathematics and Artificial Intelligence leverages the ongoing development of building better machine learning model.

1.2 Research questions

RQ1: What are the use cases in which natural language processing and machine learning model can be utilized in an E-Commerce industry?

RQ2: For a product classification task, how to define product features as an input for machine learning model?

RQ3: How does a machine learn pattern in product features for classification?

- **SRQ1:** What are the mathematical equations behind the learning process?
- **SRQ2:** What are the mathematical reasons for applying certain pytorch functions during the training process?

RQ4: What is the algorithm to train the machine learning model to predict the product taxonomy?

1.3 Methodology

1.3.1 Start with small prototype

Developing a machine learning model to predict the multi level product categories has series of task and is a complex process. In this paper, to reduce the complexity and primarily focus on the classification problem, author choose to develop a model which predicts the lowest level of category only based on one feature that is the name of the product. Once the smaller working prototype is developed, more features can be integrated along with multiple levels of category.

1.3.2 Ideate: Vocabulary pattern

Initial approach of finding a solution to product categorization is to analyze already existing solution to a different type of classification problem. For example, (Robertson, 2023) demonstrates the working of character level RNN. In which, the pattern of series of characters forming a name of a person are classified to predict which language the name belongs to. Analyzing this example author created a vocabulary level RNN to predict the lowest level of category based on the pattern of product name. In this thesis, author has recognized the complexity of task and has opted to base prediction solely on the product feature pattern. Analyzing the confusion matrix of predicted vs actual category will provide a visual evaluation of the model. The result of the model for all the existing data of product will be stored in search analytic engine Elastic search in order to manually check the falsely predicted category.

Machine learning model to text classification for predicting the product taxonomy is a methodology in which a lot of research have been conducted. Ali Cevahir and Koji Murakami, 2016 implements classification model at every level using the k -Nearest Neighbors classifier. Like wise (Gupta et al., 2016) proposes a distributional semantics representation for predicting the product taxonomies. Some classifying methods are supervised classification, decision trees, naive Bayes classifier and max entropy classifiers (Bird, Klein and Loper, 2009)

1.3.3 Understanding Mathematics of Neural Networks.

In this paper, author researches on the relevance of the mathematical concept with respect to the machine learning process. Understanding the math behind reducing the loss for prediction enables to completely understand the algorithms. Especially the training algorithm applies the activation functions. Author describes the theoretical knowledge with reasoning to apply these activation functions. Understanding the mathematical foundations on which the neural networks and classification model works enables to reverse engineer the algorithm of machine learning model.

1.4 Structure of the Thesis

This thesis is divided in three parts. Part 1, that is Chapter 1 and chapter 2 introduces the research topic, Part 2 that is chapter 2 and chapter 3 describes Natural Language Processing and basic concepts of machine learning. The following chapters are the guide to develop a classification model for predicting the product taxonomy.

In the first chapter, author introduces the domain of research, the importance of the research, research questions and methodology.

The second chapter indulges the reader to understand the basics of machine learning and product taxonomy before diving into the implementation.

The third chapter, describes the application of natural language processing into developing knowledge graph.

The forth chapter, is research on mathematical relationship between a basic neural network and its functions such as feedforward and back propagation. This chapter has a detailed description of the machine learning process. Mathematical concepts such as partial derivatives, Mean Squared Error, log of a value, probability distribution is described in this chapter. It also describes Recurrent Neural Network and its mathematical equation.

In the fifth chapter, defining or selecting the features from a document is described. It introduces various analytic engine tools and methods for feature selection. It describes various methods to standardize the text data and preprocess it before making it as an input to the classification model. The chapter focuses on retaining the missing textual data from the dataset from which the classification model will be predicting the product category. The chapter is a guideline on vectoring or transforming the text into machine usable numerical form called feature extraction.

In the sixth chapter, how author classifies the product by representing the features of the product into vector of entire vocabulary of dataset is depicted. The architecture of neural network used for the prediction is described in this chapter. Mathematical equations on probability distribution, Softmax and why to apply log to softmax is formulated.

In the seventh chapter, the author describes the classification models training process. The experimentation and analysis of model with different parameters are conducted. This chapter also describes the python code related to back propagation. Experimentation with learning rate parameter of the model is conducted.

The final chapter, concludes the thesis along with research questions and future scope of research. Author also describes the research boundaries chosen for this thesis.

Chapter 2

Literature Review

2.1 E-commerce and Product Taxonomy

A product taxonomy is a hierarchical representation of product catalog organized logically so that the customer can find the product in the fewest possible clicks. The top levels of hierarchy are general terms and low levels are narrowed down to specific and semantically closer to the product.

In figure 2.1, is a sample of product taxonomy represented in a directed acyclic graph (DAG).

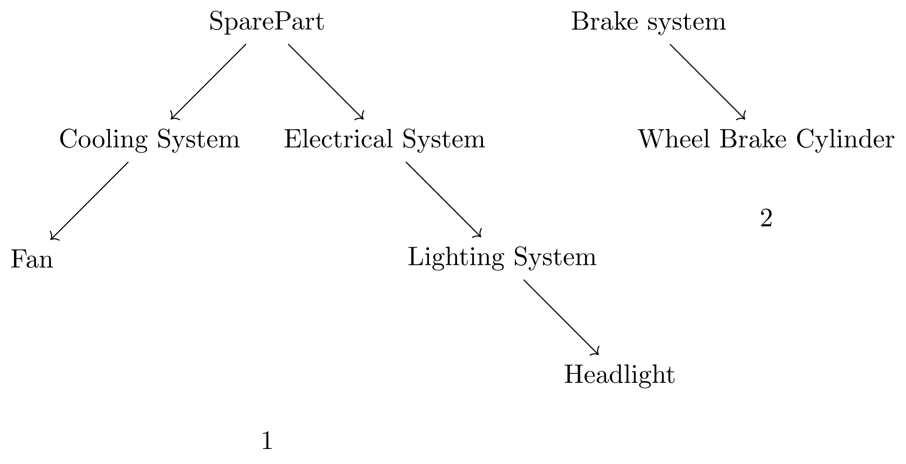


Figure 2.1: Sample - Product taxonomy

Q1: What are the characteristics of sample of product taxonomies illustrated in figure 2.1?

- The top level of root node traversing below to leaf nodes connected with unidirectional edges representing hierarchical levels between the nodes.
- Product taxonomy has different lowest levels of hierarchy. For example, category “Fan” is at third level and “Headlight” being a part of “Electrical Systems” is at fourth level.
- Few categories having similar vocabulary are part of same graph. For example, category with the word “Brake” are connected with the edges. Similarly, the category “Lighting system” and its leaf node “Headlight” share the same prefix or suffix word “light”.
- The root nodes “Sparepart” and “Break system” are isolated from each other.
- The node “Break system” could also be a part of node “Sparepart” and yet the taxonomy will remain well-defined as it is a general term.

Q2: How important is a well-defined product taxonomy in e-commerce (Jessica Howard, 2023)?

- **Simplify product searches**:
  Customer filters the categories to narrow down the search result, enabling them to find the required product in the fewest possible clicks. This improves customer experience.
- **Better product recommendation**:
  Product recommendation system analyze the customer behavior and returns the related products. Customer behavior such as to know which category the customer clicked on but did not purchase the product yet, enabling recommendation of most relevant products.
- **Improves search engine optimization (SEO)**:
  Organizing the products into structured taxonomy, businesses can ensure that their products are found easily. It also helps business drive more organic traffic.

Q3: What are the challenges in defining the product taxonomy?

- **Scalability**:
  The taxonomy requires to be able to update as the number of products in a business grows.
- **Internationalization**:
  Multi lingual taxonomy is required to reach out to customers internationally.
- **Product complexity**:
  Some products may have multiple attributes and these needs to be taken into account while creating the taxonomy. This process can complicate the process.
- **Changing product**:
  Taxonomy of product need to regularly updated as and when there is a change in product.

2.2 Product Information Management (PIM)

Product Information Management system is an information system that facilitates the manufactures and distribution channel such as an e-commerce website with a central hub of product data (Hoogeveen, 2019). PIM system manages products details such name, description, manufacturer, price as well as the products' taxonomy. A PIM system may also include the stock-keeping unit (SKU). SKUs enables the distribution channels to determine which product require reordering and also provides sales data. Distribution channels may also have an extended version of the SKU than the one provided by the suppliers.

PIMs are the point of source to verify whether the products' taxonomy are correct and also if the stock is available before distribution channels make it publically available for sales.

2.3 Product Taxonomy Prediction Approaches

A lot of research have been conducted on methodology for predicting taxonomy. Prediction of taxonomies narrows down to text classification. Text classification is a process of identifying the group or category in which the text belongs to. Few classification methods are decision trees, naive Bayes classifier and max entropy classifiers (Bird, Klein and Loper, 2009).

- Decision tree

  A decision tree is a flowchart that selects labels for input values. This flowchart consists of decision nodes, which check feature values, and leaf nodes, which assign labels.

- Naive Bayes classifiers

  In naive Bayes classifiers, each feature values determining which label should be assigned to a given input value. It begins by calculating prior probability of each label. It is determined by frequency of each label in the training set. Upon combining these prior probability, the likelihood estimation is calculated for each label. The with the highest likelihood estimate is assigned to the input values.

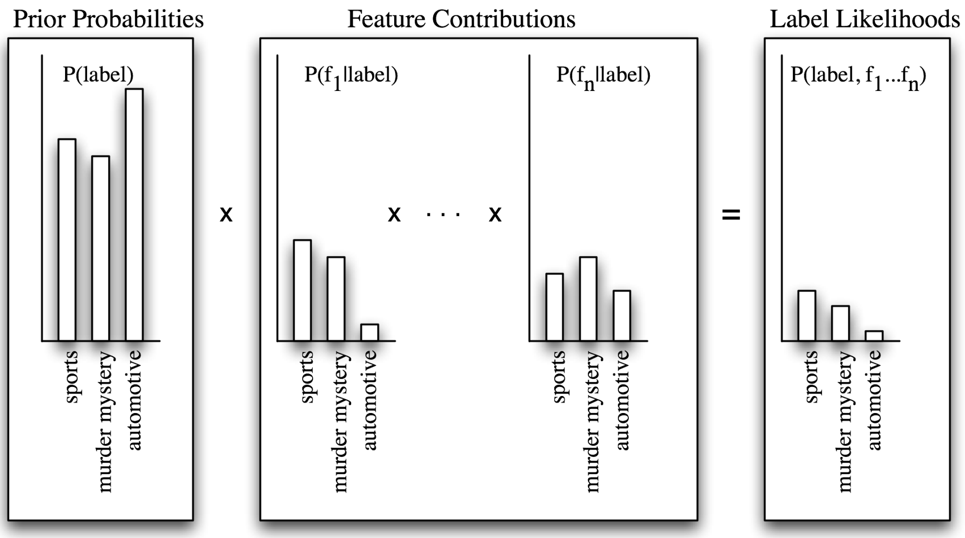


Figure 2.2: Calculating label likelihoods with naive Bayes (Bird, Klein and Loper, 2009)

- Max Entropy

  Like the naive Bayes model, the Maximum Entropy classifier calculates the likelihood of each label for a given input. However, Maximum Entropy classifier model leaves it up to the user to decide what combinations of labels and features should receive their own parameters.

2.4 Supervised Learning

The machine learning model built using existing dataset is called supervised learning. ´Figure 2.3 illustrates the supervised learning workflow. In this paper, author builds a supervised learning model to predict the product taxonomies. The work flow starts with data collection, this steps involves fetching the corpus of raw data. This raw data processed before creating a training data set out of it. The data preprocessing steps involves feature selection, text normalization, data imputation and feature extraction. The training data is passed an input to the model which compares its results with the actual value to self evaluate and learn. If any hyperparameter tuning for the machine learning model is required then it is trained again with new parameters. The model is then evaluated and then deployed for production use.

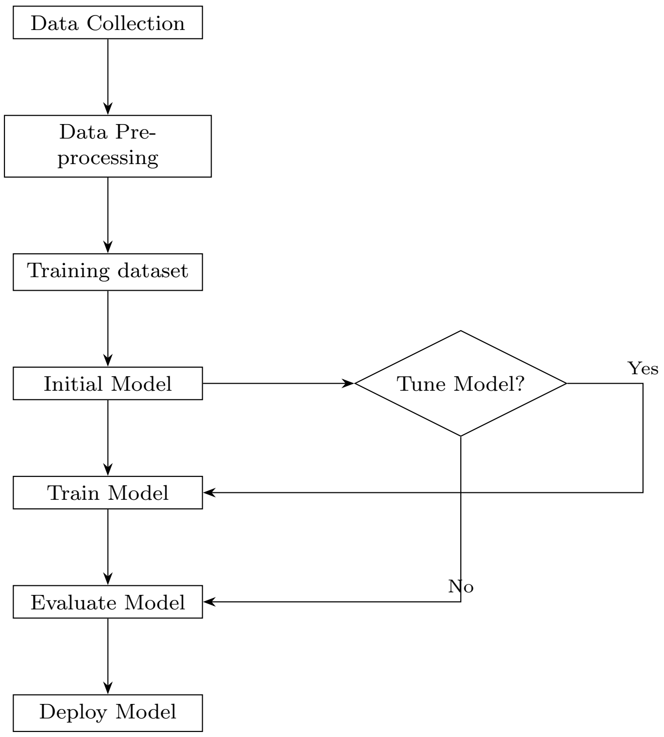


Figure 2.3: Supervised Learning Workflow

2.5 Related Works

(Ali Cevahir and Koji Murakami, 2016) describes implementing classification model by chunking the process to predict at every level using the k -Nearest Neighbors classifier. (Gupta et al., 2016) proposes a distributional semantics representation for predicting the product taxonomies.

Facet Suggestion

(Tagliabue, Yu and Beaulieu, 2020) proposes a machine learning encoder-decoder model architecture that generates the path in catalog taxonomy in real time. This model predicts the best facets - i.e. product categories in real time during type ahead on a eCommerce website search bar. In (Tagliabue, Yu and Beaulieu, 2020) work, prediction is based on user personalized model provided with shopping session and user queries. Combining linguistic and behavioral in-session data, narrows down the search results with personalized facet prediction as shown in figure 2.4.

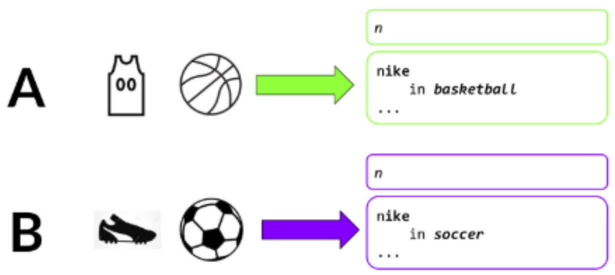


Figure 2.4: Personalized facet prediction. (Tagliabue, Yu and Beaulieu, 2020)

Learning to Rank

Learning to rank is a machine learning algorithm for Information Retrieval (IR). The application of Learn to Rank (LTR) model in E-Com is to list most relevant products based on the search query.

(Karmaker Santu, Sondhi and Zhai, 2017) uses LambdaMART (Burges, 2010) as LTR model as it is well known to work with Web search. As per (Karmaker Santu, Sondhi and Zhai, 2017), E-Com search logs contain four prominent relevance feedback signals.

- clicks : Computed as the ratio of clicks a product receives for a query and its impressions (number of times shown to the user) for the query.
- cart-adds: Computed as the ratio of add-to-carts a product receives for a query and its number of clicks for the query.
- orders: Computed as the ratio of orders a product receives for a query and its impressions for the query.
- revenue: Computed as the ratio of revenue generated by a product for a query and its impressions for the query.

(Karmaker Santu, Sondhi and Zhai, 2017) proposes an algorithm based on the statistics of feedback signals to rank the relevance of product.

Conversational commerce

Conversational commerce is a way of businesses to interact with customers using chatbots. (Messina, 2015) inventor of the hashtags, coined the term conversational commerce. In his post, he states that ecommerce will experience transformation, with integration of Artificial Intelligence (AI) specialized in complex task of online shopping and product research. Conversational commerce offers convenience, personalization and decision support while people are on the go and cannot pay full attention.

