Realistic Advanced Analytics Pipelines for End-to-End Machine Learning Benchmarking

Robertus Vincent
Student Number: 359118
Study Course : Master Program Computer Science

Master Thesis
Faculty IV
Electrical Engineering and Computer Science
Technical University Berlin

Institute:
Database Systems and Information Management Group

Supervisors:
Prof. Dr. rer. nat. Volker Markl
Prof. Dr. Tilmann Rabl

Advisors:
Behrouz Derakhshan, M.Sc.

10. August 2021
Berlin

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den 10. August 2021

[illegible]

Robertus Vincent

Abstract

Advancements in machine learning (ML) technology and services are driving widespread adoption of ML approaches. However, the diversity of ML tools and system makes it difficult for users to choose the best tool and system for their use-cases. While recent efforts to benchmark ML methods exist, none of them address end-to-end ML pipelines with realistic data. Publicly available datasets used in the most benchmark approaches are pre-cleaned and do not reflect industrial data. In a complete ML lifecycle, often times data cleaning and feature extraction take a considerable amount of processing time before running the model. The challenges involved in successfully applying ML methods in diverse enterprise settings extend far beyond efficient model training.

This thesis discusses ADABench which addresses these challenges. It is an advanced data analytics benchmarking system that covers the complete end-to-end ML pipeline for diverse, industry-relevant application domains rather than evaluating only training performance. ADABench has proposed 16 use-cases, out of which six are already presented. This thesis extends its reference use-cases that cover a wide range of industry-relevant ML applications, i.e. customer review analysis, ticket classification and inventory demand forecasting.

We present additional use-cases including corresponding metrics and run rules, and evaluate them at different scales in terms of hardware, software, and problem size. We analyze existing application in the industry in these domains and based on publicly available solutions we design a representative data schema in the industry that can be generated at scale along with with the state-of-the-art ML pipelines. The results from our evaluation demonstrate the different trade-off in terms of performance based on data size, choice of environment and complexity.

Zusammenfassung

Fortschritte in der Technologie und den Diensten des maschinellen Lernens (ML) führen zu einer weit verbreiteten Einführung von ML-Konzepten. Die Vielfalt der ML-Tools und -Systeme macht es den Nutzern jedoch schwer, das beste Tool und System für ihre Anwendungsfälle auszuwählen. Zwar gibt es in letzter Zeit Bemühungen um ein Benchmarking von ML-Methoden, aber keine von ihnen befasst sich mit End-to-End-ML-Pipelines mit realistischen Daten. Öffentlich verfügbare Datensätze, die in den meisten Benchmark-Ansätzen verwendet werden, sind vorbereinigt und spiegeln keine industriellen Daten wider. In einem vollständigen ML-Lebenszyklus nehmen die Datenbereinigung und die Merkmalsextraktion oft einen beträchtlichen Teil der Verarbeitungszeit in Anspruch, bevor das Modell ausgeführt wird. Die Herausforderungen bei der erfolgreichen Anwendung von ML-Methoden in verschiedenen Unternehmensumgebungen gehen weit über effizientes Modelltraining hinaus.

Diese Arbeit befasst sich mit ADABench, das diese Herausforderungen angeht. Es handelt sich dabei um ein fortschrittliches Datenanalyse Benchmarking System, das die komplette End-to-End ML-Pipeline für verschiedene, branchenrelevante Anwendungsbereiche abdeckt und nicht nur die Trainingsleistung bewertet. ADABench hat 16 Anwendungsfälle vorgeschlagen, von denen sechs bereits vorgestellt wurden. In dieser Arbeit werden die Referenz-Anwendungsfälle erweitert, die ein breites Spektrum an branchenrelevanten ML-Anwendungen abdecken, und zwar die Analyse von Kundenrezensionen, die Klassifizierung von Tickets und die Vorhersage der Nachfrage nach Waren.

Wir stellen die zusätzlichen Anwendungsfälle einschließlich der entsprechenden Metriken und Ausführungsregeln vor und evaluieren sie auf verschiedenen Skalen in Bezug auf Hardware, Software und Problemgröße. Wir analysieren bestehende Anwendungen in der Industrie in diesen Bereichen und entwerfen auf der Grundlage öffentlich verfügbarer Lösungen ein repräsentatives Datenschema in der Industrie, das zusammen mit den modernsten ML-Pipelines in großem Maßstab generiert werden kann. Die Ergebnisse unserer Evaluierung zeigen die unterschiedlichen Kompromisse in Bezug auf die Leistung in Abhängigkeit von der Datengröße, der Wahl der Umgebung und der Komplexität.

Acknowledgements

My sincere gratitude goes to my thesis advisors Prof. Dr. Tilmann Rabl and Behrouz Derakhshan for the continuous supports and feedbacks of my master thesis, and to Christoph Brücke for continuously helping me with working with ADABench. Their guidance helped me in all the time of my research and writing of this thesis.

I would also like to thank my family, my friends, and all my colleagues at work for their continuous encouragements.

Special thanks go to Annastasya and Ayaz, for the constant supports and proofreading of this work. I am gratefully indebted for their valuable supports. Thank you!

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

Advancements in machine learning (ML) technology and services have made enterprises apply advanced data analytics and ML tools and systems into their workflow and infrastructures. However, the diversity of ML tools and systems makes it difficult for users to choose the best tool and system for their use-cases. There are many aspects to consider when choosing them and these often lead to one-dimensional methods of comparison, such as the best model output accuracy and speed of the model training in isolation. While those metrics are important, there are many other aspects to consider while executing ML pipelines. Often times data cleaning and feature extraction take a considerable amount of processing time before running the model. Therefore, it is important to address the performance on the whole ML pipeline, which includes data integration, data pre-processing, model training, and model serving, while optimizing a ML pipeline.

Benchmarking is a good medium to determine the quality of a system and its advantages and disadvantages in compare to the others. While it can be done with any workload, it is important to have diverse and relevant workloads and data to get meaningful results. Although there are previous approaches in benchmarking ML system such as DeepBench [68], DAWNBench [27], and PMLB [73]. They mainly focus on benchmarking the models in question using toy data. There are not many that study end-to-end ML pipeline performance using real-world datasets.

ADABench [79] is a ML benchmarking tool that address this problem. It covers complex end-to-end ML pipelines, from data preparation all the way to inference. It covers various dimensions of real business analytics problems and includes different scale ranges, which typically found in industry. It follows an industry standard as specified from the Transaction Processing Performance Council[1] (TPC) and the Standard Performance Evaluation Corporation[2] (SPEC).

In ADABench paper, 16 use-cases have been proposed, which are derived from retail business applications and analytic tasks common in online bussinesses according to their market potential identified by McKinsey [25]. Six use-cases are already presented in the paper, namely trip-type classification, predictive maintenance, sales prediction, customer segmentation, spam detection, and product recommendation. They are chosen for their high impact in businesses related to customer service management and predictive maintenance.

This thesis will extend additional industry use-cases for ADABench, specifically customer review analysis, ticket classification, and inventory demand forecasting. We analyze existing application in the industry in these domains and based on publicly available solutions we design a representative data schema in the industry that can be generated at scale along with with the state-of-the-art ML pipelines. To name some examples: MonkeyLearn[3] creates a plugin for support desks to automate ticket classification with machine learning, and automatic data collection and reporting. Facebook developes a forecasting tool that is optimized for business forecast tasks that are normally encountered at Facebook [5].

[1]www.tpc.org
[2]www.spec.org
[3]https://monkeylearn.com/

We aim for diversity in the benchmark along several dimensions: the structure of the data (structured and unstructured), type of data (numerical and textual), use-cases (forecasting and classification), and ML methods and algorithms (bayesian classification, deep neural networks, and time series forecasting). It will also include the data generation for all use-cases. The data generation allows the ADABench to be tested in customizable data schema and several problem sizes. ADABench will be tested in several dimension of scalability, namely hardware, software (Python-environment vs. Spark-environment), and problem size (MBs vs. GBs).

In summary, the main contributions of this thesis are the following

1. It investigates three additional industry-relevant use-cases, including their corresponding performance metrics.
2. Design and implementation of the state-of-the-art ML pipelines for the specified use-cases.
3. Providing data generation schema for specified use-cases that reflects real business analytics problem.

The remainder of this thesis is structured as follows. Section 2 introduces ADABench and its data generation component that serve as the basis of this thesis. Section 3 provides necessary theoretical background to discuss the implementation methods for all use-cases. Section 4 explains the relevance of the use-cases in the industry along with their challenges in the ML field. Section 5 elaborates the technical details pertaining to the implementation of the data generation and the ML pipeline. The evaluations of the experiment are then described in Section 6. The researches related to the ML benchmark are highlighted in Section 7. Section 8 concludes this thesis by summarizing the contributions and results.

2 ADABench

Benchmark is the practice of assessing performance of systems by comparing them based on some metrics. The metric is a specific indicator, typically measures time, cost, and/or quality, depending on the goal of the systems. It uses a pre-determined workload to assure fairness across the systems. The workload is normally designed to mimic a particular type of workload that the system will run in practice. Evaluating the systems this way allows identifying both strengths and weaknesses of a system in compare with others.

According to [29], there are seven vital characteristics for benchmarks.

1. Relevance
   Benchmarks should measure important features. It should not leave relevant features in the calculation. For example, not considering a processing time and only benchmarking the cost.
2. Representativeness
   Benchmarks performance metrics should be broadly accepted by industry and academia. The performance metrics should be plausible and relevant to the specified problem.
3. Equity
   All systems should be fairly compared. This means that all benchmark entities should be compared under the exact same conditions.
4. Repeatability
   Benchmarks results should be verifiable. Benchmarks should produce a same result upon re-running them.
5. Cost-effectiveness
   Benchmarks tests should be economical. It should not use too much resources for the actual problem. Setting a budget before-hand can help to prevent over-engineering benchmarks.
6. Scalability
   Benchmarks test should work across a range of resources from low to high. Having benchmark tests that work across a range of resources increases the credibility of the benchmark due to them being not dependent to the resources.
7. Transparency
   Benchmarks performance metrics should be easily understandable. Having an easily understandable metric avoids the possibility of errors in the calculation and it also allows the result to be easily verified.

The term benchmark is used in machine learning (ML) to refer to the evaluation and comparison of ML methods regarding their ability to learn patterns from existing data to accurately predict the new data. Comparisons could be made over a range of evaluation ML specific metrics, e.g. prediction accuracy, time-to-accuracy [27], throughput, and model interpretability. Benchmark in ML can demonstrate the performance of a new methodological approach or to help deciding an appropriate ML method for a given problem.

ML Benchmark is typically done with datasets that have one of three forms [73]. The first is accessible, well-studied *real-world* data, taken from different real-world problem domains of interest. The second is *simulated* data, which has been artificially generated to reflect real-world data with known, underlying patterns. A good example of this type of dataset is the GAMETES [91] genetic-data simulation softwares. It generates complex single nucleotide polymorphism (SNP) data for simulation studies. The third form is *toy* data, which has also been artificially generated with known, underlying patterns but without representing real-world data.

There are currently a number of publications that describe their approach on evaluating an ML system. However, they only focus on assessing the performance of the model itself. While it is important, it does not address the fundamental challenges of ML in practice. As seen in Figure 2.1, an ML process contains more than just training and serving the model. It starts by loading and integrating the data from one or multiple sources into the pipeline. Followed by data cleaning and feature extraction, which are very important but tedious tasks in an ML pipeline. It is said that roughly 80% of the time in an ML project spent on data cleaning and feature extraction [34]. Poor quality of the data can not only hinder the training phase, but also impact the result negatively. Only after these steps, the model can be properly trained.

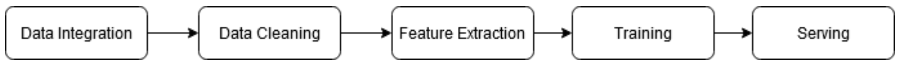


Figure 2.1: Simplified schematic of a production Machine Learning Pipeline

2.1 ADABench Approach

ADABench [79] proposed another approach for an ML benchmark. It is an end-to-end ML benchmark, which covers the complete ML lifecycle, from data preparation all the way to inference. The benchmark is built to cover real-world analytics requirements which are typically found in the industry, such as handling input data in different scale range, input data that vary in size across the pipeline, and sparse numerical representations of the data. ADABench follows an industry standard as specified from the Transaction Processing Performance Council [4] (TPC) and the Standard Performance Evaluation Corporation [5] (SPEC).

ADABench planned 16 use-cases that cover a wide range of industry-relevant ML applications, out of which six use-cases are already presented. McKinsey [25] shares an insight of commonly found retail business applications and analysis tasks in online businesses. These serve as the basis of ADABench use-cases. ADABench paper presented use-cases with high focus on customer service management and predictive maintenance, they are: trip-type classification, predictive maintenance, sales prediction, customer segmentation, spam detection, and product recommendation.

Although the end goal in ADABench is to measure the whole ML pipeline performance, it is still important that every use-case achieve a meaningful result. Each use-case has its individual performance metrics, which are then aggregated by means of geometric mean into one global metric summarizing the entire benchmark [79]. Geometric mean is used to weight the diverse use-cases equally in the final metric

[4] `www.tpc.org`
[5] `www.spec.org`

and encourage optimization on all stages. ADABench uses *throughput* in *samples per second* as the main performance measure. Most use-cases define a training and a serving stage, which are measured separately (TP_t and TP_s) and summarize in the total throughput for the use-case TP_i with i being the use-case number (see Equation 1). The total throughput is shown in Equation 2.

$$TP_i = \sqrt{TP_{ti} * TP_{si}} \tag{1}$$

$$TP = \sqrt[n]{\prod_{i=1}^{n} TP_i} \tag{2}$$

To measure the performance of different ML system for complete pipelines, ADABench introduces a combined benchmark metric to measure the throughput at a scale factor: the *ADABench Metric − ADASps@SF*. It is a geometric mean of all use case metrics, where each use-case's metric is the geometric mean across the training and serving throughput.

2.2 PDGF

ADABench also adapts Parallel Data Generation Framework [6] (PDGF) [80] for data generation. It is a flexible, generic, and parallel data generator that can be used to generate large amounts of relational data very fast. The generator uses XML configuration files for data description and distribution. This simplified the generation of different distributions of specified datasets while ensuring the randomness of elements in it. As a result, ADABench is able to build a realistic simulation of use-cases and to quickly adapt to requirement changes in the data to model more complex relationships between the data.

PDGF provides three major functionalities [79].

1. *High data generation velocity* that scales in a reasonable time frame
2. *Data generation that incorporates characteristics and dependencies concerning the respective scale factor.* This enables a higher data volume to contain new characteristics. As a result, the data becomes more complex with an increasing scale factor.
3. *Introduction of error into the data* to simulate a real-world production environment. These errors can be e.g. missing values, outliers, wrong data types.