One of the use case of conversational commerce is in the reverse logistics- return and exchange. For example, if the user has ordered and wrong product and would like to return it while being on the go. He may use the chatbot assistance and type "I want to return." The chatbot lists the delivered products list and customer selects the product to be returned.

Application of knowledge graphs within the e-commerce industry involves the creation of network of information concerning the products. This knowledge graph can serve as the basis for chatbots to provide answers to user queries.

Chapter 3

Building a Knowledge graph

Graph-based knowledge representation has been researched for decades. A Knowledge graph (KG) acquires and integrates information into an ontology and applies a reasoner to derive new knowledge (Lisa Ehrlinger, 2016). The knowledge base is a dataset with formal semantics that can contain different kinds of knowledge, for example, rules, facts, axioms, definitions, statements, and primitives (Davies, Studer and Warren, 2008, cop. 2006).

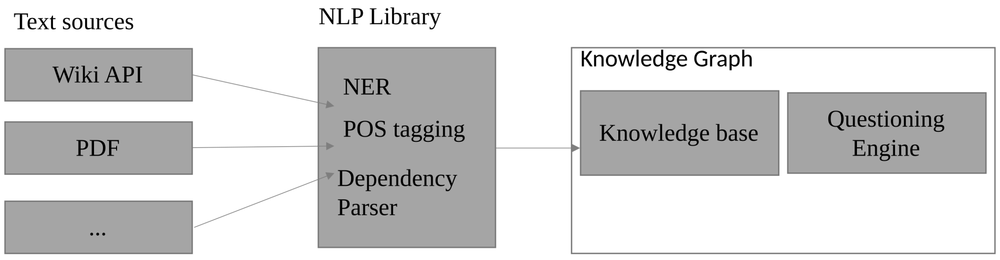


Figure 3.1: Knowledge graph architecture
(Lisa Ehrlinger, 2016, Chapter 4)

Figure 3.1 illustrates the processing of plain text from various sources such as Wikipedia API, PDF into a graph. This abstract architecture represented by Lisa Ehrlinger of a Knowledge graph portraits the assumption that a Knowledge graph is more than a Knowledge base (KB). Knowledge graph is a combination of Knowledge base and Questioning Engine (QE).

A QE is a set of graph of possible questions that could be formed in reference to a KB. Question generation (QG) for comprehensive reading is a challenging task. There are datasets available for QG, one of it is Stanford Question Answering Dataset v1.0 (SQuAD) consisting of questions posed by crowd workers on a set of Wikipedia articles (Pranav Rajpurkar et al., 2016). The limitation with such a data set is that these do not contain unanswerable questions. Building a machine learning model when no answer is supported was out of scope of SQuAD objective (Hernandez and Sam Randall, 2020). Study on automatic question generation from an attention-based sequence learning model (Vaswani et al., 2017) for QG and investigate the effect of encoding sentence- vs. paragraph-level information (Du Xinya, Shao and Cardie, 2017), reduces reliance on handcrafted rule based systems.

3.1 Fetching text corpus

An intriguing application of knowledge graphs within the e-commerce industry involves the creation of a comprehensive information network concerning products. Such a knowledge graph can serve as the basis for chatbots to provide answers to user queries. Converting unstructured text corpus into a directed acyclic graph. Knowledge base serve as a foundation for chatbot applications to traverse through the graph structure, locate relevant leafs nodes, and formulate human-readable response. Wikipedia[1] is primarily an encyclopedia with the additional benefit of heavy linking between articles and without the size constraints of paper articles (Torsten Zesch, 2008).

3.2 Knowledge graph with Networkx

The directed graphs created with Networkx (A. Hagberg, P. Swart and S Chult, 2008) is suitable for representing dependency parsing . A knowledge base are represented as triples of subject,relation and object (SRO). In which the subject and object are nodes are entities of a graph and relation are directed edges or links between the nodes.

What to use as nodes n(x) and edges e(y)?

In table 3.1, triples by POS tags is depicted.

Table 3.1: SRO and POS tags mappings

tripples	**part of speech (POS) tags**
n(subject)	nsubj , pron
n(object)	dobj , pobj
e(relationship)	adp, verb

3.3 Summary

In this chapter, using the dependency parsing of an unstructured sentence, building a knowledge graph is proposed. The process involves textual information on a product from various sources pass through various methods such as dependency parsing, part of speech tagging, named entity relationship mapping. The processed text is created into a directed graph along with the question engine is called the knowledge graph.

Further research has to be conducted on generating knowledge graph. As per (Lisa Ehrlinger, 2016), there are limitations with knowledge graphs as comprehensive reading is a challenging task.

[1]https://www.wikipedia.org/

Chapter 4

Mathematical Foundations of Neural Networks

Understanding the mathematical foundations of the deep neural network is important for searching suitable network architecture. The approach of implementing a model without understanding the mathematical foundation could lead to unexpected failure in getting the desired results from the model. Understanding fundamental mathematics of Artificial Intelligence is necessary as further development in this field is on going. (Kutyniok, 2023) states two different research direction intersects with mathematics and Artificial Intelligence . One which place the network architecture and training process on theory of mathematics and another with a goal to develop mathematical problem settings such as partial derivatives equations aiming to tap the potentials of neural network.

Often it is said that an artificial neural network mimics the structure of a biological brain. As illustrated in figure 4.1, a brain is a deep network of neurons which processes the input signals received from Dendrites and transited with Axon. Before transmitting these incoming signals are integrated or summed together in some way and if resulting signal exceeds some threshold then the neuron will transmit it to other neurons.

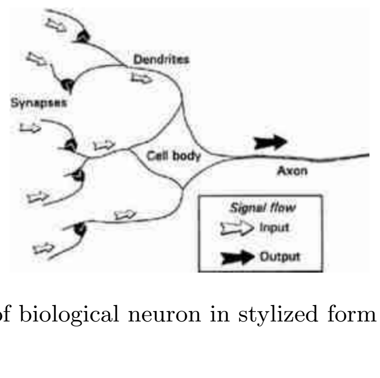


Figure 4.1: Component of biological neuron in stylized form (Gevin, 1997, section 1.1)

The artificial equivalent of a biological neuron is shown in figure 4.2. The Synapses of figure 4.1 are modelled as number or weight being multiplied to each input signals. The weighted signals are summed together by simple arithmetic addition. The resulting value is passed though an activation function representing threshold logic units (Gevin, 1997, section 1.1).

4.1 Neural Networks and Loss Minimization

Neurons

A neural network is a connection of neurons. A Neuron takes input, do some computation and returns an output. Figure 4.2 illustrates a example of basic unit of neuron, which is multiplying each unit of inputs with weight w and weighted input are added with bias b. Finally, the sum is passed though a activation function f.

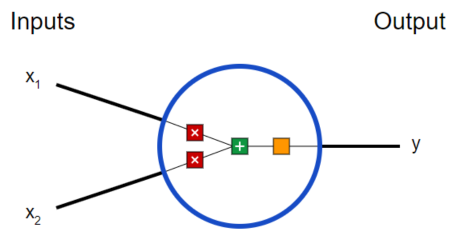


Figure 4.2: Basic unit of Neuron (Zhou, 2019)

$$y = f(x_1 * w_1 + x_2 * w_2 + b) \tag{4.1}$$

The equation 4.1 is the mathematical representation of the sample neuron. The artificial neurons can be added and concatenated into new functions eventually leading to a complex function. A neuron's characteristics is determined by its parameters and activation functions. Each of the activation functions have a mathematical particularity and their practical effects (Lederer, 2023).

Neural Network

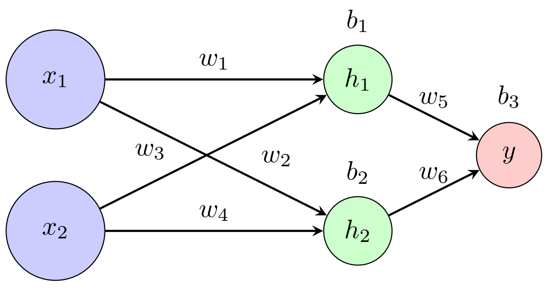


Figure 4.3: Example of feedforward neural network: two neuron hidden layer added between input and output.

Figure 4.3, illustrates addition of two neurons hidden layers h_1 and h_2 between input layer (first) and output layer (last). The data flow is represented with directed arrows, this process of passing input forward to get an output is called feed-forward. A feedforward neural network comprises many functions associated with a directed acyclic graph . These chain of functions form a deep layers of the network and such is the term deep learning being tossed (Goodfellow, Bengio and Courville, 2016, chapter 6).

Training a Network

Training a network simply means to compare the actual machine learned output (y_{ml}) and correct or true output (y_{true}) at each time step and determine how good it is at the current time step and what to do to make it better in next time step. Loss function (L) is a way to determine it.

Let us use the network of figure 4.3 to predict the gender based on weight and height. x_1 is weight and x_2 is height. Table 4.2 is a sample of dataset, genders are valued as 0 and 1. The numerical values of height and weight are adjusted to for easier calculation.

Table 4.1: Sample Training Dataset

Name	**Weight**	**Height**	**Gender**
Mary	-2	-1	1
John	25	6	0

Equation 4.2 is Mean Squared Error (MSE) which is one loss functions.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{ml})^2 \tag{4.2}$$

- n is a number of samples. In table 4.2 there are two entities.
- y is the value the machine needs to generate, in this case "Gender".
- y_{true} is the correct value, for name "Mary" the correct gender value is 1. y_{ml} is the machine predicted value.
- $(y_{true} - y_{ml})^2$ is known as squared error. This loss function takes average of the squared error.

If the machine always returns the value 0, what is the loss?

Table 4.2: Sample Training Dataset

Name	y_{true}	y_{ml}	$(y_{true} - y_{ml})^2$
Mary	1	0	1
John	0	0	0

$$\text{MSE} = \frac{1}{2}(1 + 0)^2 = 0.5 \tag{4.3}$$

Lower loss indicate the network is trained well. The goal of training is minimizing the loss.

Minimizing the Loss

From equation 4.1 it is known that tweaking weights (w) and biases (b) changes the output value. How do we change the networks weights and biases such that the loss is decreased at each time step? Answer to this question is partial derivatives. Partial derivatives is a method of small change in one of the variable of a multivariable function leading to a small change in the output of the function. For example, making a small change in variable w_1 keeping all other variables constant leading to a small change in Loss L can be represented by equation 4.4.

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{ml}} * \frac{\partial y_{ml}}{\partial w_1} \tag{4.4}$$

Referring to the network architecture of figure 4.3 w_1 does not affect h_2 but only h_1, the equation 4.4 can be rewritten as equation 4.6.

$$\frac{\partial y_{ml}}{\partial w_1} = \frac{\partial y_{ml}}{\partial h_1} * \frac{\partial h_1}{\partial w_1} \tag{4.5}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{ml}} * \frac{\partial y_{ml}}{\partial h_1} * \frac{\partial h_1}{\partial w_1} \tag{4.6}$$

Calculating partial derivatives by working backwards is known as back propagation.

The final output of the network can be represented as equation 4.7.

$$y_{ml} = o_1 = f(w_5 h_1 + w_6 h_2 + b_3) \tag{4.7}$$

Stochastic gradient descent (SGD)

Optimization problem is to find the value for set of variable that minimizes the scalar-valued loss function (Murphy, 2022, Page 273). SGD is an optimization algorithm to change weights and biases at time step t. Weight of the next time step (w_{t+1}) is calculated by subtracting partial derivatives of Loss function with respect to the weight of current time step (w_t). The constant l represents the step size or learning rate. The resulting equation is given 4.8.

$$w_{t+1} = w_t - l\frac{\partial L}{\partial w_t} \tag{4.8}$$

4.2 Probability Distribution

A probability distribution is the information of how likely a random variable or set of variables is to take each of its possible states. The way the probability distribution is described depended on whether the variables are discrete or continuous. A discrete random variable has finite or countably infinite number of states. (Goodfellow, Bengio and Courville, 2016, Page 54).

In this project, the random variables are vector-valued product features denoted as $\mathbf{x}$ and its value as x. The probability that variable $\mathbf{x} = x$ is denoted by $P(x)$, with a probability of 1 indicating that $\mathbf{x} - x$ is certain and probability of 0 indicating an impossible event (Goodfellow, Bengio and Courville, 2016, Page 55).

The probability values ranging between 0 and 1 are called Probability Mass Function (PMF) denoted as capital P. To ensure that PMF provides valid probabilities the function P must satisfy following properties.

- Domain of PMF must be defined for all possible states of random variable x. It should specify probabilities for each value x. For example, of the variable of vector representation of product name “X” the P must be defined for all categories.

- The $P(x)$ must satisfy $0 \leq P(x) \leq 1$.
- Impossible events have probability 0.
- Certain events have probability 1.
- Normalization property indicating sum of all probabilities equals to 1.

x	x_1	x_2	x_3	$\ldots$	x_n
$P(x)$	P_1	P_2	P_3	$\ldots$	P_n

Table 4.3: Probability Distribution $P(x)$ for Random Variable x

where $P_i > 0$, $i \equiv 1$ to n and $P_1 + P_2 + P_3 + \ldots + P_n = 1$

Multinoulli Distribution or Categorical Distribution

Multinoulli or Categorical distribution is a type of probability distribution over a single discrete variable with k different states or categories, where k is finite and probability of each category is separately specified.

- A multinoulli distribution is parametrized by a vector $\mathbf{p} \in [0, 1]^{k-1}$, where each element of $\mathbf{p}$ represents the probability of the i-th state.
- The probability of the final, k-th state is given by $1 - \sum_{i=1}^{k-1} p_i$.
- $\sum_{i=1}^{k-1} p_i \leq 1$ hence, the probabilities do not exceed 1.

Multinoulli distributions are often employed to model distributions over categories of objects. In this context, we do not assume that state 1 has a numerical value of 1, and so forth. The values in $\mathbf{p}$ represent probabilities associated with different categories or states, and they do not carry a numerical interpretation.

Moreover, when working with multinoulli-distributed random variables, it is typically unnecessary to compute expectations or variances since these distributions are used to describe categorical outcomes where numerical operations may not be meaningful (Goodfellow, Bengio and Courville, 2016, Page 60, 61).

4.3 Activation Functions and their Propertise

4.3.1 Softmax

The softmax function is often used to predict the probabilities associated with a multinoulli distribution (Goodfellow, Bengio and Courville, 2016, Page 79).

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} \tag{4.9}$$

Equation 4.9 is the formula for Softmax function. It applies the standard exponential function to each element x_i of the input vector $\mathbf{x}$ and normalizes these values by dividing by the sum of all these exponential.

```
>>> import numpy as np
>>> a = [1.0, 2.0, 3.0, 4.0, 1.0, 2.0, 3.0]
>>> np.exp(a) / np.sum(np.exp(a))
array([0.02364054, 0.06426166, 0.1746813 , 0.474833  , 0.02364054,
       0.06426166, 0.1746813 ])
>>> sm_val =np.exp(a) / np.sum(np.exp(a))
>>> np.sum(sm_val)
0.9999999999999999
>>>
```

Listing 4.1: Python example for softmax

Listing 4.1 is an example of applying Softmax function to an array of positive integers. It is important to note that the sum of all the softmax value is approximately equal to 1. The output has most of its weight where the “4” was in the original input. This is what the function is normally used for: to highlight the largest values and suppress values which are significantly below the maximum value.

Reasons to apply log to $SoftMax$ function

$$\log(\text{Softmax}(x_i)) = \log\left(\frac{\exp(x_i)}{\sum_{j=1}^{n}\exp(x_j)}\right) \tag{4.10}$$

Equation (4.10) shows the logarithm of the Softmax function applied to the element x_i in the vector $\mathbf{x}$.