PDGF follows a black box plug-in approach. Its components can be extended and exchanged easily. A full documentation on how to interact with the components can be found at `https://www.bankmark.de/documentation/`. Custom plug-ins can also be implemented without consideration of programming aspects like parallelism or internal details.

PDGF can be configured by two XML-based configuration files. One file configures the data schema, while the other configures the generation routine.

[6] `http://www.bankmark.de/products-and-services/pdgf/`

Data Schema Configuration File specifies how the data are structured and what one specific field value looks like. Listing 2.1 illustrates an example of the general structure of the data schema configuration file. Properties of the schema can be defined in the beginning of the file. It then allows multiple tables and multiple fields inside a table to be defined with plug-ins. In this example, a table with name *table1* will be created with 20000 $(100 * 200)$ rows. The *SF* property is the scale factor and can also be modified during the run.

```
<schema>
  ...Internal PDGF settings...
  <property name="SF" type="double">200</property>
  ...
  <table name="table1">
    <size>100 * ${SF}</size>
        <field name="field1" size="20" type="...">
          <gen_GENERATOR1>
            ...generator specific elements...
          </gen_GENERATOR1>
        </field>
        ...
  </table>
  ...
</schema>
```

Listing 2.1: Data Schema Configuration File

Data Generation Configuration File specifies how data are written out. It is possible to specify the format of the data as well as where the data should be written to. Listing 2.2 illustrates an example of the general structure of the data generation configuration file. Properties of the generation routine can be defined in the beginning of the file. It then allows each table output to be defined separately how it is written out. In this example, the output of *table1* will be stored in the *output/* folder as a csv file.

```
<generation>
  ... Internal PDGF settings ...
  <property name="includeLabels" type="double">1</property>
  ...
  <schema name="default">
    <tables>
      <table name="table1">
        ... specific output configuration for table1 only ...
        <output name="CSVRowOutput">
        <fileTemplate>...</fileTemplate>
        <outputDir>output/</outputDir>
        <fileEnding>.csv</fileEnding>
        <delimiter>,</delimiter>
        <quoteStrings>...</quoteStrings>
        <charset>...</charset>
        <sortByRowID>...</sortByRowID>
        <header>...</header>
        ...
        </output>
      </table>
    </tables>
  </schema>
</generation>
```

Listing 2.2: Data Generation Configuration File

3 Background

This section introduces several topics that serve as the basis of all use-cases introduced in this thesis.An introduction to these topics provides a better understanding of the use-cases and their solutions proposed in this thesis. First, we introduce *Natural Language Processing*, which is relevant for the first use-case Customer Review Analysis. For the second use-case Ticket Classification, we utilize a Deep Learning method called Long Short-Term Memory (LSTM). It is a *Recurrent Neural Network* (RNN) that is capable of learning order dependence problems. Lastly, we explain *Time Series Forecasting* that serves as the background of the Inventory Demand Forecasting use-case.

3.1 Natural Language Processing

NLP is a study that tries to transform and analyze natural language text to enable computer extracting its information. Due to the nature of natural language, extracting its information is a challenging task. It combines several disciplines, such as computer science, linguistics, and mathematics, to analyze and derive the meaning of natural language text.

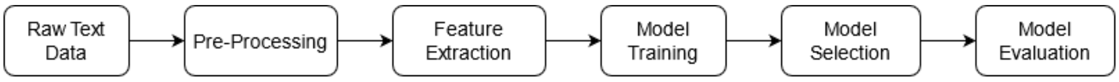


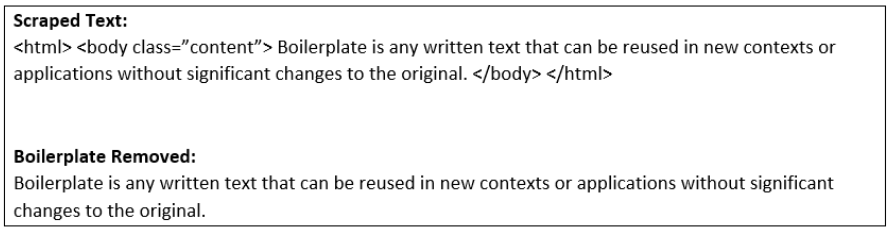
Figure 3.1: NLP Pipeline

Figure 3.1 illustrates a NLP pipeline that covers pre-processing, feature extraction, model training, model selection, and model evaluation. Every step in the pipeline has an equally important role at ensuring the quality of the NLP model. Pre-processing the data can significantly increase the performance of the model, extracting important feature from the data can improve the ability of the model to understand the data, and so on. All of these steps will be further discussed in the following sub-section:

3.1.1 Pre-Processing

The first step of the NLP pipeline is pre-processing. The goal is to improve the quality of the data by removing noises that are contain in the data or transforming the data into a certain standard.

A textual data can come from different sources, e.g. web-scraping, database, csv- or excel-file, etc. Processing text data starts by checking the data for corrupt files, character encoding, and removing noises from the data. Noises in the data can be seen as text file header, footer, HTML- and XML- tags, and markup data. It often happens that the text, which is scraped from the internet, will have them. However, these part of the data do not add much value to the information contained in the text. Therefore, they can be remove to obtain a cleaner raw text for further processing. Removing noises from the data does not only make the text more clear, but also reducing the size of it. Figure 3.2 shows an example where removing the HTML-tags makes the scrapped text shorter and easier to understand.


Scraped Text:
<html> <body class="content"> Boilerplate is any written text that can be reused in new contexts or applications without significant changes to the original. </body> </html>

Boilerplate Removed:
Boilerplate is any written text that can be reused in new contexts or applications without significant changes to the original.

Figure 3.2: NLP Pipeline

After removing the noises from the data, the text is split to sentences (sentence segmentation) and then to tokens (tokenization). The tokens are normally separated by white space and are elements that built the text. For example, a sentence *"The patty is perfectly grilled and juicy, the bun and the sauces are amplifying the whole burger even more!"* will become [“The”, “patty”, “is”, “perfectly”, “grilled”, “and”, “juicy”, “,”, “the”, ”bun”, “and”, “the”, “sauces”, “are”, “amplifying”, “the”, “whole”, “burger”, “even”, “more”, “!”] after tokenization. This process will be followed by normalization where the data will be processed on word/token level.

Normalization is considered to be a necessary processing step in any application that involves textual data. It comprises harmonizing spelling and capitalization and cleaning up unnecessary element of the data on word/token level. It might regard from abbreviation resolution, stemming, stopword removal, and/or lemmatization [63].

- **Removing Capitalization**
  It often occurs that a similar word with different formats will be recognized as different words. For example, *Normalization* and *normalization*, are identical words with different formats. Removing the capitalization on Normalization will tell that both words are identical (normalization).
- **Stop words removal**
  Some commonly used words such as “the”, “a”, “an”, “in” do not hold much information in compare to the other words in the sentence. These words are called stop words. And removing them can lead to performance increase and reducing the size of the data. The challenge with this step is that often times, we need to list the stop words ourselves.
- **Stemming**
  Stemming means reducing the word into its word stem, base or root form. It normally cuts off the end of the words[60]. For example, {*organizes*, *organized*, *organizing*} → *organize*.
- **Lemmatization**. Lemmatization is a complex reduction of inflectional forms of a word so that they can be analyzed together as a single word, identified by its lemma, the invariant root form of a word [60]. For example, {*eat*, *ate*, *eaten*} are seen as three different words and can be reduced to a single lemma, *eat*.

Figure 3.3 shows an example, where multiple sentences are tokenized and normalized. By removing the capitalization and special characters, stemming, and lemmatization, the base words that build all of the sentences can be extracted for the next step.

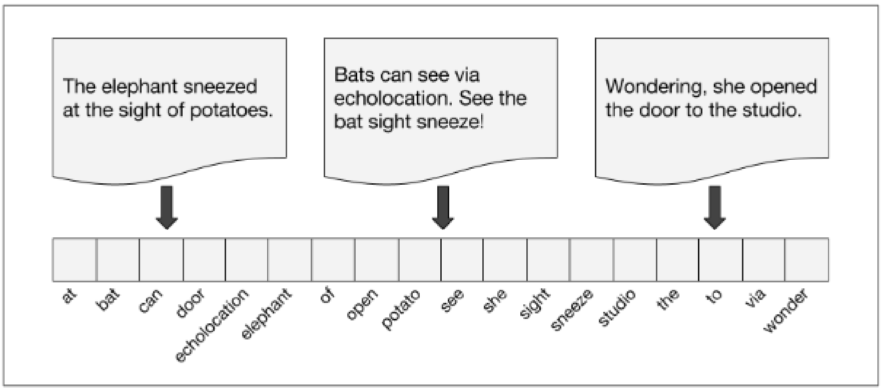

The elephant sneezed at the sight of potatoes.
Bats can see via echolocation. See the bat sight sneeze!
Wondering, she opened the door to the studio.
at
bat
can
door
echolocation
elephant
of
open
potato
see
she
sight
sneeze
studio
the
to
via
wonder

Figure 3.3: Tokenization [16]

3.1.2 Feature Extraction

The second step of the NLP pipeline is feature extraction. The goal is to extract important information from the pre-processed data to be used as the input of the model.

Machine learning algorithms operate on numeric feature space, expecting input as a two-dimensional array where rows are instances and columns are features. The process of encoding the text data in a numeric feature space is called *feature extraction* or *vectorization* [16].

"Bag-of-words" model is the start of the vectorization. The idea is that meaning and similarity of documents are encoded in the specific vocabulary used in each document. It uses an approach that documents that have a similar meaning have more common words in compare to documents that are very different in meaning. For example, an article about basketball and Michael Jordan will share a lot of common words, whereas an article about basketball and Mike Tyson will not share many words in common.

Vectorization starts with creating a vector whose length is equal to the vocabulary/token of the entire documents. The entries in the vector are unique and can be used to describe a single document.

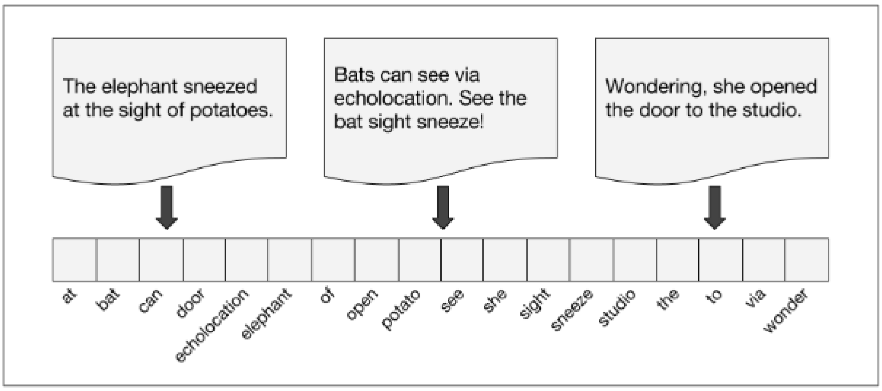

The elephant sneezed at the sight of potatoes.
Bats can see via echolocation. See the bat sight sneeze!
Wondering, she opened the door to the studio.
at
bat
can
door
echolocation
elephant
of
open
potato
see
she
sight
sneeze
studio
the
to
via
wonder

Figure 3.4: Tokenization [16]

The simplest vectorization is to simply fill in the vector with the frequency of each word as it appears in the document. Each document is represented as the multiset of the tokens that compose the document and the value for each token in the vector is its frequency of appearance. Figure 3.5 shows an example of a vectorization. The frequency of the token in the document is counted and filled to the vector creating a dictionary whose keys are the tokens in the document and whose values are the frequency that token appears in the document.

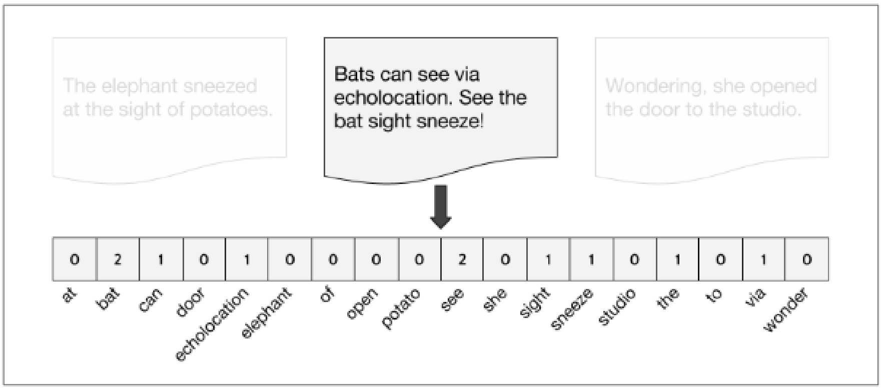

The elephant sneezed at the sight of potatoes.
Bats can see via echolocation. See the bat sight sneeze!
Wondering, she opened the door to the studio.
0 2 1 0 1 0 0 0 0 2 0 1 1 0 1 0 1 0
at
bat
can
door
echolocation
elephant
of
open
potato
see
she
sight
sneeze
studio
the
to
via
wonder

Figure 3.5: Simple Vectorization [16]

While it looks simple, straightly counting the frequency of tokens have a major problem. Some features (tokens) are orders of magnitude more significant than other less frequent tokens. The whole vector is not needed to describe a document. And taking the whole vector to describe a document will have a significant impact on some models, particularly models which expect normally distributed features.

The feature vector produced with bag-of-words model can already be used for the next process in the pipeline. However, this model has a disadvantage. It only describes the document in a stand-alone fashion and not taking into account the context of the dataset. Tokens that appear frequently in a document have more relevance to that document, particularly if they appear infrequently in the rest of the dataset. Considering the relative frequency or rareness of tokens in a document against their frequency in the whole dataset can bring a new important insight. For example in a sport article dataset, a document that discuss basketball tokens such as "dunk", "air-ball", and "lay-up" will appear more frequently in compare to documents in the rest of dataset that discuss other sports. These tokens gives a better indication that the document is about basketball, whereas tokens, such as "score", "match", and referee" have a lower indication of basketball because these tokens also appear often in other sport related documents.

Often times, a sequence of N-words hold a new meaning and create a more valuable feature. Instead of having two tokens "San" and "Fransisco", a token "San Fransisco" makes a better feature. A sequence of N-words is called N-gram. N-gram can help make next word prediction based on the previous words. A N-gram model estimates the probability of a word w given a history h [50]. A N-gram model of a size 1 is referred as "unigram", size 2 is "bigram", size 3 is "trigram". The bag-of-words model can be seen as a unigram. And our "San Fransisco" example is a bigram. A bigram model predicts the occurrence based on its previous word only. Similarly, a trigram model predicts based on its previous two words.

From the previous example, with a bigram model we can predict that it is more likely that the word "Fransisco" will appear after the word "San" in compare to other words. A N-gram model computes the probability using the chain rule of the probability

$$\begin{aligned} P(X_1, ..., X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_{1:2})...P(X_n|X_{1:n-1}) \\ &= \prod_{k=1}^{n} P(X_k|X_{1:k-1}) \end{aligned} \tag{3}$$

Applying the chain rule in Equation 3 to the words resulting in

$$\begin{aligned} P(w_{1:n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1:2})...P(w_n|w_{1:n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_{1:k-1}) \end{aligned} \tag{4}$$

The chain rule shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words. Equation 4 suggests that the joint probability of an entire sequence of words can be estimated by multiplying together a number of conditional probabilities.

A N-gram model is an efficient compression of large amounts of text. To keep the size manageable, N-grams that occur with a frequency below a particular threshold can be filtered. The N-gram model helps extracting important features in the document and especially helpful on much smaller dataset. It normally boosts the performance of the model, because it does not only considering individual words, but also sequence of words, which in many cases important for the meaning of a phrase or a sentence.

TF-IDF, *term frequency-inverse document frequency*, encoding normalizes the frequency of tokens and N-grams in a document with respect to the rest of the dataset. The idea is to accentuate terms that are very relevant to a specific document. Figure 3.6 shows an example where the word "studio" is more important, and thus have more weight, to this document than other words, because it does not appear in any other document.

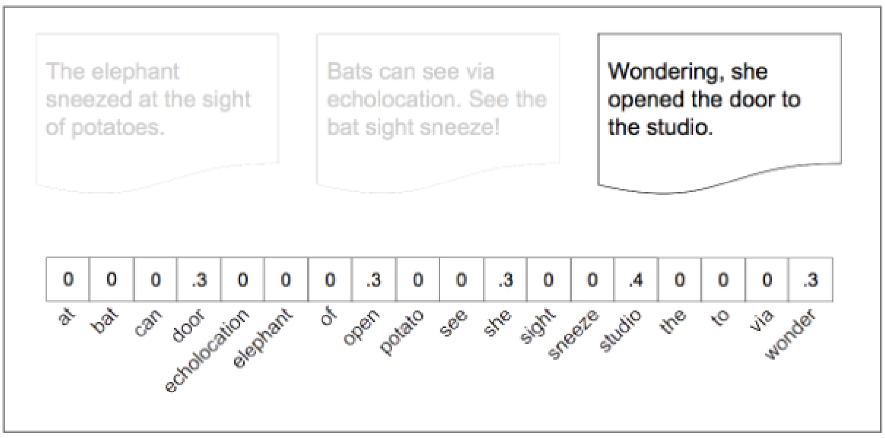

The elephant sneezed at the sight of potatoes.
Bats can see via echolocation. See the bat sight sneeze!
Wondering, she opened the door to the studio.
0 0 0 .3 0 0 0 .3 0 0 .3 0 0 .4 0 0 0 .3
at
bat
can
door
echolocation
elephant
of
open
potato
see
she
sight
sneeze
studio
the
to
via
wonder

Figure 3.6: TF-IDF Encoding [16]

TF-IDF is computed on a per-term basis, such that the relevance of a token to a document is measured by the scaled frequency of the appearance of the term in the document, normalized by the inverse of the scaled frequency of the term in the entire dataset [16]. The TF-IDF is divided into two steps, namely *Term Frequency* (TF) and *Inverse Document Frequency* (IDF).

Term Frequency assigns a *weight* to each term in a document depending on the number of occurrences of the term in the document. It computes a score between a query term t and a document d, based on the weights of t in d. This weighting scheme is denoted as $\mathrm{tf}_{t,d}$, with the subscripts denoting the term and the document in order [59].

Term frequency alone suffers from a critical problem: all terms considered equally important. However, in most cases certain terms have little or no discriminating power in determining relevance. For example, a collection of documents about sport is likely to have the term "score" and "match" in almost every document. The idea here is to scale down the term of weights of terms with high *collection frequency*, defined to be the total number of occurrences of a term in the collection.

Inverse Document Frequency uses document frequency df to scale terms' weight. Document Frequency df_t is defined to be the number of documents in the collection that contain a certain term t. In order to discriminate between documents for the purpose of scoring it is better to use a document-level statistic in compare to collection-wide statistic for the term. While terms can have similar frequency collectively, they can behave differently on document-level. One term can appear more often than others on a document or does not appear at all. The intuition is that, the few documents that contain a certain term to have a higher weight for a query on that term.

To address this problem, terms that appear on many documents should have lower weight than terms that appear on fewer documents. Denoting the total number of documents in a collection by N, idf_t of a term t is defined as:

$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t} \tag{5}$$

Through this formula, the idf of a rare term will be higher than a more common term.

Tf-idf weighting TF-IDF combines both concepts to produce a composite weight for each term in each document. The *tf-idf* weighting assigns to term t a weight in a document d given by

$$\text{tf-idf}_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t \tag{6}$$

[59] explains that, $\text{tf-idf}_{t,d}$ assigns to term t a weight in document d that is

1. highest when t occurs many times within a small number of documents (thus lending high discriminating power to those document)
2. lower when the term occurs fewer times in a document, or occurs in many documents (thus offering a less pronounced relevance signal)
3. lowest when the term occurs in virtually all documents

A document can be seen as a vector with one component corresponding to each term in the dictionary with the a weight for each term given by the equation from Equation 6.

For dictionary terms that do not appear in the document, this weight is zero. Using this, the score of a document can be defined as:

$$\text{Score}(q, d) = \sum_{t \in q} \text{tf-idf}_{t,d} \tag{7}$$

3.1.3 Model Training

The third step of the NLP pipeline is model training. Here, all the information that is extracted earlier will be used as the input. There exist many models that can handle NLP tasks. In this thesis will be focusing on the Naïve Bayes model due to its well-known performance and simplicity.

Naïve Bayes (NB) is a classification model, which is based on the Bayes' Theorem. It describes the probability of an event, based on prior knowledge of conditions related to the event [61].

$$P(\text{class}|\text{features}) = \frac{P(\text{features}|\text{class}) \cdot P(\text{class})}{P(\text{features})} \tag{8}$$

Equation 8 calculates the posterior probability $P(\text{class} \mid \text{features})$ from the likelihood $P(\text{features} \mid \text{class})$, class prior probability $P(\text{class})$, and predictor prior probability $P(\text{features})$.

It uses conditional probability to assign each observations to the most likely class, given its predictor values. It also uses a strong assumption where each of the predictors are conditionally independent of one another. However, even if the assumption is not true, NB has been shown to be robust against this assumption [31]. Also, due to the simplicity of the model (it only has 2 parameters, classes and features), it is relatively immune to overfitting [67].

Bayes classifier calculates all all conditional probabilities of each predictor given each class. They are used alongside with prior probabilities of each class to predict the class of new data. As shown in Equation 8, the posterior probability of each class will be first calculated by multiplying every its conditional probability given the class. Afterwards, the new data is assigned to the class with the highest posterior probability. In other words, we want to find

$$\arg\max_i P(\text{class}_i|\text{observation}_i) \tag{9}$$

One of the disadvantage of NB is that if there is a no occurrences of a class label or a certain attribute value together then their prior probability will be zero. This problem is called *zero-frequency problem* [93] and it skews the performance of the classification. Bayes classifier handles this problem by applying a add-one or Laplace smoothing, which involves initializing each count to 1 rather than to 0 [60]. Another problem with NB emerges when correlation between predictors exists. A correlation in the features will have an adverse effect on the NB assumption of independence.

Although NB is one of the simplest probabilistic classifier model. In practice it often competes well with more sophisticated models [83]. Some comparison researches on text classification methods [26, 40, 100, 60] have shown that Naïve Bayes produces a better than the average result if not performing the best. NB's main strength is its efficiency: training and classification can be accomplished with one pass over the data. Because it combines efficiency with good performance metric it is often used as a baseline in text classification research [60].

UC1, described in Section 4.1, shows an example of Naïve Bayes for sentiment analysis. It does not only run fast, but also managed to achieve high performance metric.

Multinomial Naïve Bayes (MNB) is one NB variant used for multinomially distributed data. In MNB, every word w_i has a part in determining to which class $c \in 1, ..., C$ a document t_i should be assigned based on its probability $\Pr(c|t_i)$ [53]. Based on Equation 8, we are looking for:

$$\Pr(c|t_i) = \frac{\Pr(c) \cdot \Pr(t_i|c)}{\Pr(t_i)}, \qquad c \in C \tag{10}$$

3.1.4 Model Selection

To build a robust model, it is important to test the model on an independent dataset. The goal of the fourth step in the NLP pipeline is to build a general model that is neither overfit nor underfit the training data. In a data-rich situation, the best approach is to randomly divide the dataset into three parts: a training set, a validation set, and a test set. The training set is used to train and fit the models; the validation set is used to estimate the prediction error for model selection; the test set is used for assessment of the generalization error of the final chosen model [43]. There is no absolute rule in how the split should look like. This is totally dependent on the signal-to-noise ratio of the underlying function and the complexity of the models. A typical split might be 50% for training and 25% for validation and testing. Figure 3.7 illustrates the partition of the dataset.

Training
Validation
Test

Figure 3.7: Partition of dataset

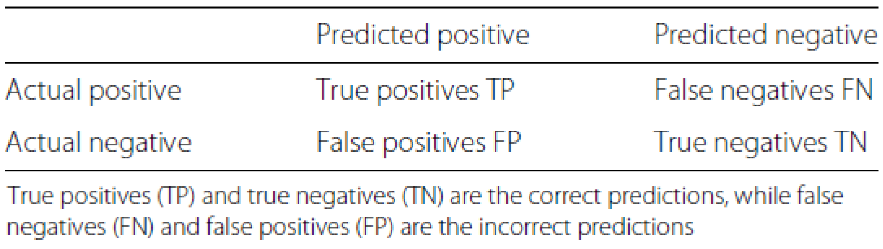
3.1.5 Evaluation

The last step in the NLP pipeline is to evaluate the prediction of the model. There are different performance metric that can be used depending on the several factors, e.g. distribution of the data, number of classes, and type of prediction.

A traditional classification metric is the ratio between the number of correctly classified samples and the overall number of samples. This metric is called *accuracy* and it does not only work for binary class classification, but also multi class classification. Accuracy metric is calculated based on four terms [24]:

1. *True positives* (TP) Actual positives that are correctly predicted positives
2. *False negatives* (FN) Actual positives that are wrongly predicted negatives;
3. *True negatives* (TN) Actual negatives that are correctly predicted negatives;
4. *False negatives* (TP) Actual negatives that are wrongly predicted positives;

The four terms can be presented in a 2×2 table called *confusion matrix* (illustrated in Figure 3.8).



	Predicted positive	Predicted negative
Actual positive	True positives TP	False negatives FN
Actual negative	False positives FP	True negatives TN

True positives (TP) and true negatives (TN) are the correct predictions, while false negatives (FN) and false positives (FP) are the incorrect predictions

Figure 3.8: The standard confusion matrix M

To analyze the confusion matrix, statisticians introduced some useful statistical rates to immediately describe the quality of a prediction [76]. Recall or Sensitivity (Equation 11) is the proportion of real positive that are correctly predicted positives. Its desirable feature is that it reflects how many of the relevant cases the predicted positives pick up. Precision or Confidence (Equation 12) is the proportion of predicted positives that are correctly real positives. This information is often used to determine how accurate the model prediction of a certain class.

$$\text{Recall} = \text{Sensitivity} \qquad = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{11}$$

$$\text{Precision} = \text{Confidence} \qquad = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{12}$$

To see how the negative class performed, Inverse Recall or Specificity and Inverse Precision are used. Inverse Recall or Specificity (Equation 13) is the proportion of real negatives that are correctly predicted negatives and thus also known as the true negative rate. Inverse precision (Equation 14) is the proportion of predicted negatives that are correctly real negatives and thus also known as the true negative accuracy.

$$\text{Inverse Revall} = \text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{13}$$

$$\text{Inverse Precision} = \frac{\text{TN}}{\text{TN} + \text{FN}} \tag{14}$$

Another important metrics that can be extracted from the confusion matrix are the accuracy and the misclassification rate. Accuracy (Equation 15) gives the overall accuracy of the model illustrating the proportion of total samples that are correctly predicted. Conversely, misclassification rate (Equation 16) shows the proportion of the incorrect prediction. It can also be seen as $(1 - \text{Accuracy})$

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{15}$$

$$\text{Missclassification Rate} = \frac{\text{FP} + \text{FN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{16}$$

Using the precision and recall, the harmonic mean (*F1-Score*) (Equation 17) can be calculated to better evaluate model's accuracy. Its value range from 0, indicating either the precision or recall is zero, to 1, indicating perfect precision and recall.

$$\text{F1-Score} = \frac{2 \cdot \text{TP}}{2 \cdot (\text{TP} + \text{FP} + \text{FN})} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{17}$$

Accuracy metric may seem easier to understand. However, it does not take into account how the data is distributed. In case of an imbalance dataset, accuracy often still shows a high result as it only looks into correctly classified data. On the other hand, F1-Score is balancing precision and recall. By taking false negatives and false positives into account, F1-Score penalizes extreme values.

An example to show advantage of F1-score: Consider a binary classification task with a data that consists of 100 samples with 90 negative samples and 10 positive samples. Suppose the classifier only predicts negative. The classifier will have 90% of accuracy, while f1-score will be 0 because the recall is 0.

An alternative metric unaffected by the imbalance dataset is *Matthews Correlation Coefficient* (MCC). It is generally regarded as a balanced metric because it takes all confusion matrix quantities (TP, TN, FP, and FN) into account, which gives a better summary of the performance of classification algorithms. MCC takes value in the interval $[-1, 1]$, with 1 showing complete agreement, -1 a complete disagreement, and 0 showing that the prediction was uncorrelated [19]. Equation 18 shows how MCC can be calculated.

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}} \tag{18}$$

$$\tag{19}$$

3.2 Recurrent Neural Network

Artificial Neural Network (ANN), usually called Neural Network, is a biologically-inspired programming paradigm which enables a computer to learn from observational data. ANNs are composed of a large number of highly interconnected processing elements (neurons) working together to solve a problem. As illustrated in Figure 3.9, ANNs are represented by node layers, containing an input layer, one or mode hidden layers, and an output layer.