Applying logarithm to the fraction:

$$\log\left(\frac{a}{b}\right) = \log(a) - \log(b) \tag{4.11}$$

Substituting $a = \exp(x_i)$ and $b = \sum_{j=1}^{n}\exp(x_j)$ in equation (4.10), we get:

$$\log(\text{Softmax}(x_i)) = \log(\exp(x_i)) - \log\left(\sum_{j=1}^{n}\exp(x_j)\right) \tag{4.12}$$

- Reduce large numbers: The softmax function normalizes any value from [-infy, +infy] by applying an exponential function. Exponential functions can lead to a very large numbers, resulting in Nan as output. Applying logarithm effectively reduce these large numbers to manageable range.
- Numerical Stability: When dealing with very small probabilities, the product of multiple probabilities can become exceedingly close to zero, leading to numerical underflow. Similarly, when dealing with very large probabilities, the product can exceed the representation range of standard floating-point numbers, causing numerical overflow. By taking the logarithm, these issues are avoided, and the values are represented in a more stable and manageable range (Goodfellow, Bengio and Courville, 2016, Page 79).
- Log Probabilities and Addition: Log probabilities are additive when events are independent. For example, if we have two independent events A and B with probabilities $P(A) = p$ and $P(B) = q$, the probability of both events occurring ($P(A \cap B)$) is $P(A) \times P(B) = p \times q$. Taking the logarithm, we get $\log(P(A \cap B)) = \log(p \times q) = \log(p) + \log(q)$. This additive property simplifies calculations involving multiple independent events.
- Logarithms and Multiplication: Logarithms convert multiplication to addition. Probabilities of independent events, represented as $P(A \cap B \cap C \cap \ldots)$. Logarithm of the product of these probabilities simplifies the computation to a sum.

4.3.2 Sigmoid or Logistic

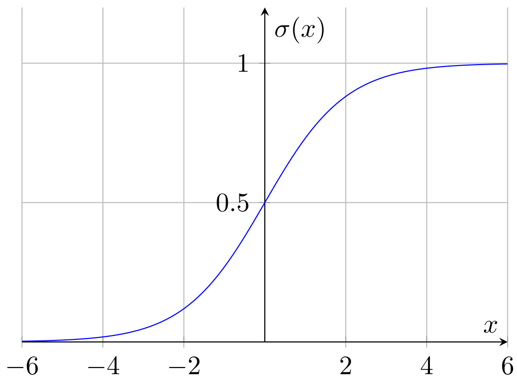
Sigmoid function, also termed as squashing function as the values are bounded in an interval. Figure 4.4a shows the graph of a sigmoid function showing a smooth "S" shaped curve. This function is denoted with greek letter σ (sigma) and is defined as (Lederer, 2023, section 2.1):

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{4.13}$$

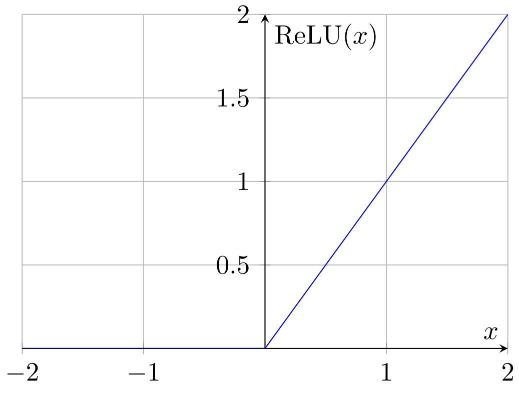
- If x to the sigmoid function is a small negative number, the output is very close to 0.
- If x is a large positive number, then the output is very close to 1.
- If x is 0, the output is 0.5.
- Types of sigmoid functions are logistics, artan, tanh and softsign.

4.3.3 ReLU Rectified Linear Unit

In electrical engineering, a rectifier is a device that converts alternating current to direct current. Similarly, this function lets positive inputs pass unaltered but cuts negative inputs as shown in figure 4.4b. Limitations of ReLU is dying-relu phenomina, which is similar to vanishing gradient decent. The solution to it is leaky-ReLu (Lederer, 2023, section 2.2.2).



(a) Sigmoid activation function



(b) ReLU activation function

Figure 4.4: Activation functions

4.4 RNN - Forward and Backward Propagation

When a feedforward neural network are extended to include feedback connections, they are called Recurrent Neural Networks (RNNs) (Goodfellow, Bengio and Courville, 2016, Chapter 6, Page 164). Recurrent Neural Networks (RNNs) is best suited for machine learning problems that involve sequential data such as sequence translation, generation and classification. A recurrent neural network is specialized for processing sequence of values $x^{(1)}, \dots, x^{(t)}$. The prediction of output y^t also depends on the hidden state of the system h^t, which gets updated overtime as the sequence is processed (Murphy, 2022, Chapter 15, Page 501).

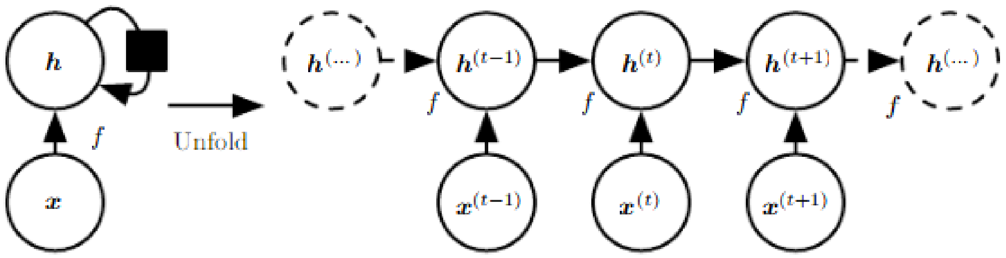


Figure 4.5: A Recurrent Neural Network with no output (Goodfellow, Bengio and Courville, 2016, Page 370)

Figure 4.5 illustrates the connected hidden units indicating the output of previous hidden state $h_{(t-1)}$ is the input of current hidden state $h_{(t)}$. The network can be seen as unfolded computational graphs, here each node is associated with one time instance. This machine learning algorithm store information of previous states and holds the memory.

Equation 4.14 depicts the common representation of RNN to define the hidden units.

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta) \tag{4.14}$$

Graph Un-rolling and Parameter Sharing

Here graph is referred to as the structure of a set of computations, such as the data flow from input layer to output layer for mapping the loss. Unfolding the equation 4.14 for $t = 3$ times, we obtain:

$$h^{(3)} = f(h^{(2)}, x^{(2)}; \theta) = f(f(h^{(1)}, x^{(1)}; \theta)) \tag{4.15}$$

The equation 4.14 is recurrent without the external signal $x^{(t)}$. This expression can be represented by directed acyclic graph . Unfolding this graph results in the sharing of parameters across a network structure.

Sequence Generation

In this type, the output sequence is generated one token at a time. At every step a sample output y^t from the hidden state h^t is again feed back in to the model to get new state h^{t+1} as illustrated in figure 4.6. This architecture could be applied in image captioning method (Murphy, 2022, section 15.2.1).

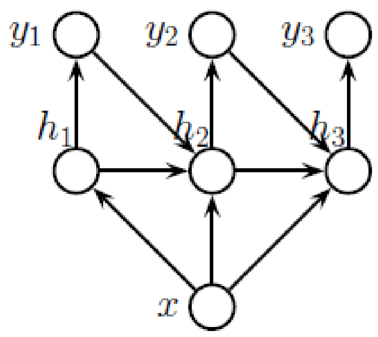


Figure 4.6: A Recurrent Neural Network for generating a variable length output sequence given an optional fixed length input vector. (Murphy, 2022, section 15.2.1)

Sequence Classification

This architecture take variable length sequence as input and predict an output vector. The output of the hidden state is depended on the previous hidden state and also on input sequence as shown in figure 4.7.

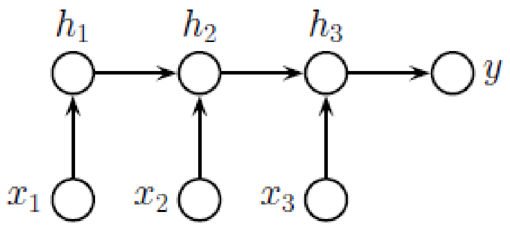


Figure 4.7: A RNN for classification (Murphy, 2022, section 15.2.1)

Sequence Translation

There are two types, one in which the input and output are of same length (aligned) and other in which input and output sequence are of variable length(unaligned). Application of unaligned sequence translation is machine translation.

4.5 Summary

In this chapter, the mathematical theories on neural network is described. Author illustrates mathematical equation for feedforward computational graphs and back propagation. Some mathematical concepts such as partial differentials to calculate the effect of small change in any one of the variable of a neural network is explained with step by step equation. Author describes the using sigmoid, softmax activation function in the output layer of neural network to bound the output values. In this chapter, author explains the Mean Squared Error loss function with a sample of training data. Author discusses the approach for numerical stability to avoid overflow and underflow of value resulting into a not-a-number values.

Further, the author describes multi level classification task. In this chapter, an overview of the Probability Distribution and one of its type Multinoulli Distribution is given. Author describes the basic architecture of a Recurrent Neural Network and explains the reason of using this network for categorizing the products based on product name.

Chapter 5

Data Preprocessing

5.1 Feature Selection

Data collection is simply a process of fetching the data from the data source. However, feature selection is a process of selecting fields from which the target field will be predicted. Suppliers provide stock details along with associated product taxonomy. Importing well-defined taxonomy requires to train the classification model with already existing correct pair of features and label. The already existing pair of features and the correct label for each input is the training data set or training corpora. The classifier built on such a training corpora is termed as supervised classification (Bird, Klein and Loper, 2009).

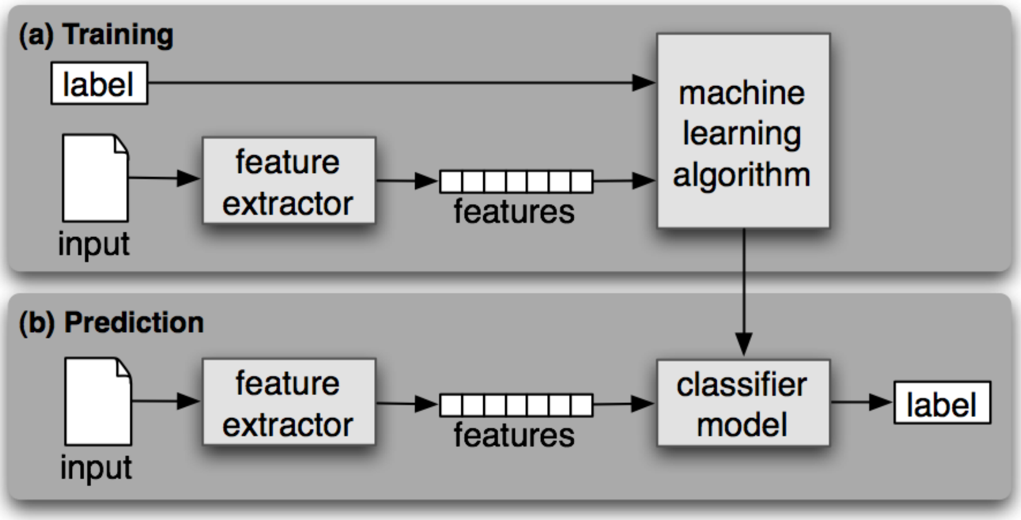


Figure 5.1: Supervised classification (Bird, Klein and Loper, 2009)

As illustrated in figure 5.1, during training, a pair of feature set and label are fed in to the machine learning algorithm to generate a classifier model. During prediction, same feature set are fed into the model to predict the label. Refer to chapter 5.5 for details on feature extraction.

In this project, author choose to create a classification model with only one feature that is the name of the product. However, author identifies the potential features to classify the target of n levels of category. These features have been identified based on the understanding of domain of the product. For products with many details will have many more features to be taken into consideration. In such case, selection based on common understanding of the nature of the product may not be sufficient. Hence, Scikit learn (Buitinck et al., 2013) provides methods to define feature appropriately.

5.1.1 Feature Selection Methods

A feature represents a dataset fine-tuned to serve as a training data for machine learning model. Choosing obvious set of features could get a decent performance on classification task. However, carefully constructed relevant set of features could impact the learning ability and have a significant gain in performance. The feature selection API from Scikit learn (Buitinck et al., 2013) also refers to it as dimensionality reduction.

(Bird, Klein and Loper, 2009) describes some approaches for feature selection as following:

- Kitchen sink

  All the features are initially included later each are checked to determine whether they are actually helpful.

- Error analysis

  The data corpus is split into development-set and test-set. Development-set is subdivided into training set and dev-test set. Training set is used to train the model, dev-test set is employed for error analysis. The individual error cases are examined for wrongly predicted labels.

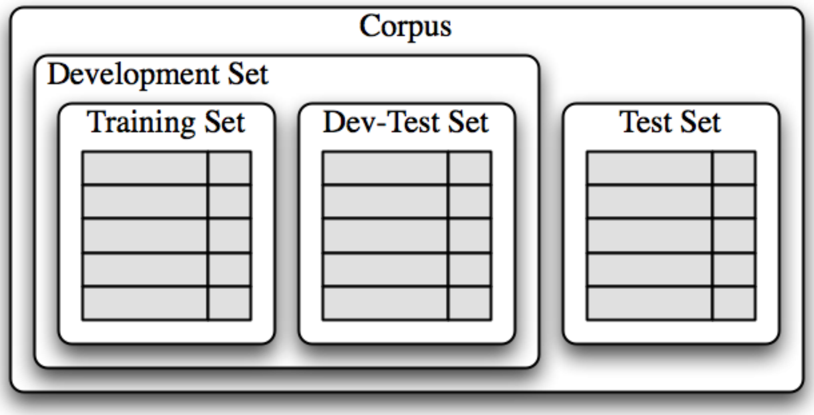


Figure 5.2: Corpus organization for supervised classification (Bird, Klein and Loper, 2009)

Methods for feature selection with Scikit Learn API are:-

- Removing features with low variance
- Univariate feature selection
- Recursive feature elimination

5.1.2 Obvious Feature Selection

In this paper, the datasets used are of an ecommerce business specializing in automotive industry domain. Secondary dataset used is from the TecDoc catalogue by TecAlliance [1].

Table 5.1 lists the obvious features for an ecommerce domain. In this paper, five levels of categories are taken into consideration. The number of category levels differ for each product. Consider an example of category tree with three levels.

sparepart/cooling-system/thermostat

In this example, the lowest level of category is “thermostat” as level 4 and level 5 does not exist and will be represented by black or NaN. These blank or NaN value records can go through data imputation as discussed in chapter 5.3.

Table 5.1: Feature description

No	Feature	Description
1	Product name	normalized form of name
2	Category tree	multi level categories
3	Description	Description with html tags
4	Short description	product info displayed
5	Supplier	supplier of the product
6	Manufacturer	manufacturer of the product
7	Price	Price of the product
8	Dimension	Height, weight of the product
9	(n-1) number of category levels	n is the total category level, one of which will be the target

5.1.3 Fetch Existing Product Taxonomy (Corpus) using Elasticsearch

Elastic search [2] is a fast and scalable search and analytics engine. It can build a powerful AI and machine learning enabled search experience. In this paper, author fetches labeled dataset of products to serve as the training data for the machine learning.

The above code fetches indexed product name and its lowest hierarchy level category.

Table 5.2: Index: english-name-category statistics

Samples total	22160
Dimensionality	2
Features	name
Label	category

Table 5.2 highlights the total number of products and category and name of fields considered as feature and label.

[1]https://www.tecalliance.net/
[2]https://www.elastic.co/

5.2 Text Normalization

Text normalization is a process of transforming the document into a standard and consistent form of a text. A document is a text from single source. Some examples of document are list of product names from databases, a pdf file, texts retrieved from web scraping. Text normalization enables to perform required operations on the text as the text inputs are consistent across the document. The process of text normalization varies based on the type of text that needs to be normalized. There is no standard method for text normalization task. In this paper, text normalization of product name and category from the database source is performed.

In table 5.3, displays the difference in text before and after normalization.