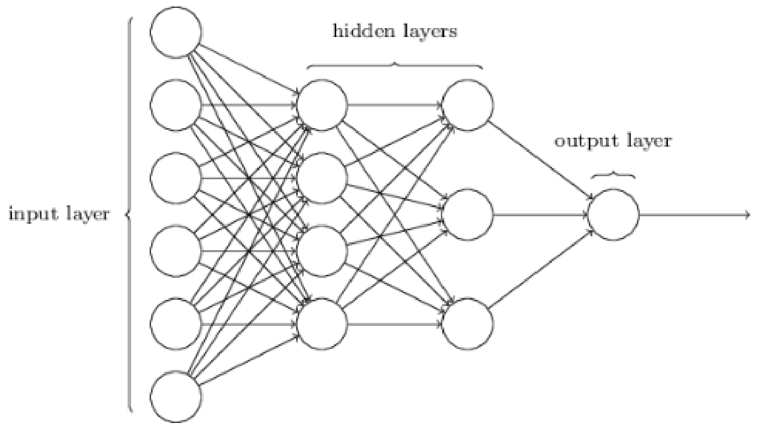


Figure 3.9: An Artificial Neural Network [70]

In recent times, ANN has become prominent and helpful model for data science tasks in many disciplines. Its full applications can be evaluated with respect to data analysis factors such as accuracy, processing speed, latency, performance, fault tolerance, volume, scalability and convergence [66]. ANNs can be developed and often used for image recognition, pattern recognition, and natural language processing. Nowadays, ANNs are mostly used for universal function approximation in numerical paradigms because of their excellent properties of self-learning, adaptivity, fault tolerance, nonlinearity, and advancement in input to an output mapping.

A Recurrent Neural Network (RNN) is a class of ANN where connection between nodes form a directed graph along a temporal sequence. It addresses the problem of a traditional ANN of not being able to keep previous information. Figure 3.10 illustrates how a RNN persists information. A looks at some input x_t and outputs a value h_t. The loop allows information to be passed from one step of the network to the next.

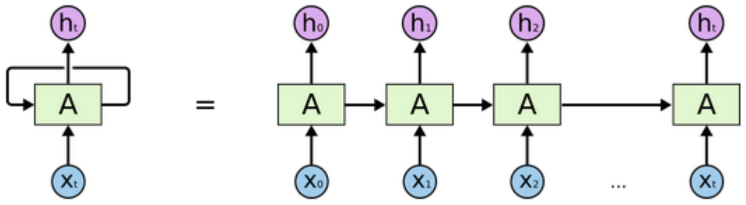


Figure 3.10: A Recurrent Neural Network [72]

Long short-term memory (LSTM) is a RNN architecture, that is capable of learning order dependence in sequence prediction problems. Although standard RNN architecture can use contextual information when mapping between input and output sequences, it is quite limited. The problem is that the influence of a given input on the hidden layer, either decays or blows up exponentially as it cycles around the network's recurrent connections. This effect is referred as the *vanishing gradient problem* [38]. Figure 3.11 illustrates the vanishing gradient problem. The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity). The sensitivity decays over time as new inputs overwrite the activations of the hidden layer, and the network forgets the previous inputs.

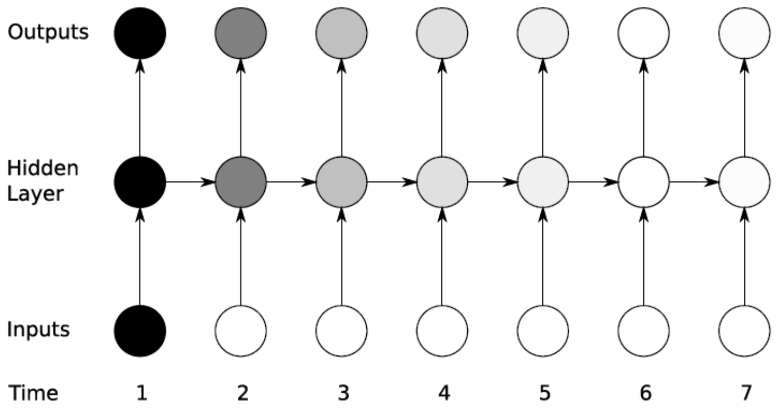


Figure 3.11: Vanishing Gradient Problem [38]

LSTM is designed to bridge the time intervals, remembering information for long periods of time [72]. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNN, this repeating module will have a very simple structure, such as a single tanh layer (Figure 3.12). LSTM also has similar structure, but with a different repeating module structure (Figure 3.13).

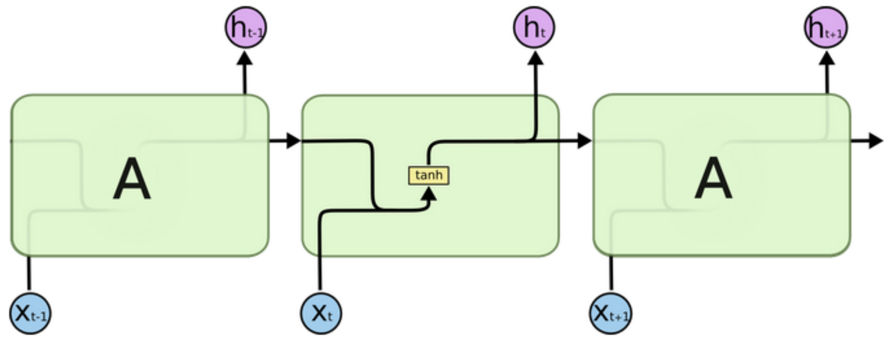


Figure 3.12: The repeating module in a standard RNN contains a single layer [72]

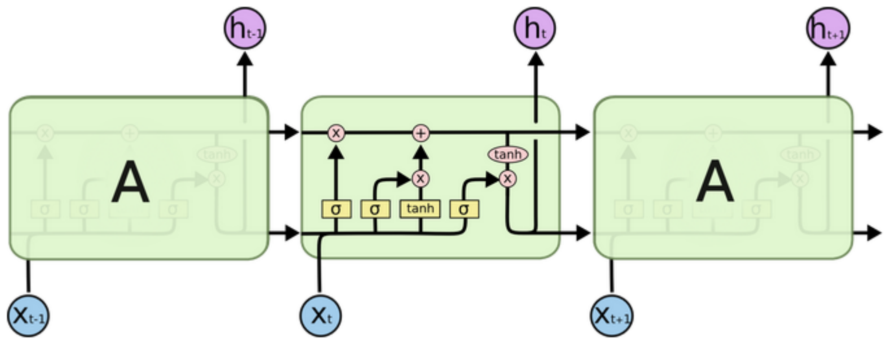


Figure 3.13: The repeating module in an LSTM contains four interacting layers [72]

The key to the LSTM is the *cell state* [72]. It gives LSTM the ability to remove or add information to the cell state, carefully regulated by structures called gates. It also has a *hidden state* that acts like a short term memory (Figure 3.14). Gates are a way to optionally let information through. They are composed out of a sigmoid neural network layer and a pointwise multiplication problem. It allow the cell to store and access information over long periods of time, thereby mitigating the vanishing gradient problem.

An LSTM has three layers, to protect and control the cell state. They are referred as input layer, output layer, and forget layer. The input and output gates multiply the input and output of the cell while the forget gate multiplies the cell's previous state. The forget gate is a sigmoid layer so that the gate activations are between 0 (gate closed) and 1 (gate opened). The input and output gates are usually tanh or logistic sigmoid layer.

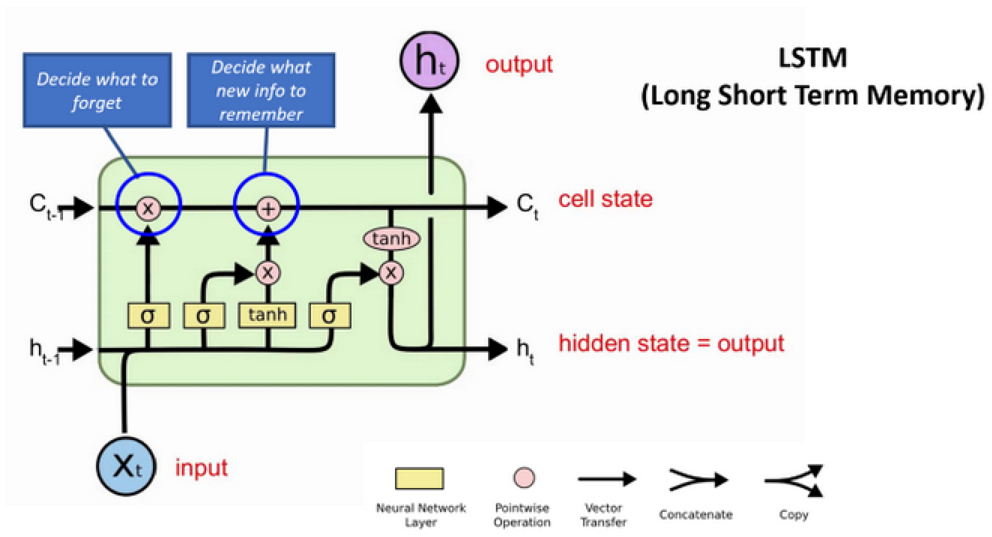
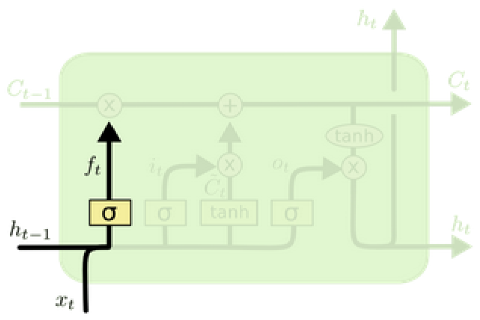


Figure 3.14: Overview of LSTM layer [72]

LSTM Walkthrough

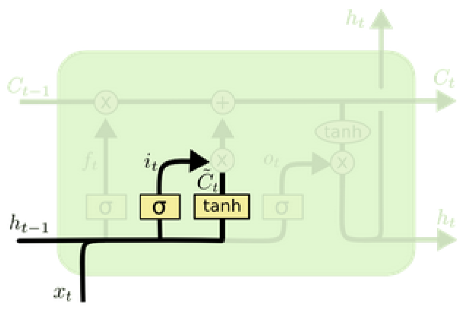
The forget gate layer decides which information is going to be thrown away from the cell state. It is made of a sigmoid layer, taking h_{t-1} and x_t as inputs, and outputs a number between 0("completely forget this") and 1 ("completely keep this") for each number in the cell state C_{t_1} (Figure 3.15).



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Figure 3.15: Forget gate layer [72]

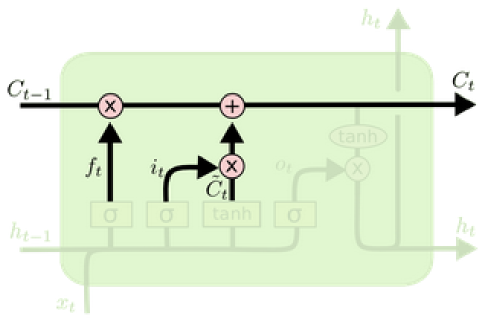
The next step is to decide which new information is going to be stored in the cell state. The input gate layer calls a sigmoid layer to decide which values will be updated, while the tanh layer create a vector of new candidate values that could be added to the state (Figure 3.16).



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Figure 3.16: Input gate layer [72]

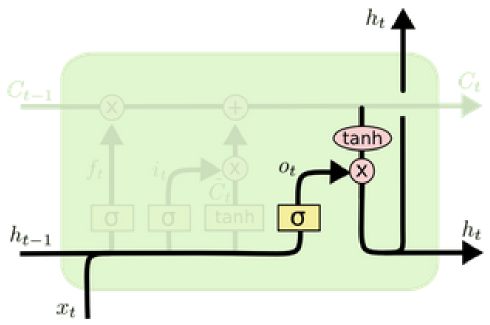
Both results from the input layer are combined to create the new candidate values, scaled by how much we decided to update each state value. Adding to it, the previous state is multiplied by the result from the forget fate layer. In other word, we drop the information from the old state and adding new information in this step (Figure 3.17).



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 3.17: Cell state updates [72]

Finally, the output will be based on filtered cell state. The sigmoid gate filters out parts of the cell state that are not going to output. Then, the cell state is put through tanh gate and multiply by the output of the sigmoid gate, so that only necessary parts are passed to the next neural network layer (Figure 3.18).



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 3.18: Output gate layer [72]

LSTM is well-suited approach for anything that has a sequence. In NLP, for example, meaning for a word can change depends on the ones that preceded it. Due to its ability to naturally model sequential correlation in the text, LSTM-based approach is quite popular in NLP-related tasks. [56, 99, 98] are some examples that utilize LSTM to handle text classification.

Bidirectional LSTM (BiLSTM)

For many sequence processing tasks, it is useful to analyze not only from the past, but also the future. It consists of two LSTMs: one taking input in a forward direction, and the other in a backward direction. Using two hidden states (outputs) combined, BiLSTM is able to exploit information from both the past and the future [39]. BiLSTM effectively increases the amount of information available to the network, improving the context available to the algorithm.

3.3 Time Series Forecasting

A time series is a sequence of observations taken sequentially in time with intervals. These intervals can vary from yearly, monthly, daily, or hourly. Time is the independent variable and the observations are the time-dependent variables. Many sets of data appear as time series in a lot of fields: a monthly sequence of the quantity of goods shipped from a factory, a weekly series of traffic accidents, daily stock price movement, hourly observations of the temperature, and so on. Time series analysis is concerned with techniques for the analysis of the time-dependent variables [92]. It is an approach to analyze time series data to extract meaningful characteristics of data and generate useful insights. Often discussed along with time series analysis is time series forecasting. While time series analysis focuses on understanding the data, time series forecasting utilizes historical data to predict future.

3.3.1 Time Series Forecasting Models

Time series forecasting can be either univariate or multivariate. *Univariate time series* refers to a time series that consists of single observations recorded over equal time intervals in a sequential manner. *Multivariate time series* on the other hand are based on multiple observations. Using these observations, temporal dependence model look into correlations between past and present values. This serves as the basis of a popular time series techniques called ARIMA (Autoregressive Integrated Moving Average) [28].

Although effective, ARIMA has some limitations. ARIMA works by first transforming a non-stationary data into stationary data. This means that ARIMA model is prone to trend error and it is more likely to fail to capture any seasonality [90]. Moreover, ARIMA models produces a poor long term forecast when the data has multiple turning points [95].

A time series can be decomposed into four components: a trend component, a cycle component, a seasonal component, and a remainder/noise component [48]. Assuming we have an additive decomposition, a time series can be thought of as

$$y_t = T_t + C_t + S_t + I_t$$

where

- T_t, the trend component at time t, which reflects the long-term progression of the series
- C_t, the cyclical component at time t, which reflects repeated but non-periodic fluctuations
- S_t, the seasonal component at time t, which reflects seasonal patterns over a fixed and known period
- I_t. the irregular or noise component at time t, which describes random, irregular influences

Alternatively, time series decomposition can be used to build a multiplicative model, in which the model can be thought of as

$$y_t = T_t \times C_t \times S_t \times I_t$$

An additive model is often used when the variations around the trend do not vary with the level of the time series whereas a multiplicative model would be appropriate if the trend is proportional to the level of the time series.

Based on the decomposition, instead of using correlation between values a time series can be seen as a summation of multiple trends. Generalized Additive Model (GAM) [42] behaves similarly to that of regression, except that instead of summing the individual predictor, it uses sum of smooth function to predict the outcome.