Table 5.3: Sample of text normalization effect

	Name	**Category**
Before	Company V10-4245 Stoßdämpfer	Stoßdämpfer
After	company v104245 stossdaempfer	stossdaempfer

5.2.1 Lower Case Conversion

Using Pandas - vectorized string method, a text can be lower cased across the document. These methos exclude the missing / NA values automatically (McKinney, 2010), shown in code 5.1.

```
    df_de["name"]= df_de["name"].str.lower()
```

Listing 5.1: Pandas vectorized string method

5.2.2 HTML Parsing

Probability of getting HTML tags in a document increases if the source of document is from web scraping or also if the document from database source has inline HTML tags. In such case, fetching normal texts from an HTML text is required. One of the method would be to use regular expressions to get text in between angel brackets <>text <>, as in code 5.2. However, this method would require to have the tags to be well-formed.

```
    text = re.sub('<[^>]*>', '', text)
```

Listing 5.2: Regular expression to get text from html

The better option would be to use Beautiful Soup [3] a Python library for pulling data out of HTLM and XML, using code 5.3.

```
    # Remove html tags
    soup = BeautifulSoup(doc, 'html.parser')
    text =soup.get_text()
```

Listing 5.3: Beautiful soap API to get text from html

[3]https://www.crummy.com/software/BeautifulSoup/bs4/doc/

5.2.3 Cleaning Text

A product detail text may contain non-word characters represented in regular expression as \W. It may also contain space separated decimals describing the dimensional values of the product such as weight, height. Regular expressions could be used for removing the non-character words and spaces between the decimal values, using code 5.4.

```
    text = (re.sub('[-\W]+', ' ', text))
    text = (re.sub('(?<=\d) (?=\d)', '', text))
```

Listing 5.4: Clean text with Regular expression

5.2.4 Normal form D - Unicode database

A product name in Deutsch language may contain umlaute characters such as Ä. Pythons unicodedata [4] module provides access to the Unicode Character Database (UCD) which defines the character properties for all Unicode characters. To change Ä to A, the Normal form D (NFD) should be applied which translated each character into its decomposed form. Further the text can be refined with the general category assigned to the character as string. Nonspacing mark characters [5] are represented as "Mn". These characters could be removed by referring the character category as shown in code 5.5.

```
    ''.join(
        c for c in unicodedata.normalize('NFD', text)
        if unicodedata.category(c) != 'Mn')
```

Listing 5.5: NFD normalization

Another option is to replace the German characters to equivalent English sounding characters. Table 5.4 lists the common German characters and their equivalent English sounding series of characters. In this way, the reverse translation is easier.

Table 5.4: German characters to English

Ä	Ö	Ü	ß
Ae	Oe	Ue	ss

5.2.5 Avoided normalization processes

In this project, some commonly practiced normalization process are being avoided. Which once and why are described below.

- Stemming

  Stemming is a process of striping off any affixes. Natural Language Processing tools such as NLTK provide stemmers. The Porter and Lancaster stemmers follow their own rules for stripping affixes (Bird, Klein and Loper, 2009). Author required not to deviate from the original vocabulary of features.

- Lemmatization

  Lemmatization remove the affixes if the word is in the used libraries' dictionary. The vocabulary for corpora and any third party library such as WordNet cannot be synced.

[4]https://docs.python.org/3/library/unicodedata.html
[5]https://www.compart.com/en/unicode/category/Mn

5.3 Impute missing text data

Machine leaning algorithms requires that their inputs have no missing values. Missing values encoded as NaNs or blanks are incompatible with estimators which represents as the numerical values and assumes all values have and hold meaning. Discarding the entire row and/or columns of a dataset containing the missing values could lead to losing valuable data. The process to fill (impute) missing values are referred to as imputation.

5.3.1 Feature imputation / Regression imputation or Predictive Imputation

Machine learning algorithm to predict values in a categorical variable based on other available features. Table 6.1 states the categorical features from the all the features mentioned in 5.1.

Table 5.5: Identify categorical features

No	Feature	Categorical
1	Product name	No
3	Description	No
4	Short description	No
5	Supplier	Yes
6	Manufacturer	Yes
7	Price	No
8	Dimension	Yes

The *dimension* column is also considered as a categorical value as it has few unique values.

If *supplier* is the missing value from the set of row. The missing *supplier* data must be predicted only from the available list of suppliers. A supervised machine learning with labeled dataset to train the model to classify the available features data to predict *supplier*. A tensor size of the number of unique *supplier* will be the output of the model. In an iterated round-robin fashion, at every step a missing feature column y and other feature columns treated as input x predicts the missing values of y. Regression imputation is more accurate than mode imputation on categorical value.

5.3.2 Mode imputation

Replacing the most frequent categorical value in a column is known as mode imputation. Pandas.DataFrame.mode (McKinney, 2010) function returns most often value. Code snippet 5.6 will fill the missing values for each column using its own most frequent value.

```
    df = df.fillna(df.mode().iloc[0])
```

Listing 5.6: Pandas DataFrames mode function

Mode imputation on categorical data are prone to fill incorrect data if the missing data is not the most frequent one.

5.3.3 *k* -Nearest Neighbors (KNN) imputation

k -Nearest Neighbors is a supervised learning algorithm and is used to search dataset with the most similar elements to a given query element, with similarity defined by a distance function. This imputation method is suitable for categorical data.

The most common distance metrics functions are:-

1. Euclidean distance.
2. Manhattan distance.
3. Hamming distance.

sklearn.neighbors.KNeighborsClassifier (Pedregosa et al., 2011) does an instance-based learning, it does not construct a model, but stores instance of training data. Classification is computed from majority vote of nearest neighbors of each point.

The k-neighbors classification implements learning based on k nearest neighbors of each query point, where k is integer value specified by user. The optimal k value can be evaluated by iterating the classifier with different *n_neighbors* Parameter and finding the minimum value of error rate, refer code 5.7. Larger k suppresses the effects of noise, but makes the classification boundaries less distinct.

```
error_rate = []
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))

print("Minimum error:-",min(error_rate),"at K =",error_rate.index(min(
error_rate)))
```

Listing 5.7: Find optimal k value in k -Nearest Neighbors

5.3.4 KNN model implementation

In table 5.6 there are 7 columns. Target column is **Category-Level-3** which is the category at level three. As per the example category hierarchy in table 5.6

Table 5.6: Sample features

Name	*product name*
Description	*product description*
Short-Description	*product short description*
Category-Level-1	Spare-Part
Category-Level-2	Cooling System
Category-Level-3	Thermostat
Category-Level-4	NaN *product has no level 4 category*

```
from sklearn import preprocessing

X = df.drop(['catL3'], axis = 1)
y = df['catL3']

X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2,
random_state=4)
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
```

```
#Train Model and Predict
k = 3
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
Pred_y = neigh.predict(X_test)
print("Accuracy of model at K=3 is",metrics.accuracy_score(y_test, Pred_y))
```

Listing 5.8: Finding k-nearest neighbor

Code snippet 5.8 performs the following tasks:

- Collect independent data features into the X data frame and target field into a y data frame.
- Split data into training and testing.
- Import the classifier model from sklearn library and fit the model with *k* value equal to the optimal value.

5.3.5 Mean/median imputation

Pandas.DataFrame.median (McKinney, 2010) returns the median value, refer code 5.9. These can be applied to the features represented in numerical value. In reference to table 6.1, *price* could be a column on which median value can be filled. However, using median to fill missing value could underestimate or overestimate the value.

1. Median - The mid point value
2. Mean - The average value

```
df = df.fillna(df.median().iloc[0])
```

Listing 5.9: Mean imputation

5.4 Log transformation of data

Log transformation is a mathematical operation applied on data to change its scale and to reduce its skewness and to make data more symmetric. Skewness is a measure of the asymmetry of a distribution, and a right-skewed distribution has a long tail on the right side of the distribution (Ma, 2019). The Numpy library provides natural logarithmic function to apply log transformation on array of data.

5.5 Feature extraction

Transforming text or image data into numerical representation usable for machine learning is called feature extraction. Raw sequence of data cannot be fed directly into a machine learning algorithm as they expect data in numerical vector and in fixed size. Whereas raw text document are of variable lengths.

5.5.1 Count Tokenization

Tokenization is splitting the text in sequential words which can be embedded in a vector space. For example, the normalized product name “abc joint kit drive shaft” can be tokenized into “abc”,“joint”,“kit”,“drive”,“shaft”. Number of occurrence of each token in a document is called counting in terms of numerical feature extraction.

5.5.2 Count Vectorization or One-Hot encoding

CountVectorizer class of scikit learn API implements both tokenization and occurrence counting (Buitinck et al., 2013).

Consider a data frame with columns *ProductName* and data value as in tbale 5.7

Table 5.7: Sample data for count vectorization

ProductName
abc joint kit drive shaft
xyz joint kit drive shaft

The length of the vocabulary is the number of unique tokens in the data frame column. In the sample data the vocabulary length is six. The vector has the dimensionality equal to the size of the vocabulary. Adding one in the dimension to each of the word in the vocabulary represents one hot encoding.

5.5.3 *n-grams* Vectorization

Count Vectorization can also be performed on range of grams of words.

```
    bigram_vectorizer = CountVectorizer(ngram_range=(1, 2))
    bigram_vectorizer.fit(df["ProductName"])
```

Listing 5.10: *n-gram* vectorization

The code in listing 5.10, will add additional vocabularies of two words such as ”drive shaft”. This enables to preserve local ordering information.

5.5.4 Pickling the Vectorizer

Pickle [6] module creates a portable serialized representation of python object. Reusing the CountVectorizer object once it has been fit with the document can be enabled by Pickle module, refer code 5.11.

```
    pickle.dump(self.vectorizer, open("vector.pickel", "wb"))
```

Listing 5.11: Pickle vectorization

The memory use will grow as the vocabulary in the text corpus grows. Pickling and un-pickling of vectorizer will slow for large vocabulary. This issue can be tackled by hashing the features.

[6]https://docs.python.org/3/library/pickle.html

Table 5.8: Sample One-Hot encoding

text	**encoding**
abc	[1,0,0,0,0,0]
joint	[0,1,0,0,0,0]
kit	[0,0,1,0,0,0]
drive	[0,0,0,1,0,0]
shaft	[0,0,0,0,1,0]
xyz	[0,0,0,0,0,1]

5.6 Summary

In this chapter, search analytic engine named Elastic search is introduced. The author as a prototype of classification model choose to include only two-dimensionality. One is the feature (product name) and other is the label to be predicted (category.)

Table 5.1 list the features for determining the product taxonomy. These features are listed based on general knowledge and understanding. However, productive method for refining the feature such as error analysis can be utilized for feature selection. Scikit learn API (Buitinck et al., 2013) also provides some functions for feature selection.

Before extracting the features (refer chapter 6) it is important to normalize and standardize the data set. If the text normalization step is skipped then while creating the numerical representation of the text the data will be inconsistent. For example, same text with proper case (Car) or lower case (car) will have two representation even though the semantic meaning is the same.

The methods involved in text normalization are lower casing the text, removing the HTML tags, removing irrelevant numerical data within the text such as product dimensionality, and making the text across the document with one Unicode database.

Missing input values sometimes denoted as blanks or NaN's cannot be considered for transforming into a numerical representation value. Machine learning algorithm feed in input data as a numerical representation of data terms as extracted features. Removing the missing values from the dataset sometime may not be feasible. As this may lead to loss of vital information of the product.

Predicting the missing values, replacing the missing value with most frequent values, finding most similar value with K - nearest neighbors, replacing the values with median or mean values are some methods to replace the missing values from the dataset.

Feature extraction is a process of transforming the normalized text into numerical representation for machine learning. Tokenization of text and Counting are terms to split the text in sequence of words and there occurrence.

In this thesis, author has used the n-grams vectorization method to convert the feature - product name into a vector format. This creates a vocabulary of product name. For example, consider total number of unique words in entire product name column is 100. The name of the product will be a vector shape of 1 with each text represented in one-hot encoded format. Refer table 5.8 for visual representation.

Chapter 6

Defining Model Architecture: Neural networks

6.1 Understanding Pytorch tutorial on *classifying names with a character-level RNN*

(Robertson, 2023) tutorial on *classifying names with a character-level RNN* provides a basic foundation for classification algorithm. In this tutorial, Robertson trains on few thousand surnames from 18 languages of origin, and predicts which language the name is from based on the spelling.

6.1.1 One-Hot vector representation

Robertson uses one-hot vector of size 1 x no_letters (26 letters). A one-hot vector is filled with 0s except for a 1 at index of the letter. For example, letter b is represented as 0,1,0,0...0. To make a word, author joins a bunch of letters into 2D matrix name_length x 1 x no_letters.

Table 6.1: One-Hot vector representation of name James

Letter	**a**	**...**	**e**	**...**	**j**	**...**	**m**	**...**	**s**	**...**
J	0	0	0	0	1	0	0	0	0	0
a	1	0	0	0	0	0	0	0	0	0
m	0	0	0	0	0	0	1	0	0	0
e	0	0	1	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	0	1	0

6.1.2 Vector input and scalar output of the classification model

It is a character level RNN which reads words as a series of characters. Figure 6.1 shows the flow of name information to the neural network as a sequence of characters. One character at a time is feed in classification model built using RNN and outputs the language corresponding to that name.

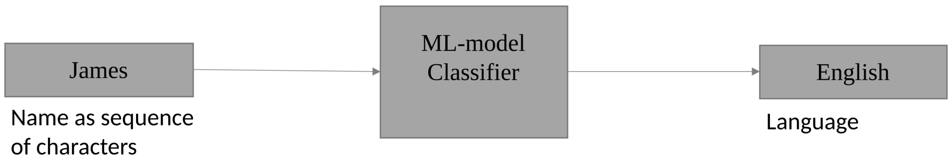


Figure 6.1: Language prediction based on name

6.2 Ideate: Vocabulary level RNN

Inspired from the character level RNN mentioned in section 6.1, author predicts category based on the name of the products by creating a vocabulary level RNN. As described in section 5.5.2, product names are converted into one-hot encoded format. Vector representation of vocabulary in product name across the data frame is created. These encoded pattern of name serves as an input to the machine learning model. Model learns these patterns and predicts the category in which these patterns belong to. In section 6.2.1, the input tensors are modified to fit for use case of predicting category based on product name.

6.2.1 Convert product name to tensors

```
1     def nameToTensor(self,name):
2         vectorizer= pickle.load(open("vector.pickel", "rb"))
3         inputSize=len(vectorizer.vocabulary_)
4         vectorized=vectorizer.transform(list(name.split()))
5         name_tensor=torch.zeros(1, inputSize)
6         for index in vectorized.indices:
7             name_tensor[0][index] = 1
8 
9         return name_tensor
```

Listing 6.1: Convert product name to tensors

Summary of code snippet 6.1:-

1. vectorizer:
   In section 5.5.4, author describes using Pickle module to store vector object. The object contains the vector representation of vocabulary of product names. Code line number 2 loads the object and stores the value in vectorizer variable.

2. vector size:
   Code line number 3 gets the length of the vector vocabulary. The number of unique words in the product name across entire column of *ProductName*

3. transform :
   As described in section 5.5.3, based on the existing vocabulary, vectorizer object's transform function returns token counts out of raw text documents using the vocabulary fitted with fit method or the array of tokens into the one hot encoded vectorized form.

4. torch.zero [1]:
   Initialize a tensor variable filled with scalar value 0.

5. Set 1 for each vectorized index.

For example, consider a vocabulary of 100 unique words in the *ProductName* dataset. The name of the product is *abc joint kit drive shaft*. Assuming the index value of each of the token are as per table 6.2 then the tensor value of the product will be as per table 6.3

Table 6.2: Example: Index value of product name

Token	**Index**
abc	0
joint	10
kit	26
drive	78
shaft	99

Table 6.3: Tensor value of example product name

0	**...**	**10**	**...**	**26**	**...**	**78**	**...**	**99**
1	0	1	0	1	0	1	0	1

6.3 Architecture of Recurrent Neural Network (RNN)

RNN architecture is from (Robertson, 2023) tutorial, this module contains two linear layers which operate on input and hidden state, with *LogSoftmax* layer after the output layer. The class RNN represents the RNN model with an input layer, hidden layer, and output layer. It uses the nn.Linear and nn.LogSoftmax functions from PyTorch's neural network module. The code listing 6.2 is a snippet of python code to define class named RNN.

```python
class RNN(nn.Module):

    def __init__(self, input_size, hidden_size, output_size):

        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)

        self.h2o = nn.Linear(hidden_size, output_size)

        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, current_input, previous_hidden):
        compbined = torch.cat((current_input, previous_hidden), 1)

        next_hidden = self.i2h(compbined)

        output = self.h2o(next_hidden)
        output = self.softmax(output)

        return output, next_hidden

    def initHidden(self):
        return torch.zeros(1, self.hidden_size)
```

Listing 6.2: Recurrent Neural Network (RNN) class

[1] https://pytorch.org/docs/stable/generated/torch.zeros.html

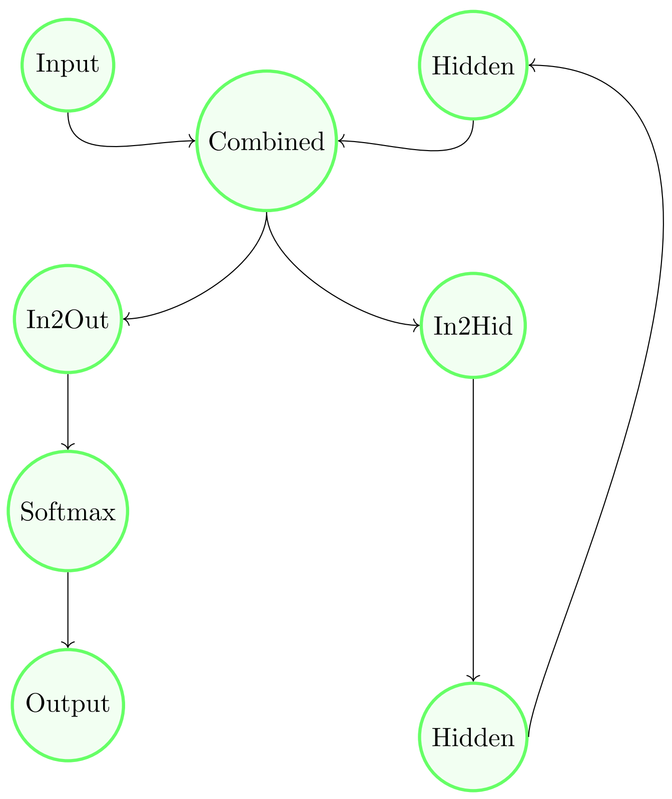
Summary of the RNN class:

- *Constructor (__init__ method):*
  Initializes the RNN with the input size, hidden size, and output size. It creates the layers and initializes the hidden size. The `nn.Linear` objects `i2h` and `h2o` represent the weight matrices for the input-to-hidden and hidden-to-output transformations, respectively. The `nn.LogSoftmax` function applies the logarithm and softmax operations to convert the output into log probabilities.

- *Forward pass (`forward` method):* Takes the current input and previous hidden state as input and computes the forward pass of the RNN. It concatenates the current input and previous hidden state using `torch.cat`, passes the concatenated tensor through the input-to-hidden layer, calculates the output using the hidden-to-output layer, and applies the $LogSoftmax$ operation to obtain the normalized log probabilities.

- *Initialization of hidden state (`initHidden` method):* Returns the initial hidden state, which is a tensor of zeros with dimensions (1, hidden_size).

Figure 6.2: RNN Architecture (Robertson, 2023)



The graphical representation of code listed in 6.2 is illustrated in figure 6.2.

- input_size : Input parameter of class is size of the data. Size of one-hot encoded feature vector.
- hidden_size : Dimensionality of hidden state of the RNN cell.
- output_size : Number of categories in which the input need to be classified.
- i2h : input-to-hidden, a fully connected linear layer gets the next hidden state from the current input and previous state.
- i2o : input-to-output,a fully connected linear layer gets the next output state from the current input and previous hidden state.
- softmax: Layer used for classification.

6.4 Pytorch's *SoftMax* function and its variations

Softmax or normalized exponential.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} \tag{6.1}$$

- Pytorch's nn.Softmax applies the Softmax function to an n-dimensional input Tensor rescaling them so that the elements of the n-dimensional output Tensor lie in the range [0,1] and sum to 1.
- In this representation, x_i is the i-th element of the input vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$. The softmax function calculates the probability of the element x_i being chosen by dividing the exponential of x_i by the sum of the exponentials of all elements in the vector $\mathbf{x}$. This calculation ensures that the resulting probabilities sum up to 1, forming a valid probability distribution over the elements in the vector $\mathbf{x}$.
- The softmax function transforms a vector of logits into a probability distribution, making it useful for tasks like classification, where the model needs to make a decision based on the input data.
- Pytorch's nn.LogSoftmax function applies $\log(\text{Softmax}(\mathbf{x}))$ to an n-dimentional input tensors.

Adaptive LogSoftmax with Loss (ALL)

- Pytorch's (Paszke et al., 2019) nn.AdaptiveLogSoftmaxWithLoss module implements Adaptive LogSoftmax with Loss (ALL).
- Adaptive Softmax (Grave et al., 2016) is effective when large number of classes are to be handled. Traditional Softmax computation becomes computationally expensive and memory intensive if number of output classes are very high. Softmax requires calculating the exponential of the logits for each class. However, when the number of classes or labels are very high, the Softmax based classification becomes computationally expensive.
- Adaptive Softmax counter this issue by clustering the classes according to their frequency of occurrence. A simple strategy to reduce the overall computation time is to partition the labels V into two clusters as V_h and V_l, where V_h denotes the distribution consisting of the most frequent classes, and where V_l are many rare classes. The classifier frequently accesses V_h, which motivates the fact that it should be computed efficiently. In contrast, V_l less frequently, and the corresponding computation can be slower. This suggests defining clusters with unbalanced cardinalities $|V_h| \geq |V_l|$ and probabilities $P(V_h) \geq P(V_t)$.

6.5 Summary

In this chapter, author describes how he ideated the concept of vocabulary based RNN. This method is a modified version of character level RNN (Robertson, 2023). The method of converting the product name into a tensor format is described in this chapter.

Architecture of the RNN and pictorial representation of flow of data within the neural network is depicted. The chapter gives an overview of the activation functions and forward propagation method. The code snippet of the defining architecture of Recurrent Neural Network class is given along with its explaination.

In this chapter, Pytorch's loss functions such as SoftMax, LogSoftMax, Adaptive Softmax and its equivalent mathematical representations is described.

Chapter 7

Training

7.1 Loading the preprocessed data

After processing the document into a standard and consistent from as discussed in chapter 5.2. the normalized form data can be stored in any database for frequently accessing the data for training purpose. In this experiment, the database used is Elastic Search. The author initially trained the classification model with only one feature “Product name” and the target being the lowest level of category. Table 7.1 displays the difference in the number of records after removing the duplicate records on the normalized text. An Index in elastic search is a logical namespace for storing data in JSON format. The indexer names “english-taxonomy-all” contains the data before normalization. “english-taxonomy-normal” contains the data after normalization and removing the duplicate entries.

Table 7.1: Record count before and after normalization

Index	**Features**	**Count**
english-taxonomy-all	Product name, Category	22160
english-taxonomy-normal	Product name, Category	1507

Pandas data frame (McKinney, 2010) provides the *drop_duplicates()* method to remove duplicate entries in the document. Code snippet in listing 7.1 has two methods “normalize” and “clean”. In Pandas (McKinney, 2010), “apply” method executes a function along a specified axis of data frame. Here for every text in the document of data frame variable “df_eng”, the method “normalize” is called to process the data. “normalize” function excepts a “document” as a parameter. These texts are processed to remove any html tags using Beautiful Soup a python library. Using regular expression library the digits in between the characters are removed, white spaces are removed. Any special character within the text has been transformed to its normal form.

```python
    def clean(self):

        self.df_eng["name"] = self.df_eng["name"].str.lower().apply(lambda n:self.normalize(n))
        self.df_eng["category"] = self.df_eng["category"].str.lower().apply(lambda c:self.normalize(c))
        self.df_eng=self.df_eng.drop_duplicates()


    def normalize(self,doc):

        # Remove html tags
        if doc:
            soup = BeautifulSoup(doc, 'html.parser')
            text =soup.get_text()
            text = (re.sub('[-\W]+', ' ', text))
            text = (re.sub('(?<=\d) (?=\d)', '', text))
            text = (re.sub("([a-z]\d+)|(\d+)", '', text))


            return ''.join(
            c for c in unicodedata.normalize('NFD', text)
            if unicodedata.category(c) != 'Mn')
```

Listing 7.1: Function to normalize text and remove duplicate

7.2 Fetch normalized data

The normalized document is indexed in Elastic search analytic tool. In Elastic Search, the “search” function allows querying and retrieving data from indexed document. Code snippet in listing 7.2 fetches the indexed document.

```python
    def getNormal(self):
        self.df_en = pd.DataFrame(columns=['name','category'])
        resp=self.es.search("english-taxonomy-normal",{"_source":["name","category"],                          'size' : 5000,
        "query": {"match_all": {}}})
        for hit in resp['hits']['hits']:
                list_row_en = dict (name=None,category=None)
                list_row_en["name"]=hit['_source']['name']
                list_row_en["category"]=hit['_source']['category']
                new_row = pd.Series(list_row_en)
                self.df_en=pd.concat([self.df_en, new_row.to_frame().T], ignore_index=True)

        return self.df_en
```

Listing 7.2: Fetch normalized data from Elastic search

7.3 Train class initialization

As illustrated in code listing 7.3, the class Train is instantiated by passing the normalized form of data. This is the data that will be used for training the machine learning model.

```python
    df_en = df.getNormal()
    train = Train(df_en)
```

Listing 7.3: Object of the Train class

```python
from sklearn.feature_extraction.text import CountVectorizer
class Train():
    def __init__(self,df_en):

        self.vectorizer = CountVectorizer(1,1)
        self.df_en = df_en
        self.df_category = self.df_en.groupby("category")
        self.all_category = list(self.df_category.groups.keys())
        self.vectorizer.fit(doc=df_en["name"])
        self.inputSize=len(self.vectorizer.vocabulary_)
        self.n_categories=len(self.all_category)
        self.n_hidden = 128*3
        self.rnn= RNN(self.inputSize, self.n_hidden, self.n_categorie)
        self.learning_rate=0.005
        self.criterion = nn.NLLLoss()
        self.current_loss = 0
        self.all_losses = []
```

Listing 7.4: Train class constructor

The code snippet listing 7.4 shows the initialization of *Train* class attributes. Code listing 7.4 describes the initialization the required attributes qualifying the building blocks of a neural network, such as object of class *RNN* defining the input size, number of hidden layers, and size of the output. The loss function L, learning rate l, training dataset containing the true labels y_{true} are initialized.

1. self.vectorizer : Is an instance of **CountVectorizer** from scikit-learn (Buitinck et al., 2013). The details of Count vectorization is in section 5.5.2. The argument (1,1) indicates that the vectorizer will consider individual words as the vocabulary.
2. self.df_category: Stores the categorical classification of input data df_en
3. self.all_category : This attribute is created to store all unique categories found in the category column of the input df_en.
4. self.vectorizer.fit(doc=df_en["name"]): The fit method of the vectorizer is called on "name" column of input data "df_en" to build the vocabulary and transform text data in to numerical representation.
5. self.inputSize : It sets the input size of the neural network to the size of the vocabulary.
6. self.rnn: Recurrent Neural Network is initialized with updated input size, and number of hidden units and the number of unique categories as the output size.
7. self.criterion: An instance of Negative log likelihood loss (NLLLoss) pytorch function is stored. The function to minimize or maximize is called the objective function, or criterion (Goodfellow, Bengio and Courville, 2016, Section 4.3).
8. self.all_losses and self.current_loss: These attributes are used to track the loss values during the training process.

7.4 Fetch Training Dataset

The code listing 7.5 defines a method "randomTrainingExample". This method is responsible for generating training examples that will be used during the training process of the neural network. This function fetches the pair of vectorized input x^i that is the tensor representation of product name and true target label y^i_{true} which is the tensor representation of category.

```python
def randomTrainingExample(self):

    randcategory = random.choice(self.all_category)
    # get feature name from the category
    random_feature_indices = self.df_category.indices[randcategory]

    index = random_feature_indices[random.randint(0, len(random_feature_indices) - 1)]

    name =self.df_en.iloc[index]["name"]

    category_tensor = torch.tensor([self.all_category.index(randcategory)], dtype=torch.long)

    name_tensor = self.helper.nameToTensor(name)

    return randcategory, name, category_tensor, name_tensor
```

Listing 7.5: Train class constructor

- randcategory = random.choice(self.all_category) : This fetches random category from the list of categories extracted during the initialization of "Train" class
- random_feature_indices = self.df_category.indices[randcategory]: The indices of the randomly obtained category from step 1 is stored from the grouped dataframe "'df_category"'
- index = random_feature_indices[random.randint(0, len(random_feature_indices) - 1)] : Random index is obtained from the list of indices of data sample belonging to the selected category.
- name = self.df_en.iloc[index]["name"]:
  Name of the product is accessed from the "name" column of the data frame. It is accessed using randomly chosen index. It accesses the name (text data) associated with selected category.
- category_tensor = torch.tensor([self.all_category.index(randcategory)], dtype=torch.long :
  A tensor is created to represent the index of the randomly selected category. This tensor will be used during training as the target label for the corresponding input "name_tensor".
- name_tensor = self.helper.nameToTensor(name): The product name is converted into a tensor with a custom defined "helper" class. Code listing **??** is defined within the helper class.

7.5 Training Execution

In code listing 7.6 the Negative log likelihood loss is calculated between the output of the Recurrent Neural Network (RNN) y_{pred} which is the predicted-log-probabilities and the ground truth category tensor y_true.

```

    def train(self,category_tensor, name_tensor):

        hidden = self.rnn.initHidden()

        self.rnn.zero_grad()

        output, hidden = self.rnn(name_tensor, hidden)
        loss = self.criterion(output, category_tensor)
        loss.backward()

        # Add parameters' gradients to their values, multiplied by learning rate
        for p in self.rnn.parameters():
            p.data.add_(p.grad.data, alpha=-self.learning_rate)


        return output, loss.item()

```

Listing 7.6: Train function

7.5.1 Negative log likelihood loss

The negative log likelihood loss is useful to train a classification problem with "C" classes or target labels (Paszke et al., 2019). These labels are integers representing class indices. The Negative log likelihood loss is often used with $LogSoftMax$ activation function. In the final layer of the neural network, adding the $LogSoftMax$ functions return the predicted log-probabilities. The loss is calculated by comparing the predicted log probabilities with the true target labels y_{true}.

7.5.2 Back Propagation through time (BPTT)

Detailed explanation of back propagation is given in section 4.1.
In code listing 7.6 the loss.backward() method of the loss function performs back propagation. RNN is a computational graph with parameters holding memory of previous time step. A computational graph is a record of all data (tensors) and all executed operations in a directed acyclic graph (DAG). The leaves of a DAGs are the input tensor, roots are the output tensors. Unrolling the computational graph and then applying the back propagation algorithm is called Back Propagation through time (Murphy, 2022, section 15.2.5).

How the backward() method works?

1. During forward propagation of RNN module (refer section 4.4) with output of $LogSoftMax$ activation function, the log probabilities computation operations are recorded along with the computational graphs.

2. The **backward()** on a scalar tensor start traversing the computational graph in reverse order (from DAG root to leaves). For each step it applies chain rules of calculus flowing backward through the network in order to compute the gradients (Goodfellow, Bengio and Courville, 2016, section 6.5.2).

7.5.3 Model Parameter Optimization

At line number 13 of code listing 7.6, the loop updates the model parameter.

- The gradients are accumulated in the Pytorchs **torch.Tensor.grad** attributes of the input tensors. [1]
- These gradients can be accessed to update the model parameters through an optimization algorithm like Stochastic gradient descent (SGD) (refer chapter 4 for more details).
- Iteratively trained neural network's gradient-based optimizers drive the cost function($LogSoftmax$) to a very low value.

7.6 Experimentation: Variation in the training parameters.

Training without BPTT

These are the observations when **loss.backward()** function in code listing 7.6 is not used.

1. Pytorch's tensor.grad value is None. Hence, the manual optimization step does not update the model parameters.
2. Commenting the line number 9 and 10 from code listing 7.6, completes the training process. However, the model does not learn anything.
3. The figure 7.1 shows the loss across the training process. As the loss are not nearing to zero this shows that the model has failed in supervised learning of classification task.