GAM can be formally written as [94]:

$$\begin{aligned} &g(E(y_i)) = \beta_0 + f_1(x_{i1}) + ... + f_p(x_{ip}) + \epsilon_i, \\ &y_i \sim \text{some exponential family distribution} \end{aligned} \tag{20}$$

where $i = 1, ..., N$, f is a link fnction (identical, logaritmic, or inverse), y is a response variable, $x_1, ..., x_p$ are dependent variables, β_0 is an intercept, $f_1, ..., f_p$ are unknown smooth functions and ϵ_i is an i.i.d. random error.

Because GAMs are based on functions rather than variables, they are not restricted by the linearity assumption in regression that requires predictor and outcome variables to move in a straight line. Furthermore, we can isolate and study effects of individual functions in a GAM on resulting predictions.

Another approach to do time series forecasting is to use exponential smoothing [**?**]. Forecasts produced using Exponential Smoothing methods are weighted averages of past observations, with the weights decaying exponentially over time. It works under the assumption that the time series is stationary. While this method is good in making short-term forecasts under this assumption, it performs poorly when the time series contains trends or seasonality.

Double Exponential Smoothing [45] is an extension to Exponential Smoothing that explicitly adds support for trends. This method involves one forecast equation and two smoothing equations (one for the level and one for the trend). This method is then further extended to also capture seasonality, making it to have three smoothing equations. Triple Exponential Smoothing is often called Holt-Winter Exponential Smoothing [22].

Besides statistical approaches, there are also deep-learning based approaches such as DeepAR [84]. DeepAR tailors a similar LSTM-based recurrent neural network architecture to the probabilistic forecasting problem. It features a minimal manual feature engineering to capture complex, group-dependent behavior and ables to learn from similar items, to provide forecasts for items with little or no history at al, a case where traditional single-item forecasting methods fail [84].

Statistical and deep learning methods in forecasting have their own advantages and disadvantages. Deep learning forecasting offers high flexibility, simplicity, and immediacy of use since they do not require a time-consuming model selection for each different cell. On the other hand, statistical methods have their superiority in providing more precise forecasting results. However, they require domain knowledge and computationally expensive techniques to select the best parameters [21, 57, 74].

3.3.2 Existing Tools and Libraries

Forecasting is a challenging task that involves multiple expertises and building a statistic model from scratch is not an easy task. There exists several libraries on different languages that can be used to help us developing a time series model, such as Python statsmodels library[7] and R forecast package[8]. Facebook also provides a forecasting tool based on an additive model called Prophet [9]. In addition to that, Amazon provides a deep learning based forecasting tool called GluonTS [10] [11]. In this work, we will focus on using Prophet as our tool of choice.

Despite having well-developed forecasting libraries, producing high quality forecasts is not an easy problem for either engineers or analysts. Engineers at Facebook have observed two major topics in the practice of creating business forecasts [90]. First, completely automatic forecasting techniques can be hard to tune and are often too inflexible to incorporate useful assumptions or heuristics. Second, specialized data science skill requirement makes it hard to find analysts who can produce high quality forecasts. These topics have become Facebook's motivation to create a forecasting tool for experts and non-experts to make high quality forecasts that keep up with demand.

With this in mind, Facebook developed an open source forecasting tool available in both Python and R, called Prophet [11] [90]. Prophet is optimized for the business forecast tasks that is normally encountered at Facebook. They typically have following characteristics [5]:

1. hourly, daily, or weekly observations with at least a few months (or even years) of history
2. strong multiple "human-scale" seasonalities: a day of week and time of year
3. important holidays that occur at irregular intervals that are known in advance (e.g. the Super Bowl)
4. a reasonable number of missing observations or large outliers

[7]https://www.statsmodels.org/
[8]https://cran.r-project.org/web/packages/forecast/forecast.pdf
[9]https://facebook.github.io/prophet/
[10]https://ts.gluon.ai/index.html
[11]https://facebook.github.io/prophet/

5. historical trend changes, for instance due to product launches or logging changes
6. trends that are non-linear growth curves, where a trend hits a natural limit or saturates

Prophet is designed to produce forecasts that are often as accurate as those produced by skilled forecasters, with much less efforts. Non-experts who may have domain knowledge about the data generating process but little knowledge about time series models and methods can improve or tweak forecasts using a variety of easily-interpretable parameters.

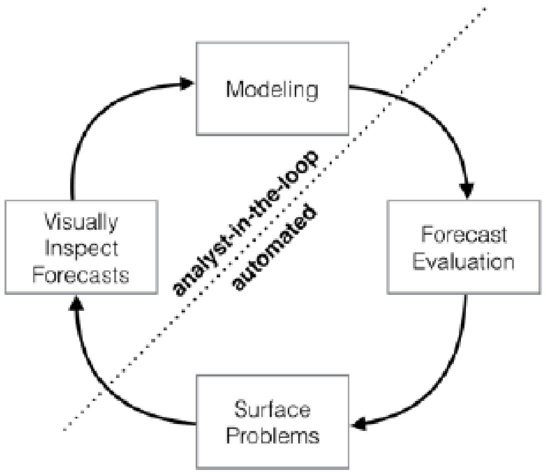


Figure 3.19: Schematic view of Facebook analyst-in-the-loop [90]

Figure 3.19 summarizes Facebook's analysts-in-the-loop approach to business forecasting at scale [90]. It starts by modeling the time series using a flexible specification that has a easily-interpretable parameters. It then produces forecasts for this model and a set of reasonable baselines across a variety of historical simulated dates and evaluate its performance. When there is poor performance or other aspects of the forecasts warrant human intervention, it flags these potential problems to a human analysts in a prioritized order. The analysts can then inspect the forecast and potentially adjust the model based on this feedback.

Prophet uses a decomposable time series model [41] with three main components: trend, seasonality, and holidays [90]. Equation 21 show how all three components are combined, where $g(t)$ is the trend function which non-periodic changes in the value of the time series, $s(t)$ represents periodic changes (e.g., weekly and yearly seasonality), and $h(t)$ represents the effects of holidays which occur on potentially irregular schedules over one or more days. The error term ϵ_t represents any idiosyncratic changes which are not accommodated by the model.

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t \tag{21}$$

At its core, Prophet is a GAM model. It tries to fit several linear and non-linear functions of time as components. Coupling it with exponential smoothing technique like in Holt-Winter technique [23], Prophet is able to model seasonality as an additive component. Prophet is framing the forecasting problem as a curve-fitting exercise rather than looking explicitly at the causal relationships between past and present observation within a time series.

An example of a characteristic forecast is shown on Figure 3.20. It illustrates a forecast of log-scale page views of an American football player, Peyton Manning's Wikipedia page. The black dots represent the number of views, while blue line represents the fitted model and the forecast.

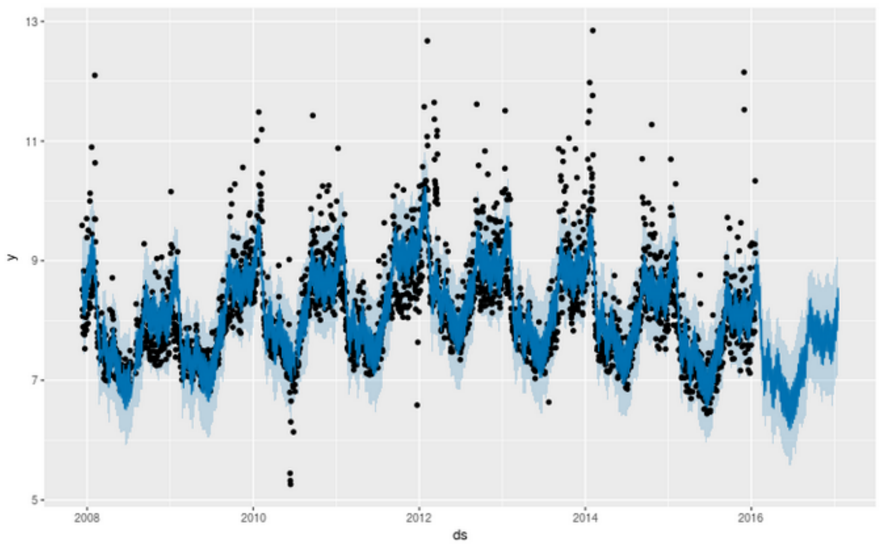


Figure 3.20: Characteristic Forecast of Log-Scale Page-Views of Peyton Manning's Wikipedia Page [5]

An important benefit of the decomposable model is that it allows us to look at each component of the forecast separately. Prophet provides a components plot (trend, yearly, and weekly) which graphically describes the model it has fit:

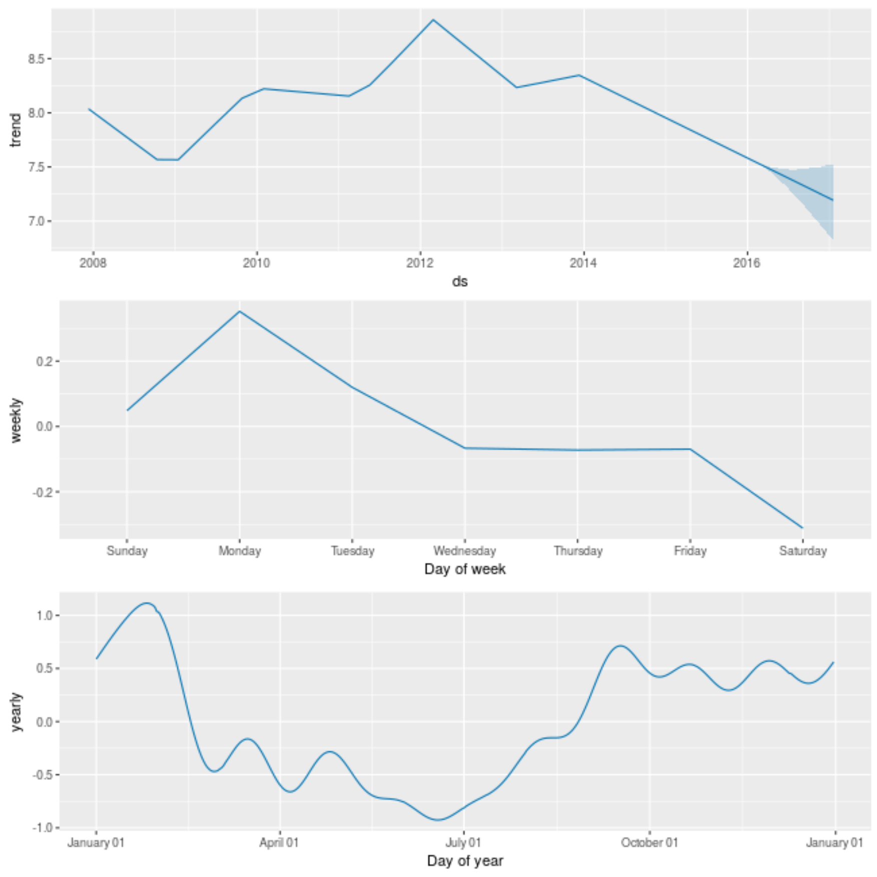


Figure 3.21: Components Plot of the Forecast [5]

Through Figure 3.20 and Figure 3.21 we can immediately notice that there is a certain seasonality and trend on the data. The yearly seasonal component shows a correlation between the search on Peyton Manning's page and football season and playoffs, e.g. there are less views on off-season in compare to on-season. The weekly seasonal component also shows that there are more views on the day of and after games (Sundays and Mondays). The trend component shows his popularity over the years. In this case, the downward trend is caused by his retirement on 2016.

4 Use-Cases

Machine Learning (ML) and Deep Learning (DL) stands out currently in the industry as a transformational technology. It does not only pique the interest in the IT industry but also other non-IT industries, such as healthcare [36], manufacturing [65], energy [86], and agriculture [55]. It brings a potential to various industries enabling applications that can provide an incremental lift beyond that from more traditional methodology. To name few examples: The Healthcare industry starts to use ML/DL systems to detect various diseases from observational data [36]. E-commerces use recommender systems to give a better product suggestion to their customers [85]. Energy sector utilizes it to forecast how much power needed at a given time. This leads to a more efficient operation and not only saving customers some costs, but also reducing annual CO2 emissions from the power generation [86].

ADABench derives its use-cases from retail business application and analysis tasks common in online businesses according to their market potential identified by McKinsey [25]. As previously mentioned in Section 2, this thesis extends additional use-cases that has a substantial potential of applying ML/DL techniques in its industry.

In the rest of this section, we introduce three use-cases: *Customer Review Analysis*, *Ticket Classification*, and *Inventory Demand Forecasting*. These use-cases have enormous potential optimizing existing operations within companies and increasing companies' insight into their customers. We start by explaining their relevance in the industry and their benefits. Followed by ML/DL approaches required to solve these problems and challenges in implementing them.

4.1 Use-Case 1: Customer Review Analysis

In today's digital world, companies and customers are adopting technology at a rapid pace. It opens up a new opportunity for companies to expand their business through technology trends. Marketplace, social network, mobile computing, cloud computing, and analytics are some technology trends that have been utilized to attract more customers to products. With a growing number of customers, it is important to understand their satisfaction to the provided services and their interests.

In an E-commerce website, it is often found that the customer conveys what they think and feel about the products that they bought. This is called as opinion or review. It aims to determine the level of customer satisfaction towards the product. Customer satisfaction has been a central construct in marketing research with outcomes of customer satisfaction having positive impacts in organizations. Customer satisfaction may impact future sales of companies as a result of increasing customer loyalty and attracting new customers [75]. And for this reason, companies strive to find out customer or public opinions about their products and services.

With the rapid growth of social media (reviews, blogs, micro-blogs, comments and postings in social network sites), if one is looking for a product or a service, one is no longer limited to asking his friends thanks to a lot of reviews on the web about the product. Therefore, reviews available on the web can act as a guiding tool for companies to understand their customers better. Electronic reviews serve as a good example. There are many reviews of electronic tools (e.g. mobile phone, laptop, a certain computer part, etc.) out there in the Internet. When the majority of the feedback is negative for a certain product, they will not do well in the market until the underlying problem is fixed. On the other hand, having a positive feedback can attract new customers and build their trust on the brand.

Customer review analysis is the generic term to address this problem. By analyzing customers' review, it is possible to detect issues as well as confirm successes of the product and let companies act accordingly. The main goal is to establish a correlation between customer's satisfaction and sales figures. Brand monitoring, competitive market research, and customer service are some examples of its application.

Different platforms offer different options to their customers for writing their reviews. For instance, the customer can provide a numerical rating from 1 to 5 or to write comments about the product or in most cases, both. However, understanding the sentiment behind the reviews is not always a straightforward task. A five-star review might mention aspects the customer did not like, despite liking the experience as a whole. Supposing a fast-food chain customer posted a 4-stars review *"The patty is perfectly grilled and juicy, the bun and the sauces are amplifying the whole burger even more! The pickle is too sour though, I need to throw it away!"*. This is a good example where, identifying the sentiment can be tricky. It has two sentiments in the reviews, one is positive (*The patty is perfectly grilled and juicy, the bun and the sauces are amplifying the whole burger even more!*) and the other is negative (*The pickle is too sour though, I need to throw it away*). The objective of sentiment analysis is to discover all sentiments in the text, weight them, and express it as a positive, negative, or neutral sentiment, or a numeric rating score expressing the intensity of the sentiment.

Analyzing the sentiment of customer reviews refers to the use of natural language processing (NLP), text analysis and computational linguistics to capture the subjectivity in terms of the semantic orientation associated with the constituents of a text [89]. It is fundamentally a classification task, as it tries to make decision based on pre-defined classes (positive or negative sentiment).

The importance of sentiment analysis for brand monitoring is well-known in analytic industry. Many companies that offer analytic services publish white papers [49, 6] on this topic to inform the value of sentiment analysis for their brand. Some competitions also held place in order to find a more optimal way to solve this problem. As an example, Kaggle held a competition where the competitors need to pick out the specific part of a tweet (comment on Twitter), word or phrase, that reflects the sentiment [12].

Sentiment analysis does not only attract industry, but also academia. Due to its potential, many researches are exploring various aspects of it, starting from showing its (novel) application [75, 10], discussing its techniques [32], exploring novel methods [97, 15], and also exploring multilingual capabilities [18, 30, 13].

Challenges

Finding out the sentiment of reviews means to work with natural language text. A text does not have a structure that can be directly computed to extract its information. These information contain list of words or phrases that are instrumental in sentiment

[12] https://www.kaggle.com/c/tweet-sentiment-extraction

analysis. However, using them is far from sufficient to interpret its information. Several issues are [96]:

1. Several words can have both sentiment orientations, depending on the context. For example, "suck" usually indicates negative sentiment, e.g. *"This camera sucks"*, but it can also imply positive sentiment, e.g. *"This vacumm cleaner really sucks"*.
2. A conditional or interrogative sentence may contain sentiment words, but expressing no sentiments. For example, *"Can you tell me which camera is good?"* and *"If I can find a good camera, I will buy it."* Both these sentences contain word "good" that normally expresses positive sentiment, but neither expresses a positive or negative sentiment on any specific camera.
3. Sarcastic sentences are hard to deal with, e.g. *"What a great camera! It stopped working in two days."*
4. Many sentences without sentiment words can also imply opinion. They are objective sentences that express some factual information. The sentence "The dishwasher uses a lot of water." implies negative sentiment about the dishwasher, since it uses a lot of water.
5. Use of acronyms and slangs (e.g. "lol", "wym", "idk", and etc.) or word abbreviations also pose interpretation challenges.

These are just some technical challenges. (Hussein, 2016) [47] divides the challenges into two types, i.e. theoretical and technical. While technical aspect goes around how to deal with words, extracting features and/or keywords. The theoretical challenges consider algorithm overhead, negation, domain dependence, etc.

4.2 Use-Case 2: Ticket Classification

Customers reach out to companies through multiple channels, from emails, live chats, and social media platforms. The advancement in technologies, especially mobile technologies, also making it easier for customers to get in touch at any time of the day. While it is a good thing that customers can reach out companies easier and faster, it brings a concern to the customer support and service department. The number of incident tickets start to pile up.

The rapid shift of companies toward better customer experience and satisfaction has resulted in higher number of incident tickets. This raises the importance of a support ticketing system in order for companies to meet their customers expectations. It has become a strategic element in business competitiveness. Hence, it is crucial for companies to identify slow running processes, perform a root cause analysis, and improve their performance. Addressing incident tickets to appropriate person or unit in the customer support and service department has critical importance in order to provide improved customer satisfaction while ensuring better allotment of support recourses.

In a traditional system, tickets need to be assigned manually to the appropriate person or unit depending on their content. The manual assignment is a straightforward method, but have major drawbacks: it is time consuming and requires human efforts [12]. Requiring human efforts means that it is prone to human errors, especially when incoming tickets never end. Also resource consumption is carried out ineffectively because of the misaddressing. Often times manual assignment increases the response time which result in customer satisfaction deterioration. Multiple-choice systems which provide customers to choose related categories or unit within defined categories has been placed to address this problem. Unfortunately, this does not resolve the problem completely. There are a lot of cases where the tickets are not properly assigned, the communication channel does not provide the options, or customers are not sure which category to choose.

An automated approach such as ML based ticket classification can provide higher quality customer support and boost customer satisfaction. It reduces manual efforts and thus human errors while ensuring high service levels. The assignment of tickets to category is basically a multi-class classification problem where the content of a ticket is first analyzed before it is classified under the given categories. This problem is a widely studied problem in which various NLP algorithms and feature extraction techniques can be used [12, 78, 87].

A lot of companies, such as Zendesk[13], HelpScout[14], Freshdesk[15], and Salesforce[16] keep innovating and researching in how to offer a new and improved way to the ticket classification problem that goes beyond an automated ticket classification and routing. An popular extension solution to this problem includes chatbots.

Challenges

Similar to the Section 4.1 textual data problem exists here as well. Having to classify the textual data given multiple categories only further increases the complexity of the problem. Several issues that are commonly found in ticket classification:

1. Having multiple categories means that words can have multiple meanings that are very different. A good example would be the word "*Apple*". *Apple* can refer to the fruit and also a technology company.
2. It is common to have a lot more data on some popular categories and only few to none data on some unpopular categories.
3. Choosing the appropriate categorical encoding approach can be tricky. Different type and size of category (ordinal or not ordinal, small number or large number of categories) require different encoding methods. For example, label-encoding can unintentionally introduce order in the labels, and one-hot encoding can lead to high memory consumption.

4.3 Use-Case 3: Inventory Demand Forecasting

With the development of consumer goods industry, companies have launched innovative products to meet growing needs and desires of their customers. And to make this growth possible and profitable, it requires them to develop an efficient value and

[13]`https://www.zendesk.com/`
[14]`https://www.helpscout.com/`
[15]`https://freshdesk.com/`
[16]`https://www.salesforce.com/`

supply chains. An efficient management of supply chain ensures highest quality of customer services and striving for minimization of the costs generated by flow between the link.

Inventory management is an important part in supply chain that controls the process of ordering, storing and using a company's inventory. Distributor, wholesaler, manufacturer, or retailer businesses typically have one or multiple warehouses to store their products before delivering them to customers. While overstocking products will increase inventory holding costs, not having enough (stocks out) leads to missing out on potential sales. This is a very important and costly question to ask regarding demand planning strategies. The goal is to reduce inventory level while still being able to fill incoming orders.

Inventory demand forecasting is a process that gives companies insights over future demand so that they can make an informed decisions over their inventory. These forecasts are gathered through a combination of analyzing sales data, looking at trends and patterns in customer purchase behavior, as well as assessing the performance of previous inventory volumes across different demand levels. It can be done qualitatively as well as quantitatively.

Qualitative inventory forecasting predicts demand based on the broad perspective of the industry the business is operating in. Factors such as current industry performance, market trend, regulation changes, technology shifts, and evolving customer demand are assessed rigorously. It includes interpretation of data and business expertise to read the market. With qualitative forecasting, demand is forecast relies on expert knowledge of the marketplace and will always be far more subjective than quantitative methods.

Quantitative inventory forecasting refers to time series forecasting described previously in Section 3.3. It involves analyzing historical data, such as previous sales data and website traffic, to estimate customer demand. Looking at sales history through different timelines can help companies to get an accurate estimate of inventory levels. It takes factors such as demand trends and seasonality into account to look for patterns in the data. Thus, quantitative forecasting relies on having sufficient, good quality data to make a reasonable assessment. And when data is not available, such as for new businesses or products, this approach is impractical.

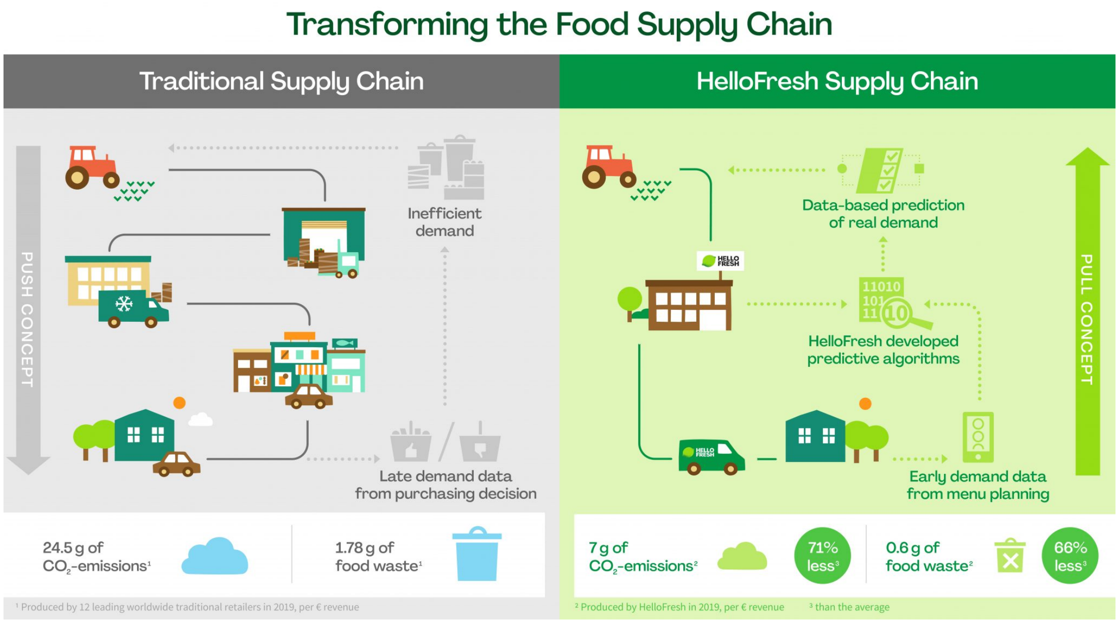

Transforming the Food Supply Chain
Traditional Supply Chain
HelloFresh Supply Chain
PUSH CONCEPT
Inefficient demand
Late demand data from purchasing decision
24.5 g of CO₂-emissions¹
1.78 g of food waste¹
¹ Produced by 12 leading worldwide traditional retailers in 2019, per € revenue
Data-based prediction of real demand
HelloFresh developed predictive algorithms
Early demand data from menu planning
PULL CONCEPT
7 g of CO₂-emissions²
71% less³
0.6 g of food waste²
66% less³
² Produced by HelloFresh in 2019, per € revenue
³ than the average

Figure 4.1: HelloFresh Food Supply Chain [4]

A successful inventory demand forecasting can effectively reduce inventory holding and waste costs which leads to higher profit. HelloFresh[17] serves as a good example, where a company have managed to developed a successful inventory demand forecasting to its supply chain operation. HelloFresh's business model is to prepare the ingredients needed for a meal, and deliver them to customers, who must then cook the meal using recipe cards. To be able to deliver fresh ingredients every time while not overstocking them, HelloFresh optimizes supply chain operation in various aspects. Figure 4.1 illustrates how HelloFresh uses customer data to predict order volume with high accuracy. This consumer demand-driven pull model brings a new level of sophistication and efficiency to the supply chain and leads to significant food waste reductions[3, 1, 2].

Challenges

Forecasting is a challenging task that involves many inconsistent factors. However, a successful forecasting helps improve cost effectiveness and availability in the supply chain. Several issues that are commonly found in inventory demand forecasting:

1. Inaccurate stock levels: Often times the stock level data is stored inaccurately. Inventory demand forecasting becomes much more difficult when actual stock levels are hard to monitor. The inaccuracy of stock levels can quickly build up and heavily affect the forecasting result.

2. Availability of data: Forecasting makes prediction based on historical data. When there are no data on the product, e.g. new product introduction, predicting the correct level of stocks will be much harder. In this case, data from similar product or older products can be utilized to simulate the forecasting as a base line.

[17]`https://www.hellofresh.de`

3. Signal vs. Noise: An abnormal shift in demand can happen unpredictably which if not explained can be treated as a signal in the forecasting models. Establishing the right cause and effect relationships between abnormal shifts in demand that may not repeat itself until the same conditions are met, is critical to interpret the trends appropriately and driving correct strategies to accommodate the change.

5 Experiments

Three use-cases are introduced in the Section 4 and the theories to approach the use-cases are explained in Section 3. This chapter elaborates two important parts of this thesis. First is the data generation through PDGF and second the ML pipeline. For each use-case, we explain how we structured our PDGF XML configuration files. The idea is to have a dataset that reflects the data that can be found in the industry. Using PDGF can we not only achieve this, but we are also able to generate the data at a chosen scale factor (SF). Furthermore, we describe the ML pipeline for all use-cases.

5.1 Use-Case 1: Customer Review Analysis

The first use-case (UC1) is a customer review analysis, which falls under the category of classification task. Customer interaction through review is often found in business that provide an online platform. UC1 is based on an E-commerce scenario, where customers submit reviews for their purchased products. In UC1, we want to look into the sentiment of each product's review to determine customer's reaction to the product. The goal is to eventually build a system that can differentiate positive and negative review.

Data Generation

The data is based on Amazon review dataset[18] [69] that includes various information such as product name, product category, rating, and review. We also supplement the data with another Amazon review dataset[19] [33] that in addition to regular review contains irony and sarcasm reviews. Irony dataset introduces noises into the dataset, as good words do not always translate to good review. The dataset contains 9 categories and 79 classes in total. Appendix A lists all classes and categories.

For our purpose, we will generate product review data based on some assumptions: each customer order 7 products and in average 20% of the total order will be reviewed. Mathematically, we will generate *#CUSTOMER* × *#ORDER_PER_CUSTOMER* × *#REVIEW_RATE*. The number of customer will grow depending on the scale factor times 10000. All these variables can be adjusted in the schema XML file.

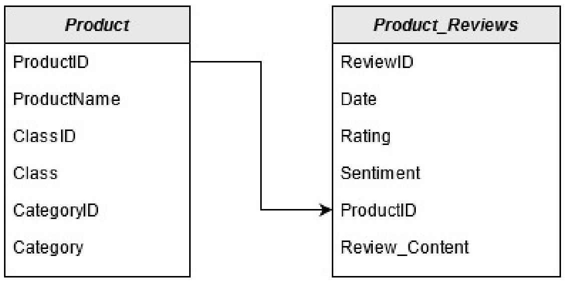


Figure 5.1: Product Review Schema

[18]`https://nijianmo.github.io/amazon/index.html`
[19]`https://github.com/ef2020/SarcasmAmazonReviewsCorpus`

Figure 5.1 illustrates the product dataset schema for UC1. The *Product* table contains all the product metadata information (name, class, and category). And *Product_Reviews* table contains all the reviews and ratings. The sentiment is the attribute that we want to predict.