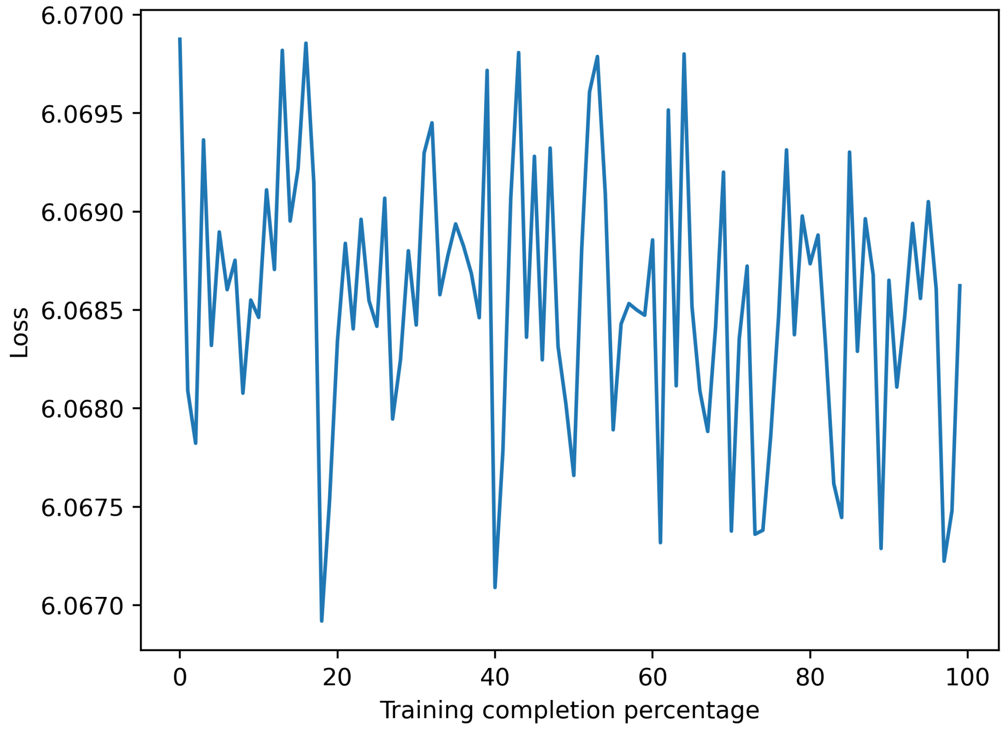


Figure 7.1: Loss without back propagation

[1]https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html

Training without updating the model parameters

In this experiment, the **loss.backward()** method calculated the gradients. However, the model parameters were not updated by commenting the line number 16 and 17 from code listing 7.6. Notice in the figure 7.1 and figure 7.2, the loss is not reaching zero towards the end of the training. Indicating that in both the cases the model has not learned the product name patterns for the classification task.

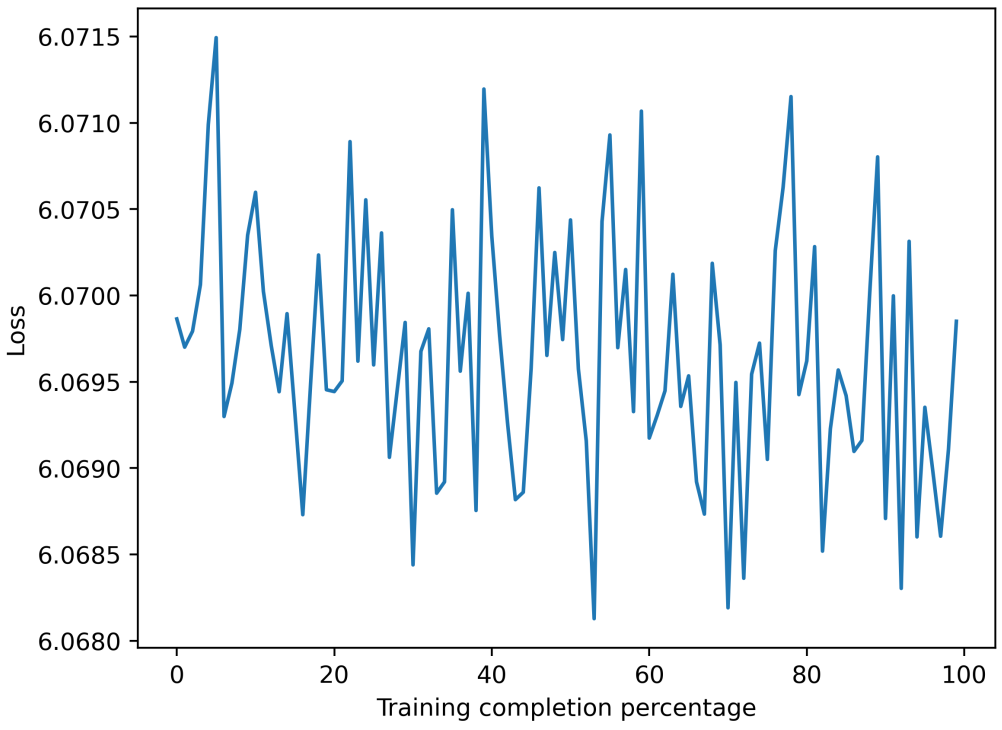


Figure 7.2: Loss without updating the model parameters

7.6.1 Training with manually updating the model parameters

The code listing 7.7 iterates through the parameters and updates their value based on the gradients computed during back propagation.

```
    # Add parameters' gradients to their values, multiplied by learning rate
        for p in self.rnn.parameters():
            p.data.add_(p.grad.data, alpha=-self.learning_rate)
```

Listing 7.7: Manual gradient updation

In code 7.7, the parameter “p.data” is a tensor. The method “add_()” multiplies the argument and then add the product value to the tensor data “p.data”. In this case, the negative value of “learning_rate” is multiplied with the parameter gradient data and then the value is added to the “p.data” attribute.

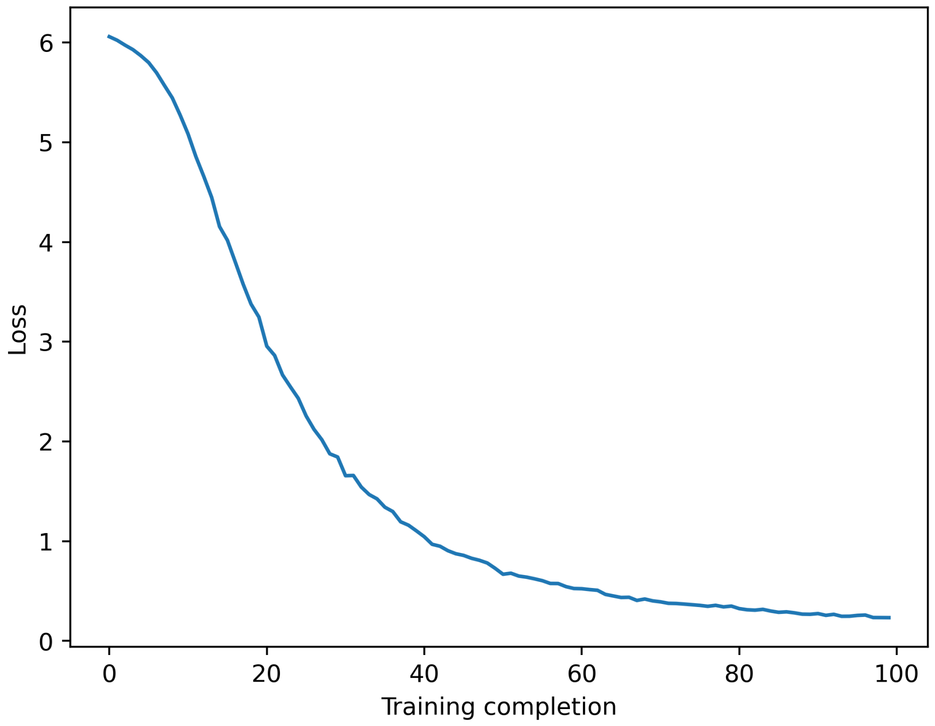


Figure 7.3: Loss with manually updating the model parameters with learning rate

Updating the hidden number of units

The model is described as a directed acyclic graph describing how functions are composed together. Consider having three functions $f(x) = f_3(f_2(f_1(x)))$ connected in a chain. f_1 is the first layer of the network, f_2 the second and so on. In the example of a classifier, $y = f_*(x)$ maps an input x to a category y. The final layer of the network is called the output layer. The training data specifies what the output layer must do at each point x, that is to produce a value close to y. As the training data does not show the desired output for other layers, they are called **hidden layers** (Goodfellow, Bengio and Courville, 2016, Chapter 3). (Zhang et al., 2021) describes the concept of **hidden state** which are inputs at step t can only be computed with previous input at step $t - 1$. Recurrent Neural Network (RNN) are neural network with hidden states.

Model with only one hidden state

In this experiment, only one hidden state has been assigned to the RNN network. As illustrated in figure 7.4, the loss reduces gradually. However, the model is yet under fit for the classification task. The loss not reaching zero towards the end of the training indicates that the model has not learned anything significant.

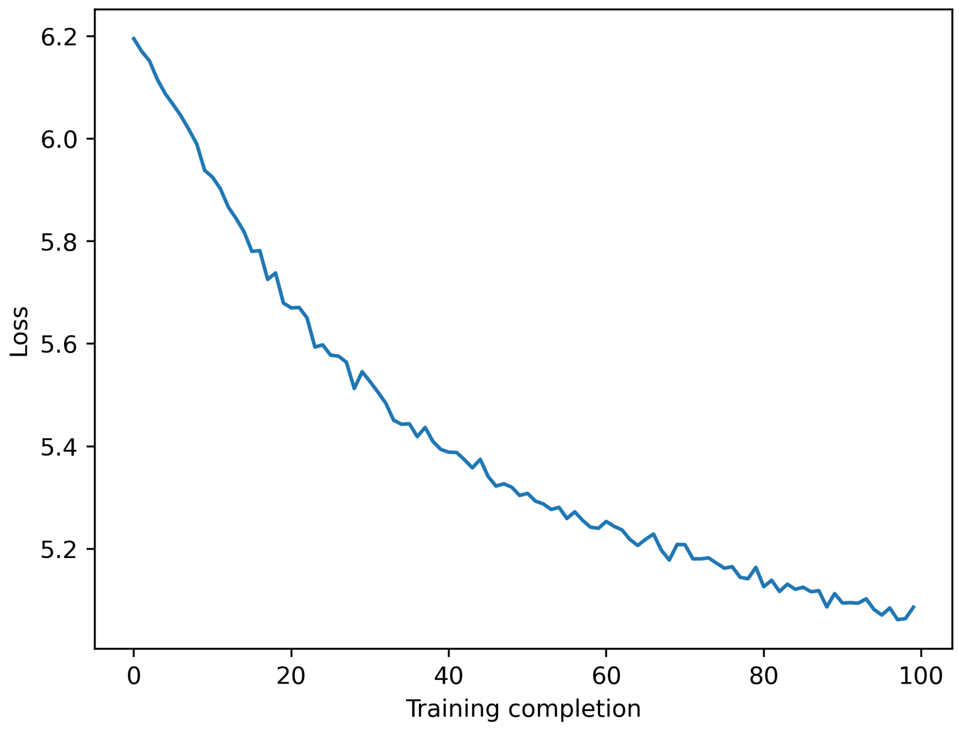


Figure 7.4: Model with only one hidden state

Model with only 10 hidden state

Increasing the number of hidden state to 10 units gave the desired result. As seen in figure 7.5, the loss has reached near zero by end of the training. This model is able to predict the category based on the pattern of name.

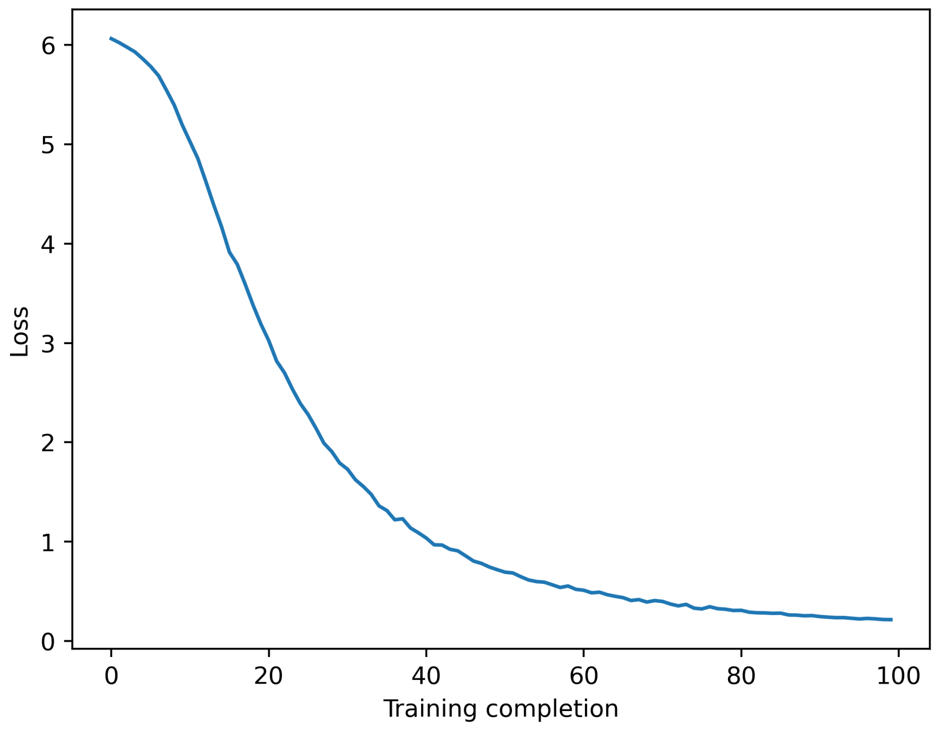


Figure 7.5: Model with only 10 hidden state

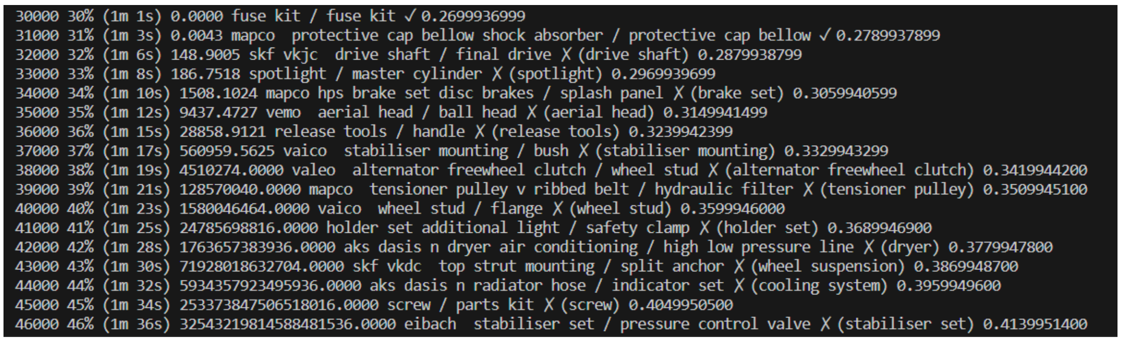
7.7 Gradient decent step adjustment

As stated in section 7.5.2, the method **loss.backward()** computes the gradients by applying chain rule of calculus for each function $f(x)$. The derivative of this function is denoted as $f'(x)$ or as $\frac{dy}{dx}$. $f(x)$ can be reduced by moving x in small with steps opposite sign of derivative is called gradient decent (Cauchy, 1847). (Goodfellow, Bengio and Courville, 2016, section 4.3) presents an equation 7.1.

$$x_i = x_i - \epsilon \frac{\partial}{\partial x_i} f(x) \tag{7.1}$$

- x_i: The ith parameter of the model (weight or coefficient).
- ϵ: The learning rate, a positive scalar which determines the size of the step.
- $f(x)$: The cost or loss function trying to minimize.
- $\frac{\partial}{\partial x_i} f(x)$: The partial derivative measures how f changes as only the variable x_i increases at point x.