PDGF offers a built-in text generator module that uses Markov chain approach. It creates a finite set of states with fixed conditional probabilities of jumping from a given state to another. To use this in PDGF, we need to feed the module with a dictionary that contains transition matrix for the Markov chain. The required dictionary can be generated through PDGF as well. To be able to generate our own *Product_Reviews*, first we create dictionaries of product reviews taken from Amazon review dataset.

We utilize two datasets, Amazon regular review dataset and Amazon irony and sarcasm dataset, to create the dictionaries. First, we split all products in both datasets based on its category, class, and rating. For each category, class, and rating combination, it outputs a csv file containing all of the reviews. These csv files are the inputs for the PDGF's Markov Chain builder that generates us the binary files. These binary files are the *dictionaries* for the product reviews that contain the transition matrix for the Markov Chain text generator.

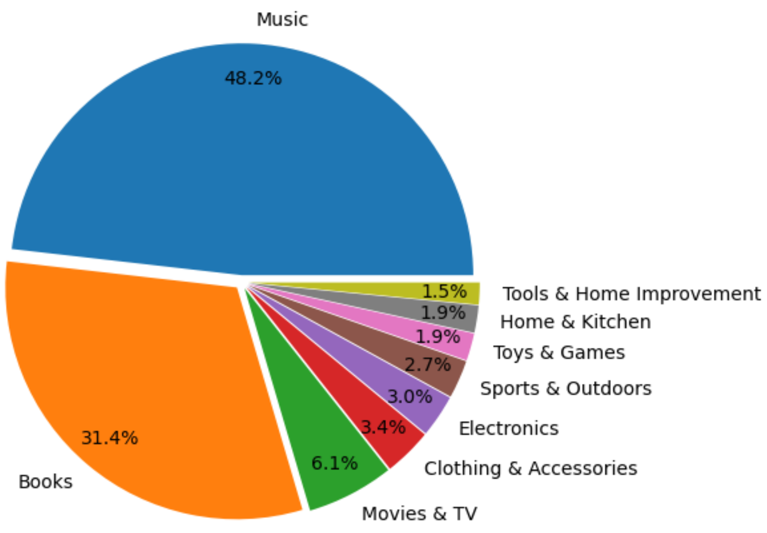
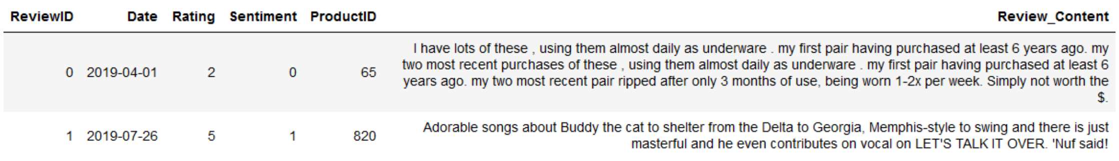


Figure 5.2: Example of the Generated Product Category Distribution

We also introduce a distribution of the products in the dataset. Figure 5.2 illustrates an example of the distribution of product categories that we generated based on the Amazon dataset. We prepare the weight list for each category, class, and rating combination similar to what Amazon review dataset has. The *Product* table is constructed based on this weight list. The *Product_Reviews* table uses the *Product* table to generate an entry with a product from the *Product* table. The review is generated using the dictionary based on the class, category, and rating of the product. Figure 5.3 shows an example of the generated reviews in the *Product_Reviews* table.



ReviewID	Date	Rating	Sentiment	ProductID	Review_Content
0	2019-04-01	2	0	65	I have lots of these , using them almost daily as underware . my first pair having purchased at least 6 years ago. my two most recent purchases of these , using them almost daily as underware . my first pair having purchased at least 6 years ago. my two most recent pair ripped after only 3 months of use, being worn 1-2x per week. Simply not worth the $.
1	2019-07-26	5	1	820	Adorable songs about Buddy the cat to shelter from the Delta to Georgia, Memphis-style to swing and there is just masterful and he even contributes on vocal on LET'S TALK IT OVER. 'Nuf said!

Figure 5.3: Example of the Generated Review

Pre-processing

Pre-processing step aims to clean the dataset and extract relevant information. It is important to clean the review data from noise and duplicate words that can affect the prediction result. In order to clean the data, we run several transformations:

- Lowering the text
- Removing all break lines and multiple spaces
- Stemming and lemmatizing all words
- Removing all punctuations and numbers
- Removing stop words, such as 'the', 'and', 'or', etc.
- Removing single character that left after removing stop words
- Tokenizing the text

Although the goal is to use the review to determine the sentiment, it is required that we have the sentiments in the training data. However, the Amazon dataset does not give us the sentiment. We use the rating to determine the sentiment of the training data. Ratings that are more than 3 is considered positive (1) and anything else is negative (0). This serves as the label in our classifier.

Feature Engineering

After cleaning the review data, we now have tokens to represent the reviews in every row. Using this token, we can now create a bag of words model and also add the TF-IDF values for every token creating a numerical feature vector. This feature vector serves as the model's input.

Model Training and Evaluation

Naïve Bayes classifier has proven effective in many practical applications. As explained in 3.1.3, it competes well with more sophisticated models despite its simplicity. The goal is to build a system that can predict the sentiment of a product through its review. For the training, we fed the model with the review data along with its corresponding sentiment as the label. The output is an array of label predictions.

To evaluate the quality of the classification, we use F1-Score, since it takes not only accuracy, but both precision and recall of the classification into the consideration. We deem that a model is qualified enough, if it achieves an F1-Score of at least 85%. It achieves a similar performance metric based on some other sentiment analysis examples explained in Kaggle submission or blogposts [62, 64, 58].

5.2 Use-Case 2: Ticket Classification

The second use-case (UC2) is ticket classification, which falls under the category of classification task. This use-case is inspired by a Kaggle competition [20] that use a financial complaint dataset. A lot of complaints that are submitted online via a platform or emails often come uncategorized. An automation that can automatically assign a category to the complaint can improve the process of answering these complaints tremendously. The goal of UC2 is to build a classification system that can automatically categorize complaints.

Data Generation

The data is based on Consumer Financial Protection Bureau (CFPB) dataset[21]. It is a collection of complaints about consumer financial products and services. Figure 5.4 illustrates the complaint dataset schema for UC2. The *Complaint* table contains *ID*, *date*, *complaint*, and *category*. The *complaint* column will be our feature column, where we extract the information. And the *category* column is the attribute that we want to predict for the new data. For our purpose, we will generate complaint data that is equivalent to *#CUSTOMER* $\times$ *#COMPLAINT*. The number of customer will grow depending on the scale factor times 10000 and we assume that every customer makes 2 complaints.

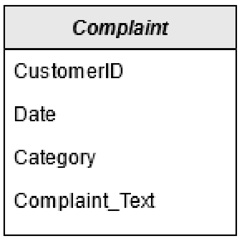


Figure 5.4: Complaint Schema

The original dataset contains 18 categories. However, we find that some of these categories overlap with each other. From Figure 5.5, we can observe that the overlapping categories in the original dataset. For example, there is a category for 'Payday loan' even though there is also a category for 'Payday loan, title loan, or personal loan'. After combining the overlapping categories, we are left with 12 categories as illustrated in the right table of Figure 5.5.

[20] `https://www.kaggle.com/selener/consumer-complaint-database`

[21] `https://www.consumerfinance.gov/data-research/consumer-complaints/`

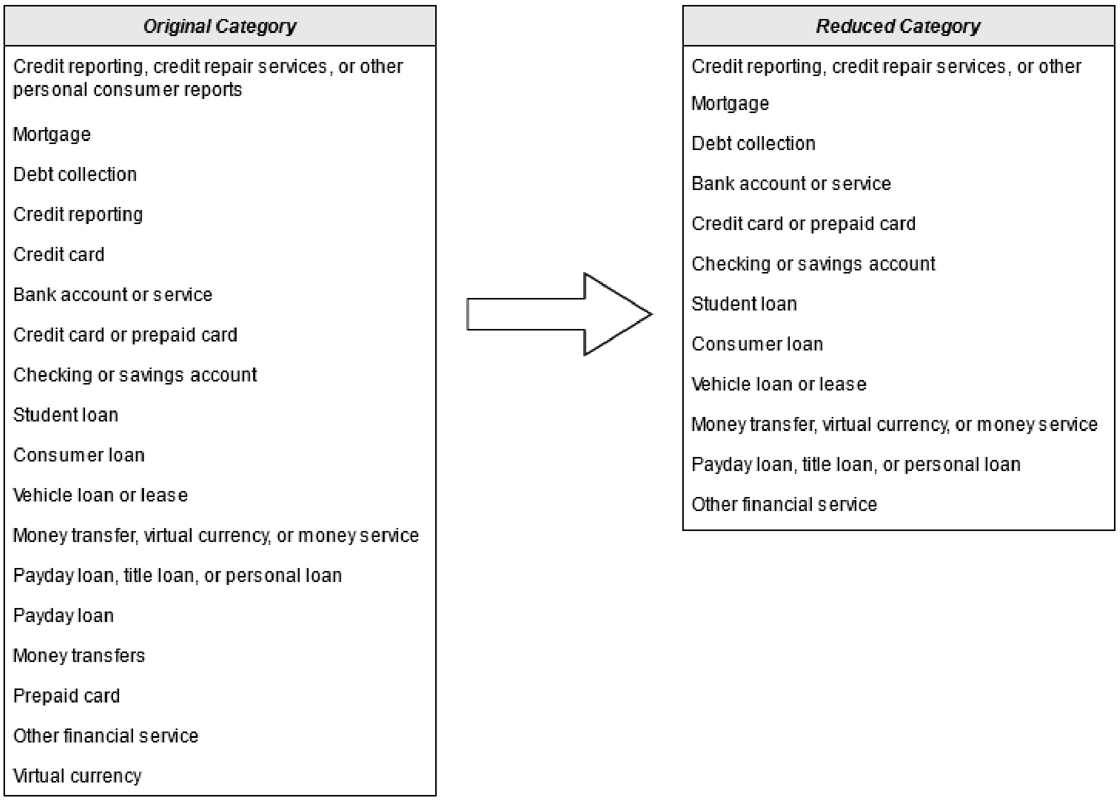

Original Category
Credit reporting, credit repair services, or other personal consumer reports
Mortgage
Debt collection
Credit reporting
Credit card
Bank account or service
Credit card or prepaid card
Checking or savings account
Student loan
Consumer loan
Vehicle loan or lease
Money transfer, virtual currency, or money service
Payday loan, title loan, or personal loan
Payday loan
Money transfers
Prepaid card
Other financial service
Virtual currency
Reduced Category
Credit reporting, credit repair services, or other
Mortgage
Debt collection
Bank account or service
Credit card or prepaid card
Checking or savings account
Student loan
Consumer loan
Vehicle loan or lease
Money transfer, virtual currency, or money service
Payday loan, title loan, or personal loan
Other financial service

Figure 5.5: Category Transformation

After cleaning the category of the data, we split the data based on its category. We create a csv file for each category, containing all the data for that particular category. These csv files are then fed to the PDGF's Markov Chain builder. It returns us binary files of each category to work with. These binary files are the *dictionary* that contain the transition matrix for the Markov Chain text generator.

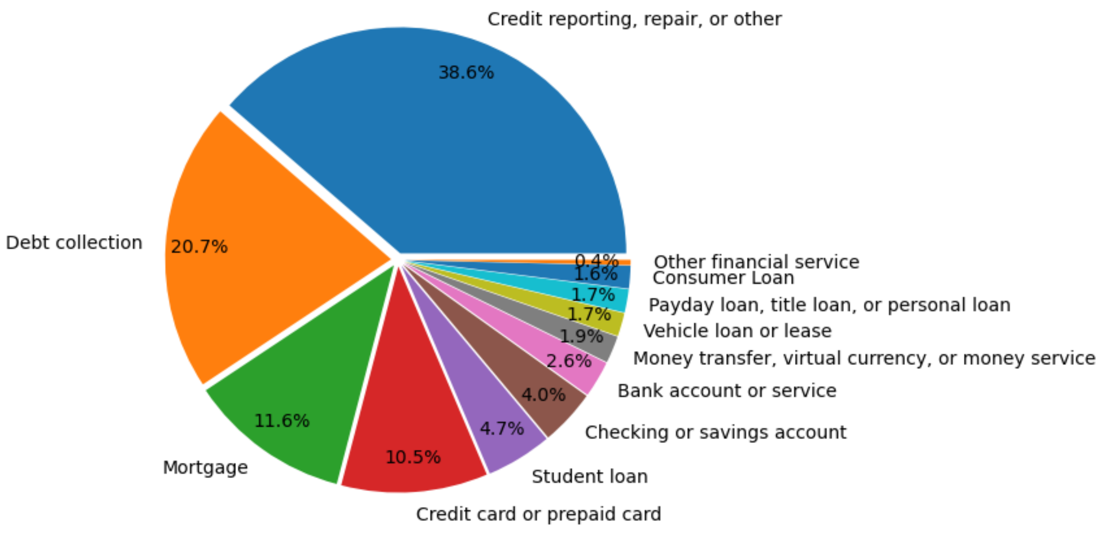
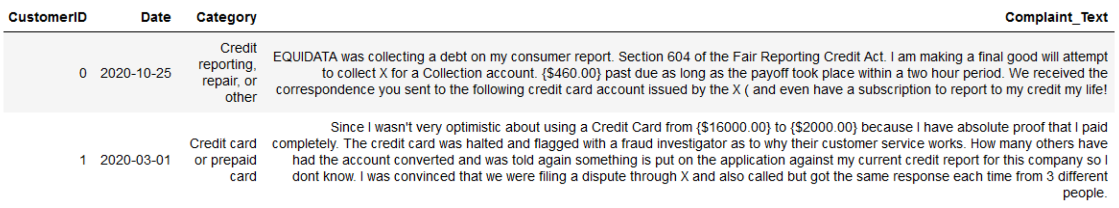


Figure 5.6: Example of the Generated Complaint Category Distribution

Not only the complaint data and its category, but we also take the distribution from the original dataset. We set it as the weight to generate our dataset. The complaint data distribution is constructed based on this weight list and it will always generate similar distribution every time. Figure 5.6 shows an example of the distribution of complaint categories that is generated by PDGF. Also, Figure 5.7 shows an example of the generated complaints.



CustomerID	Date	Category	Complaint_Text
0	2020-10-25	Credit reporting, repair, or other	EQUIDATA was collecting a debt on my consumer report. Section 604 of the Fair Reporting Credit Act. I am making a final good will attempt to collect X for a Collection account. {$460.00} past due as long as the payoff took place within a two hour period. We received the correspondence you sent to the following credit card account issued by the X (and even have a subscription to report to my credit my life!
1	2020-03-01	Credit card or prepaid card	Since I wasn't very optimistic about using a Credit Card from {$16000.00} to {$2000.00} because I have absolute proof that I paid completely. The credit card was halted and flagged with a fraud investigator as to why their customer service works. How many others have had the account converted and was told again something is put on the application against my current credit report for this company so I dont know. I was convinced that we were filing a dispute through X and also called but got the same response each time from 3 different people.