In section 7.6.1, refer to the code listing 7.7, based on the equation 7.1 we conclude that the variable *p.grad.data* represents the partial derivative $\frac{\partial}{\partial x_i} f(x)$ and the negative value of variable *self.learninig_rate* is included to subtract product of variable *p.grad.data* and variable *self.learninig_rate* with variable *p.data*. The same result can be achived by using pytorch's *subtract_* method without negating the value of *self.learninig_rate*.



```
30000 30% (1m 1s) 0.0000 fuse kit / fuse kit ✓ 0.2699936999
31000 31% (1m 3s) 0.0043 mapco  protective cap bellow shock absorber / protective cap bellow ✓ 0.2789937899
32000 32% (1m 6s) 148.9005 skf vkjc  drive shaft / final drive ✗ (drive shaft) 0.2879938799
33000 33% (1m 8s) 186.7518 spotlight / master cylinder ✗ (spotlight) 0.2969939699
34000 34% (1m 10s) 1508.1024 mapco hps brake set disc brakes / splash panel ✗ (brake set) 0.3059940599
35000 35% (1m 12s) 9437.4727 vemo  aerial head / ball head ✗ (aerial head) 0.3149941499
36000 36% (1m 15s) 28858.9121 release tools / handle ✗ (release tools) 0.3239942399
37000 37% (1m 17s) 560959.5625 vaico  stabiliser mounting / bush ✗ (stabiliser mounting) 0.3329943299
38000 38% (1m 19s) 4510274.0000 valeo  alternator freewheel clutch / wheel stud ✗ (alternator freewheel clutch) 0.3419944200
39000 39% (1m 21s) 128570040.0000 mapco  tensioner pulley v ribbed belt / hydraulic filter ✗ (tensioner pulley) 0.3509945100
40000 40% (1m 23s) 1580046464.0000 vaico  wheel stud / flange ✗ (wheel stud) 0.3599946000
41000 41% (1m 25s) 24785698816.0000 holder set additional light / safety clamp ✗ (holder set) 0.3689946900
42000 42% (1m 28s) 1763657383936.0000 aks dasis n dryer air conditioning / high low pressure line ✗ (dryer) 0.3779947800
43000 43% (1m 30s) 71928018632704.0000 skf vkdc  top strut mounting / split anchor ✗ (wheel suspension) 0.3869948700
44000 44% (1m 32s) 5934357923495936.0000 aks dasis n radiator hose / indicator set ✗ (cooling system) 0.3959949600
45000 45% (1m 34s) 253373847506518016.0000 screw / parts kit ✗ (screw) 0.4049950500
46000 46% (1m 36s) 32543219814588481536.0000 eibach  stabiliser set / pressure control valve ✗ (stabiliser set) 0.4139951400
```

Figure 7.6: Result: Loss value returning nan

```python
    # Add parameters' gradients to their values, multiplied by learning rate
        for p in self.rnn.parameters():
            p.data.subtract_(p.grad.data, alpha=self.learning_rate)
```

Listing 7.8: Manual gradient updation with substract_

7.8 Cyclic learning rate (CLR)

Learning rate is an important parameter to tune the training of deep neural network. (Smith, 2015) describes a new method to set the learning rate called Cyclic learning rate (CLR). In this method the learning rate in the training cyclically vary between the reasonable boundaries of the learning rate instead of having a fixed learning rate.

The steps involved in training a model with CLR are as follows :

1. Define the Learning rate range or reasonable boundary value:
   In this step, by linearly increasing the learning rate during training can specify the learning rate range in which an optimal accuracy is obtained. The lower point of range can be referred as *base_lr* and higher point as *max_lr*.

2. Training the model by gradually decreasing the learning rate to *base_lr* and again increasing the learning rate to *max_lr* over an interval of a constant step size.

Experimentation and analysis to determine learning rate range

1. Experiment by setting *base_lr* and *max_lr* manually :
   In this experiment, the learning rate range is set manually to very low and high values. Numpy np.linspace() (Harris et al., 2020) returns the evenly spaced numbers between the *base_lr* and *max_lr* over a specified interval.

   The result of training process the model with $base\_lr = 1 \times 10^{-11}$ and $max\_lr = 0.9$.
   In figure 7.6 notice that at 32 % of training completion with learning rate ≈ 0.2 the loss started to explode. Hence, we conclude that the $max\_lr < 0.2$

2. In the next experiment, we set the $max\_lr = 0.2$.
   The classification model successfully completed the training and was able to predict the category.

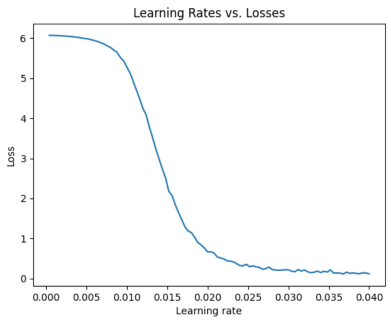
   As illustrated in figure 7.7b, in the graph plotted of Loss Vs Learning rate. The model started to learn at learning rate $\epsilon = 0.04$ onwards. We will further reduce the *max_lr*.

3. In the next experiment, we set the $max\_lr = 0.04$.
   Notice in figure 7.7a, there is a plateau until $\epsilon = 0.010$. After which there is a descent until $\epsilon = 0.015$

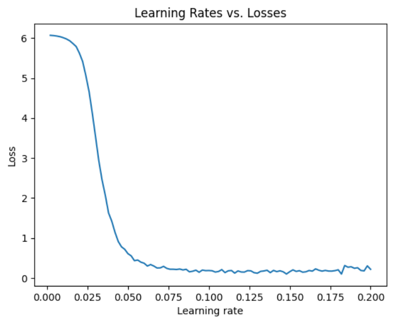
4. In the next experiment, we set the $max\_lr = 0.015$.
   Notice in figure 7.7c, the loss had not neared zero and the model fails to predict the classification.

5. From the earlier mentioned experimentation with max_lr, the loss in classification model is nearing zero by the end the training when $max\_lr > 0.015$ Figure 7.7d illustrates the loss reaching near zero when $max\_lr = 0.02$. The figure 7.7d, there is a plateau until the $\epsilon = 0.0075$. Indicating that the model did not learn until the learning rate $\epsilon = 0.0075$.

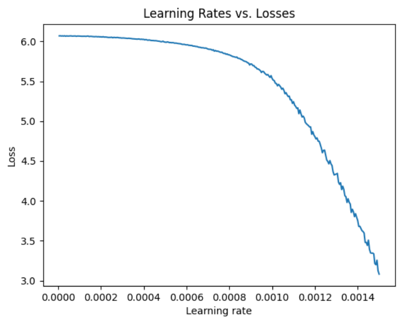
6. Updating the $base_lr = 0.0075$ and $max_lr = 0.02$, resulted a steady slop in the loss vs learning rate. Figure 7.7e illustrates the loss reaching near zero when $max\_lr = 0.02$ and $base_lr = 0.0075$. The steady decent indicates the learning rate range is optimal.



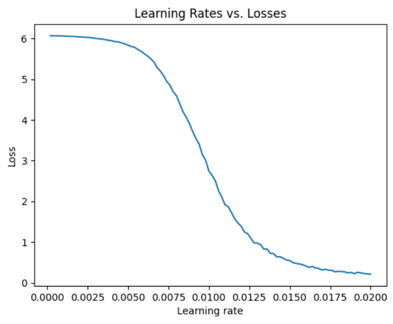
(a) Loss vs Learning rate; where $max\_lr = 0.04$



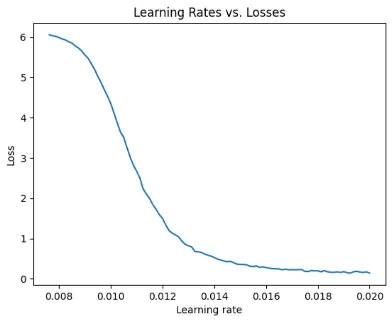
(b) Loss vs Learning rate; where $max\_lr = 0.2$



(c) Loss vs Learning rate; where $max\_lr = 0.015$



(d) Loss vs Learning rate; where $max\_lr = 0.02$



(e) Loss vs Learning rate; where $base\_lr = 0.0075$ and $max\_lr = 0.02$

7.9 Triangular learning rate

(Smith, 2015) introduced the concept of letting the learning rate vary within the range of minimum and maximum boundaries of learning rate known as learning rate range. In this method the learning rate linearly increases until step size t and then linearly decreases.

How triangular learning rate works?

1. **Initialize**: The step size t, minimum bound or base learning rate $base\_lr$ and maximum bound or the max learning rate max_lr are initialized. These hyperparameters are tuned based on the architecture of the model. In this experiment, the values set are $base\_lr = 0.0075; max\_lr = 0.02; t = 10000$.
2. **Up phase**: During training, the learning rate is linearly increased from $base\_lr$ to max_lr over a certain number of step size t.
3. **Down phase**: During training, the learning rate is linearly decreased from max_lr to $base\_lr$.

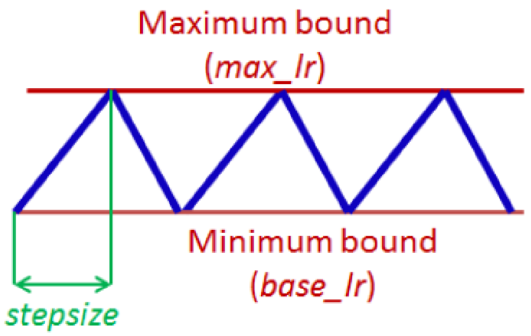


Figure 7.8: Triangular learning rate policy. (Smith, 2015)

Figure 7.8 is pictorial representation of triangular learning rate. The blue lines represent learning rate values changing between the bounds. The input parameter step-size is the number of iterations in half a cycle (Smith, 2015).

```python
def runBatchWithTLR(self):
# each unit is half cycle iteration
    batch=[10000,10000,10000,10000,10000,10000,10000,10000,10000,10000]
    start = time.time()
    no_batch=0
    no_iterations=0
    learning_rates =[]

    for n_iters in batch:

        no_batch =no_batch+1
        rem=no_batch % 2
        # linearly space the learning rate by number of iterations using np.
    linspace
        learn_rates=np.linspace(self.lr_low,self.lr_max,n_iters)
        # for even number of batch flip the array for down phase
        if rem==0:
            learn_rates=np.flip(learn_rates)

        # run the iterations along with the linearly spaced learning rate.
        for epoch,lr in zip(range(1, n_iters + 1),learn_rates):
            no_iterations= no_iterations+1

            self.learning_rate=lr
            category, name, category_tensor, name_tensor = self.data.
    randomTrainingExample()

```

```
            output, loss = self.train(category_tensor, name_tensor)
            self.current_loss += loss
            # Print ``iter`` number, loss, name and guess
            if epoch % 5000 == 0:
                guess, guess_i = self.data.categoryFromOutput(output)
                correct = 'yes' if guess == category else 'no (%s)' % category

                print('%d %d%% (%s) %.4f %s / %s %s %.10f' % (no_iterations,
                                            no_iterations / np.sum(batch) * 100,
                                            self.helper.timeSince(start),
                                            loss,
                                            name,
                                            guess,
                                            correct, self.learning_rate))


            if epoch % 5000 == 0:
                if (math.isnan(self.current_loss)!=True):
                    self.all_losses.append(self.current_loss / 5000)
                    learning_rates.append(self.learning_rate)
                    self.current_loss = 0
                else:
                    break
```

Listing 7.9: Triangular learning rate

Listing 7.9 is the code to train the model using triangular learning rate. Below is the explanation of the code.

1. **Variable initialization**:
    - **batch**:
      A list of elements representing number of iterations in a batch. Each batch represents half cycle or step-size.
    - **start = time.time()** :
      This stores current time at the beginning of the training process.
2. Batch iteration:
    - **rem = no_batch % 2**:
      Calculates reminder of no_batch. It determines whether to flip the learning rates array. If rem is 0 (indicating an even batch number), the array is fliped using numpy's (Harris et al., 2020) *np.flip* method.
    - learn_rates :
      This stores linearly spaced learning rates between *base_lr* and *max_lr* divided into batch size.

7.9.1 Experimentation and analysis

In this experiment, the model is trained with and without triangular learning rate method. The total number of iterations is 10000×10. The graph of Loss Vs Learning rate and time taken to complete the iterations illustrates the performance of the model. The graph plotted for each 2500 iterations provide insight on how the model is performing.

1. Triangular learning rate; where $base\_lr = 0.0075$ and $max\_lr = 0.02$:

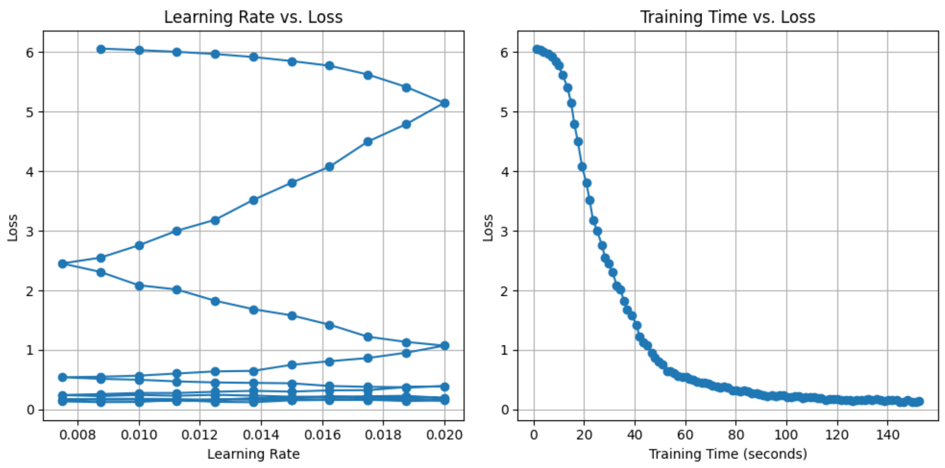


Figure 7.9: Loss with TLR, where $base\_lr = 0.0075$ and $max\_lr = 0.02$

7.10 Model Evaluation

The model is evaluated using the confusion matrix. Confusion matrix is square matrix of predicted label y_{pred} verses the true target label y_{true}. Figure 7.10 is the confusion matrix of the model trained only for 20 categories. The actual model is trained to predict around 300 categories. Author choose to attach the miniature version of the confusion matrix. Refer Appendix A code 2 for the python code to generate a confusion matrix.

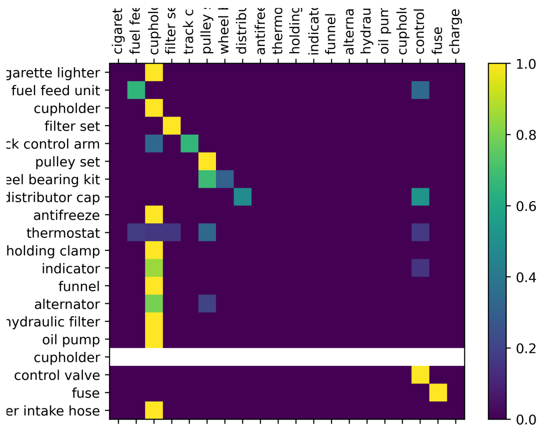


Figure 7.10: Confusion matrix: with little training.

Actual vs Predicted category

Author tested the model with test data of 10000 pairs of product name and category. Analyzing the falsely predicted result of 1455 records gained insight of incorrect categorization of existing products. Table 7.2 refers to a sample of record which the model falsely predicted the category. Which means $y_{pred} \neq y_{true}$. However, this records indicate a false negative record, which means the y_{pred} is incorrectly saved possibly due to human error.

Table 7.2: False negative - sample data

Name	y_{true}	y_{pred}
RPM Sensor, engine management	Sensor	RPM Sensor

7.11 Model Deployment

A **.pt** file represents the PyTorch model checkpoint file. These files store the parameter and architecture of trained PyTorch model. The PyTorch's torch.save and torch.load function uses the pythons Pickle (refer section 5.5.3) utility for serialization (Inkawhich, 2023).

As the Product Information Management systems get updated with the latest details of product information, the model requires retraining for accurate prediction. Refer Appendix A code 3 for Python code to load the trained PyTorch model.

7.12 Summary

This chapter is about the result of classification model. How the normalized text data is indexed and fetched from the elastic search is mentioned in this chapter. Initialization of the Train class and its input parameters detailed explanation is given in this chapter.

What is Back Propagation through time (BPTT)? How its relevant backwards methods works? are mentioned in the chapter.

Author experiments with the model parameters and evaluates the model based on the logarithmic loss value.

Few training experiments are:

- Training model without any learning parameters.
- Updating the number of hidden states in a neural network.
- Determining the learning rate range.
- Training with Triangular learning rate.

This chapter also describes the gradient decent (Cauchy, 1847) step adjustment and compares the equation with the Pytorchs's tensor gradient method of adjustment with learning rate.

Chapter 8

Conclusion and Outlook

8.1 Answer to Research Questions

RQ1: What are the use cases in which natural language processing and machine learning model can be utilized in an E-Commerce industry?

Machine learning model can generate well-defined product taxonomy. Author introduces a methodology for such a model in section 6.2. A well-defined product taxonomy is the foundation for many use cases like the ones listed below:

- Product recommendation system.
  The machine learning model can classify and recommend products from the wide range of products to the customer based on the customer's transaction history. The transaction history containing product relevance feedback such as click "clicks", "cart-adds", "orders", "revenue" (Karmaker Santu, Sondhi and Zhai, 2017) are usable if the taxonomy is well-defined.
- Search bar autocompletion.
  A lot of research has been conducted in the field of user experience in terms of product search. A facet in user interface is an extended drop down menu with filtered result set. (Tagliabue, Yu and Beaulieu, 2020) proposes a real time personalized catalog generation by combining customer search logs and in-session data.
- Customer conversational shopping.
  (Messina, 2015) states that gradually traditional commerce will shift towards conversational commerce. An experience of shopping on the go or hands-free assistance could be trending. Major Ecommerce industries offering voice assistance devices includes feature of buying products on voice command.
- Building a knowledge graph based on the textual details of a product. Refer section 3
- Virtual customer support chatbot may leverage the knowledge graph to answer queries raised by customers.

RQ2: For a product classification task, how to define product features as an input for machine learning model?

Based on the level of product taxonomy the number of features may vary. Table 5.1 lists some features are listed simply by prior knowledge of the product domain. However, section 5.1.1 describes various methods to reduce dimensionality.

RQ3: How does a machine learn pattern in product features for classification?

SRQ1: What are the mathematical equations behind the learning process?

Author has dedicated Chapter 4 for step by step illustration of learning process. Author begins the chapter with basic mathematical representation of a neural network, followed by equations for feedforward, back-propagation. Further, author describes the probability distribution and its relation with the project in this paper. What is the role of partial derivatives to reduce the loss function is described.

SRQ2: What are the mathematical reasons for applying certain pytorch functions during the training process?

Section 4.3.1 describes mathematical reasons behind using Pytorch's functions such as LogSoftMax. Chapter 4 is detailed with the explanation of applying log to a value.

RQ4: What is the algorithm to train the machine learning model to predict the product taxonomy?

Author modified the algorithm of (Robertson, 2023) classifying names with character-level RNN. Instead of series of tensor representation of characters, author use tensor representation of features. The RNN model learns the pattern and predicts the product taxonomy.

8.2 Future scope

In this paper, author has researched on the machine learning task to text based classification. Research has been conducted on use cases of text based classification in E-Commerce. Author limited the research surrounding basic understanding of neural network, its mathematical equations, creating a model which predicts product taxonomy. This paper is also a guide for individual who seek basic understanding of machine learning as well as overview of product taxonomy and its importance in E-commerce.

There is a scope of research on image classification and its use cases in the E-commerce industry. Images are vital component of an E-commerce website. There is saying "A picture is worth a thousand words". The product image itself has so much information to be tapped.

Few of them are listed below:

- Image captioning.

  Generating text from the image is a useful process in E-commerce. For example, product image also describes its features such as color. Such information can be added to the caption of the image. (Murphy, 2022, Section 15.4.7) describes an example of image captioning using the sequential data processing network with **attention algorithm**.

- Attention based image classification.

  (Vaswani et al., 2017) proposes a network architecture named Transformers which uses attention in the encoder and decoder in the machine translation tasks. Using the transformer model to process the sequence of pixels in a product image to identify the clustered components or to identify which other product could be a part of the product. For example, image from Appendix B figure 1 is product image, using attention based image classification, identifying the products which fit in and around the processed product image could enable E-Commerce industries cluster the products.

- Image based categorization.

  A Machine learning model can generate product taxonomy based on the image of the product. Clustering the products based on the image could also enable to define a better product taxonomy.

- Image to image translation.

  Using generative adversarial networks (GANs) (Goodfellow, 2016), a sketch can be converted to photorealistic image.

List of Figures

List of Tables

Listings

References

Ali Cevahir and Koji Murakami 2016.
Large-scale Multi-class and Hierarchical Product Categorization for an E-commerce Giant.
Available at: `https://aclanthology.org/C16-1051.pdf`
Accessed on: 10/05/2023.

Bird, Steven, Ewan Klein and Edward Loper 2009.
Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit.
Beijing: O'Reilly.
ISBN: 978-0-596-51649-9.
DOI: `http://my.safaribooksonline.com/9780596516499`.
Available at: `http://www.nltk.org/book`
Accessed on: 10/04/2023.

Bishop, Christopher m. 1995.
Book-Bishop-Neural Networks for Pattern Recognition.
Accessed on 01-05-2023.
Clarendon Press.

Buitinck, Lars et al. 2013.
'API design for machine learning software: experiences from the scikit-learn project'.
In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*,
Pp. 108–122.

Burges, Chris J.C. June 2010.
From RankNet to LambdaRank to LambdaMART: An Overview.
Tech. rep. MSR-TR-2010-82.
Available at: `https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/`
Accessed on: 12/05/2023.

Cauchy, Augustin-Louis 1847.
'Gradient decent'.
In: Information from Wikipedia, original paper not available.

Davies, J., Rudi Studer and Paul Warren, eds. 2008, cop. 2006.
Semantic web technologies: Trends and research in ontology-based systems.
Reprinted.
Chichester: J. Wiley & Sons.
ISBN: 978-0-470-02596-3.

Du Xinya, Junru Shao and Claire Cardie 2017.
Learning to Ask: Neural Question Generation for Reading Comprehension.
Available at: `https://arxiv.org/pdf/1705.00106.pdf`
Accessed on: 28/05/2023.

Gevin, Gurney 1997.
An introduction to neural networks.
London: UCL Press.
ISBN: 9781857286731.
Available at: `http://www.macs.hw.ac.uk/~yjc32/project/ref-NN/Gurney_et_al.pdf`
Accessed on: 10/08/2023.

Goodfellow, Ian 2016.
NIPS 2016 Tutorial: Generative Adversarial Networks.

Available at: https://arxiv.org/pdf/1701.00160.pdf
Accessed on: 03/09/2023.

Goodfellow, Ian, Yoshua Bengio and Aaron Courville 2016.
Deep Learning.
MIT Press.
Available at: http://www.deeplearningbook.org
Accessed on: 15/05/2023.

Grave, Edouard et al. 2016.
Efficient softmax approximation for GPUs.
Available at: https://arxiv.org/pdf/1609.04309.pdf
Accessed on: 02/08/2023.

Gupta, Vivek et al. 2016.
Product Classification in E-Commerce using Distributional Semantics.
Available at: https://arxiv.org/pdf/1606.06083.pdf
Accessed on: 03/09/2023.

Hagberg, Aric, Pieter Swart and Daniel S Chult 2008.
Exploring network structure, dynamics, and function using NetworkX.
Tech. rep.
Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Hagberg, Aric A., Daniel A. Schult and Pieter J. Swart 2008.
'Exploring Network Structure, Dynamics, and Function using NetworkX'.
In: *Proceedings of the 7th Python in Science Conference.*
Ed. by Gaël Varoquaux, Travis Vaught and Jarrod Millman.
Pasadena, CA USA,
Pp. 11–15.

Harris, Charles R. et al. Sept. 2020.
'Array programming with NumPy'.
In: *Nature* 585.7825, pp. 357–362.
DOI: 10.1038/s41586-020-2649-2.

Hernandez, Lupe and Ahmad Nazeri Sam Randall 2020.
'Question Generator Natural Language Processing'.
In:
Available at: http://cs230.stanford.edu/projects_fall_2020/reports/55771015.pdf
Accessed on: 29/05/2023.

Honnibal, Matthew and Ines Montani 2017.
'spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing'.
spaCy excels at large-scale information extraction tasks.

Hoogeveen, Martijn 2019.
What is Product Information Management (PIM) in a Multimedia World?
Available at: https://iceclog.com/what-is-product-information-management-in-a-multimedia-world/
Accessed on: 09/08/2023.

Inkawhich, Matthew 2023.
SAVING AND LOADING MODELS.
Available at: https://pytorch.org/tutorials/beginner/saving_loading_models.html
Accessed on: 09/08/2023.

Jessica Howard 2023.
Why is Product Taxonomy So Important for eCommerce.
Available at: https://datacluster.com/product-taxonomy-ecommerce
Accessed on: 01/09/2023.

Karmaker Santu, Shubhra Kanti, Parikshit Sondhi and ChengXiang Zhai 2017.
'On Application of Learning to Rank for E-Commerce Search'.
In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval.*
Ed. by Noriko Kando.
New York, NY: ACM,
Pp. 475–484.

ISBN: 9781450350228.
DOI: 10.1145/3077136.3080838.
Available at: https://arxiv.org/pdf/1903.04263.pdf
Accessed on: 12/09/2023.

Kutyniok, Gitta 2023.
The Mathematics of Artificial Intelligence.
Available at: https://arxiv.org/pdf/2203.08890.pdf
Accessed on: 03/08/2023.

Lederer, Johannes 2023.
Activation Functions in Artificial Neural Networks: A Systematic Overview.
Available at: https://arxiv.org/pdf/2101.09957.pdf
Accessed on: 05/06/2023.

Lisa Ehrlinger, Wolfram Wöß 2016.
'Towards a Definition of Knowledge Graphs'.

Liu, Tieyuan et al. 2023.
'Iterative heterogeneous graph learning for knowledge graph-based recommendation'.
In: *Scientific reports* 13.1, p. 6987.
DOI: 10.1038/s41598-023-33984-5.

Ma, Bonnie 2019.
Feature Transformation for Multiple Linear Regression in Python.
Available at: https://towardsdatascience.com/feature-transformation-for-multiple-linear-regression-in-python-8648ddf070b8
Accessed on: 10/05/2023.

Mayo, Matthew 2017.
Building a Wikipedia Text Corpus for Natural Language Processing.
[Online; accessed 01-May-2023].

McKinney, Wes 2010.
'Data Structures for Statistical Computing in Python'.
In: *Proceedings of the 9th Python in Science Conference.*
Ed. by Stéfan van der Walt and Jarrod Millman,
Pp. 56–61.
DOI: 10.25080/Majora-92bf1922-00a.

McKinney, Wes et al. 2010.
'Data structures for statistical computing in python'.
In: *Proceedings of the 9th Python in Science Conference.*
Vol. 445.
Austin, TX,
Pp. 51–56.

Messina, Chris 2015.
Conversational commerce, Messaging apps bring the point of sale to you.
Available at: https://medium.com/chris-messina/conversational-commerce-92e0bccfc3ff
Accessed on: 10/05/2023.

Murphy, Kevin P. 2022.
Probabilistic Machine Learning: An introduction.
MIT Press.
Available at: probml.ai.

Paszke, Adam et al. 2019.
PyTorch: An Imperative Style, High-Performance Deep Learning Library.

Pedregosa, F. et al. 2011.
'Scikit-learn: Machine Learning in Python'.
In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Pranav Rajpurkar et al. 2016.
'SQuAD: 100,000+ Questions for Machine Comprehension of Text'.
In.

Řehůřek, Radim and Petr Sojka May 2010.
'Software Framework for Topic Modelling with Large Corpora'. English.
In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks.*
Valletta, Malta: ELRA,

Pp. 45–50.
Retromotion.com 2023.
Spare parts shop.
Available at: `https://retromotion.com/p/vaico-v70-0613`
Accessed on: 05/08/2023.
Robertson, Sean 2023.
Classifying names with a character level RNN.
Available at: `https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html`
Accessed on: 01/05/2023.
Rumelhart, David E., Geoffrey E. Hinton and Ronald J. Williams 1986.
‘Learning representations by back-propagating errors’.
In: *Nature* 323.6088, pp. 533–536.
ISSN: 1476-4687.
DOI: `10.1038/323533a0`.
Smith, Leslie N. 2015.
Cyclical Learning Rates for Training Neural Networks.
Sukhbaatar, Sainbayar et al. 31/03/2015.
‘End-To-End Memory Networks’.
PhD thesis.
Tagliabue, Jacopo, Bingqing Yu and Marie Beaulieu 2020.
How to Grow a (Product) Tree: Personalized Category Suggestions for eCommerce Type-Ahead.
Torsten Zesch Iryna Gurevych, Max Mühlhäuser 2008.
‘Analyzing and Accessing Wikipedia as a Lexical Semantic Resource’.
In.
Vaswani, Ashish et al. 2017.
Attention Is All You Need.
Available at: `https://arxiv.org/pdf/1706.03762.pdf`
Accessed on: 05/09/2023.
Wikipedia 2023.
Wikipedia special exports.

Accessed on: 01/05/2023.
Wikipedia API documentation 2023.
Available at: `https://wikipedia-api.readthedocs.io/en/latest/wikipediaapi/api.html`
Accessed on: 02/06/2023.
Zhang, Aston et al. 2021.
‘Dive into Deep Learning’.
In: *arXiv preprint arXiv:2106.11342.*
Zhou, Victor 2019.
Machine Learning for Beginners: An Introduction to Neural Networks.
Available at: `https://victorzhou.com/blog/intro-to-neural-networks/`
Accessed on: 05/08/2023.

Appendices

.1 List of Abbreviations

KG	Knowledge graph	9
KB	Knowledge base	9
QE	Questioning Engine	9
NLP	Natural Language Processing	1
QG	Question generation	9
adp	adposition	
pobj	prepositional object	
dobj	direct object	
nsubj	nominal subject	
POS	part of speech	10
pron	pronoun	
SRO	subject,relation and object	10
KNN	*k* -Nearest Neighbors	i
RNN	Recurrent Neural Network	1
ALL	Adaptive LogSoftmax with Loss	35
NLLLoss	Negative log likelihood loss	38

BPTT	Back Propagation through time	ii
DAG	directed acyclic graph	4
SGD	Stochastic gradient descent	14
CLR	Cyclic learning rate	ii
LTR	Learn to Rank	8
IR	Information Retrieval	8
AI	Artificial Intelligence	8
PMF	Probability Mass Function	14
MSE	Mean Squared Error	13
GAN	generative adversarial network	53
PIM	Product Information Management	1
SKU	stock-keeping unit	5
SEO	Search Engine Optimization	1

.2 Appendix A

For this project, python client elasticsearch 6.8.2 is installed as the client needs to be compatible with Elastic search version being used. The official Python client provides mapping with Elasticsearch REST APIs.

```
    resp=self.es.search("english-name-category",{"_source":["id","name","category"],
    'from':_from,
    'size' :_size ,
    "query": {"match_all": {}}})

```

Listing 1: Elastic search

Confusion matrix

```
def confusionMatix(self,df_en):
# Keep track of correct guesses in a confusion matrix
data = Data(df_en)
n_categories = len(data.all_category)

batch = random.choices(data.all_category,k=20)
```

```

confusion = torch.zeros(n_categories, n_categories)
n_confusion = 10000

# Go through a bunch of examples and record which are correctly guessed
for i in range(n_confusion):
    category, name, category_tensor, name_tensor = data.randomTrainingExample()
    if category in batch:
        output = self.evaluate(name_tensor)
        guess, guess_i = data.categoryFromOutput(output)
        category_i =data.all_category.index(category)
        confusion[category_i][guess_i] += 1

# Normalize by dividing every row by its sum
for i in range(20):
    confusion[i] = confusion[i] / confusion[i].sum()


# Set up plot
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(confusion.numpy())
fig.colorbar(cax)

# Set up axes
ax.set_xticklabels([''] + data.all_category, rotation=45)
ax.set_yticklabels([''] + data.all_category)

# Force label at every tick
ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

# sphinx_gallery_thumbnail_number = 2
plt.show()
plt.savefig('confusion.png', dpi=400)
```

Listing 2: Confusion matrix

Load PyTorch model

```
class Predict():
    def __init__(self):
         self.rnn = torch.load('ngram-rnn-classification.pt')

```

Listing 3: Load PyTorch model

.3 Appendix B

Figure 1: Hydraulic filter automatic transmission VAICO V70-0613 for Lexus RX (Retromotion.com, 2023)