Figure 5.7: Example of the Generated Complaint

Pre-processing

The goal of the pre-processing is to remove noises from the data so that we can extract only important information. Upon inspecting the data, we find that we often have URL links and masking words. Additionally, we find that numbers are important in the dataset, as it can lead to some categories. In order to clean the data, we run several transformations:

- Lowering the text
- Removing all break lines and multiple spaces
- Stemming and lemmatizing all words
- Removing all punctuations
- Removing stop words, such as 'the', 'and', 'or', etc.
- Removing single character that left after removing stop words
- Tokenizing the text

Model Training and Evaluation

As previously explained in Section 3, LSTM is designed to retain information for long periods of time. Due to its structure, it can encode long-distance word dependencies effectively. For this reason, we build our ticket classification system based on Bi-LSTM model. It has an additional advantage in compare to the traditional LSTM, because Bi-LSTM has access to both preceding and subsequent contextual information. It effectively increases the amount of information available to the network.

Our Bi-LSTM model is depicted in Figure 5.8. We follow an example provided from a blog post ProgrammerSought [77]. The model start with an embedding layer to represent each words. Unfortunately, due to the memory limitation we need to limit our vocabulary size. We choose a vocabulary size of 50000 to represent the dataset. It then fed to the embedding layer as the input. This provides a dense vector representation of words and documents. The feature vector is then fed to the Bi-LSTM layer, where it is propagated forwards and backwards through the LSTM layer and then concatenates the output. It is then followed by a dropout layer which is responsible regularizing our model to avoid overfitting. It randomly "drop out" or omit units during the training process ~~the training process.~~ We then fit it to a dense neural network with *softmax* activation and categorical cross entropy loss function to do classification.

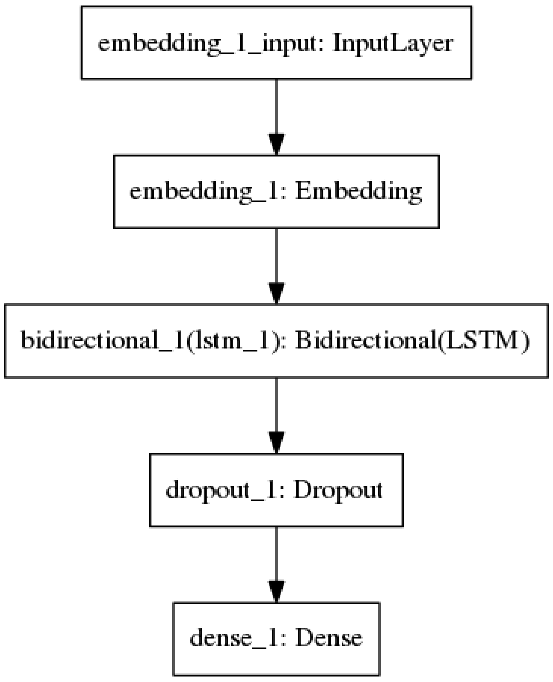

embedding_1_input: InputLayer
embedding_1: Embedding
bidirectional_1(lstm_1): Bidirectional(LSTM)
dropout_1: Dropout
dense_1: Dense

Figure 5.8: Bi-LSTM Architecture

To evaluate the quality of the classification, we use the standard accuracy metric. In addition, we also observe the loss value, as we want to minimize the error for each training during the learning process. We deem that a model is qualified enough if it achieves an accuracy of at least 80%. It achieves a similar performance metric based on some other ticket classification examples explained in Kaggle submission or blogposts [54, 82, 9].

5.3 Use-Case 3: Inventory Demand Forecasting

The third use-case (UC3) is inventory demand forecasting. This use-case is inspired by a Kaggle competition [51], where the goal is to forecast demands of products from multiple stores over a certain period of time. A good inventory planning strategy can significantly reduce inventory costs for businesses, who need to store their products. It will reduce operational cost and lead to a higher margin of profit for the business. UC3 is based on a scenario of a retail chain that sells multiple products and have multiple stores. The goal is to build a system that can forecast sales of products on each stores in the following year. This will give the business a better insight to keep its inventory in an appropriate level.

Data Generation

The data is inspired by a Kaggle competition, where we have the data of previous sales for each product and store. Figure 5.9 illustrates the sales schema for UC3. The data contains all product-store combination with its daily sales for 4 years. The first 3 years will be used as training/historical data, and the last year as validation data.

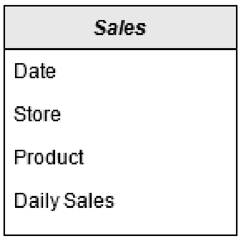


Figure 5.9: Sales Schema

The size of data that will be generated depends on the number of stores and products. While the period of time will be fixed, the number of stores and products will grow as we increase our scale factor in our experiment. In order to create a sales dataset that is not static and not completely random, there are some variables involved in creating the data. Figure 5.10 shows the 2 tables needed for us to create sales data that not only has variance but also seasonality and trend.

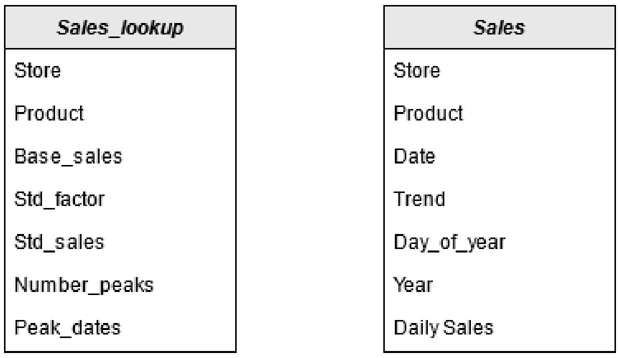


Figure 5.10: Sales Generation Tables

It starts by creating a lookup table that contains all variables needed to generated the data. First, we create all store-product combinations based on out scale factor. This gives us *#STORE* × *#PRODUCT* rows on the *Sales_lookup* table. Each combination has some variables that help us to simulate sales data that have different sales for each combination, some peak values, a seasonality, and a trend. These values are adjustable and chosen arbitrarily for our purpose. The *Sales_lookup* table contains following information:

- *Store*
  It serves as the Store ID. The distinct number of the generated ID is based on the scale factor. It is formulated as $\sqrt{\text{SF}+2}$.
- *Product*
  It serves as the Product ID. The distinct number of the generated ID is based on

the scale factor. It is formulated as $(\log(\text{SF}) + 1) * 100$.

- *Base_sales*
  It serves as the base sale number of a product in a certain store. It randomly generate a sale number from 3 to 10.
- *Std_factor*
  The standard deviation factor is used to create the non-static nature of the sales number and used as the multiplier to create the standard deviation sale variable. It randomly generates a standard deviation factor from 0.3 to 1.0.
- *Std_sales*
  Standard deviation sale is based on standard deviation factor times the base sale, rounded down. It will be used in the final equation to create final sale numbers for each combination that is not stationary.
- *Number_peaks*
  It generates number of peak dates a data should have in a year. The peaks will create a jittery effect in the time series data, allowing us to have a more realistic data. Each combination will have different number of peak dates, that ranges from 0 to 150 days in a year.
- *Peak_dates*
  It takes the information from *Number_peaks* to generate the corresponding number of random dates, when the peak should happen in the year.

After the lookup table is generated, we can then start creating our actual sale dataset based on it. Each store-product combination consists of 4 years worth of data. Figure 5.11 illustrates examples of the generated store-product data in the span of 4 years. It demonstrates that the PDGF will create a non-stationary time series data with an upward trend that are different for each combination based on our configurable schema.

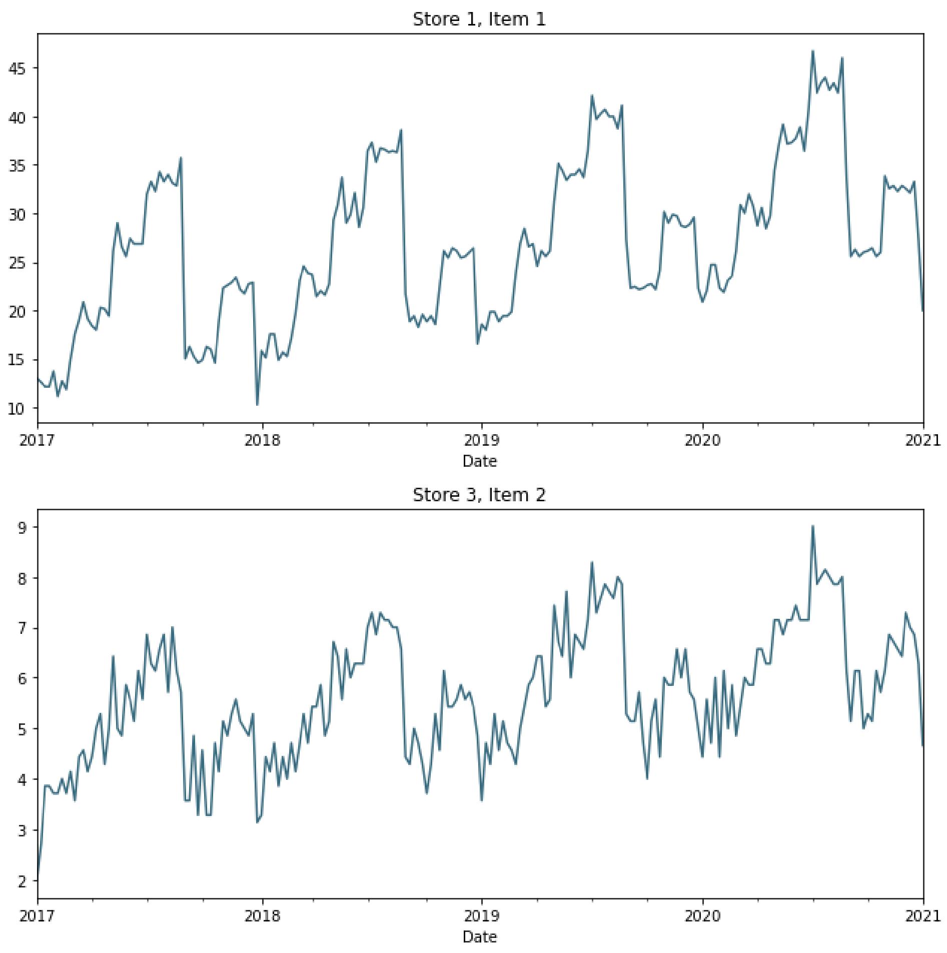


Figure 5.11: Example of the Generated Store-Product Time Series

Although the only important columns in the *Sales* table that matters for the benchmark are *Store*, *Product*, *Date*, and *Daily_Sales*. The *Sales* table contains several other variables needed to create the *Daily_Sales* for our purpose:

- *Store*
  The Store ID, taken from *Sales_lookup* table.
- *Product*
  The Product ID, taken from *Sales_lookup* table.
- *Date*
  It generates the date of the data based on *Year* and *Day_of_year* columns. For example, *Day_of_year* 1 and *Year* 2017 produces 2017-01-01, *Day_of_year* 32 and *Year* 2018 produces 2018-02-01, and so on.
- *Day_of_year*
  It generates the day number of the year for each combination in each year.
- *Year*
  It generates the year of the data.
- *Trend*
  It serves as an indicator, when the direction of the seasonality should change. For

our purpose, we will change the seasonality of the data every 60 days (roughly 2 months) with either an upward or downward trend movement. It results in an indicator value that range between 0 to 6.

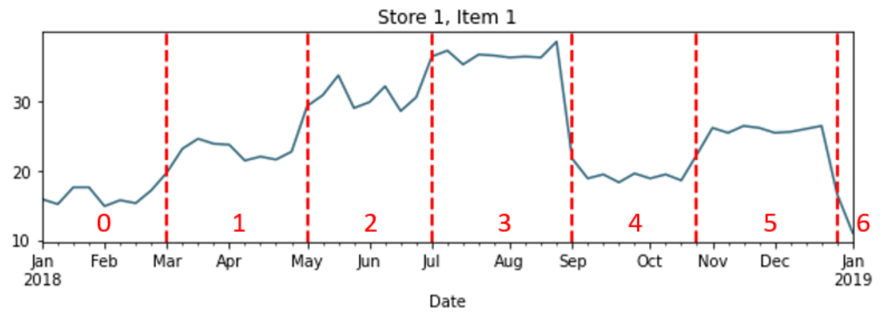


Figure 5.12: Trend segmentation

The red lines in Figure 5.12 illustrates the 60 days range in a year. There are 7 sections, representing the range of value from 0 to 6. Figure 5.12 shows an example, where the data has an upward trend in the most part of the year (index 0 to 3). It is then followed by a downward trend until the end of the year (index 4 to 6).

- *Daily_Sales*
  It shows the number of sales on a particular date. It utilizes *Base_sales*, *Std_sales*, *Year*, and *Trend* to generate a sale number.

  The *Daily_Sales* is differentiated into 2 types, namely, normal day and peak day. They will have a different min-max range required to generate the random sale value. The peak days of a store-product combination are flagged in *Peak_dates* column on *Sales_lookup* table.

  The *Daily_Sales* is defined as:

  $$b + (d * (T + Y/2)) \tag{22}$$

  where:

  b : Base sale
  d : Deviation of the sale
  T : Trend multiplier
  Y : Year number

  The T component utilizes the *Trend* column t to decide the range of the value. As it will be multiplied with the deviation of sale and added to the base sale, the range of the sale will be higher along with the time. To create a seasonality and trend effect, we play around with the multiplier so that for certain periods, the summand to the base sale is lower than in the other periods, creating a downward trend behavior.

  Table 5.1 shows the equations to generate the min-max range on normal and peak days. For our purpose, we decide to generally have an upward trend and a downward trend in the last few months of the year. t value range from 0 to

6. When t value is bigger than 3, we use a modulo operation, resulting in a smaller multiplier. Otherwise, the maximum value will have 1 point higher than the minimum value. These calculated range of value will be then used by PDGF to generate a random value at the end.

Normal Value	
Min	**Max**
(t <= 3) ? t : t % 3	(t <= 3) ? t+1 : t % 3

(a) Trend Component for Normal Day

Peak Value	
Min	**Max**
(t <= 3) ? t+1 : t % 3	(t <= 3) ? t+2 : t % 3

(b) Trend Component for Peak Day

Table 5.1: T component calculation

While the T component introduces a seasonality and trend effect to the data, it only consider a year worth of data as it does not include the year component in its calculation. The Y component ensures that there is an upward trend as time passes. By adding Y to the T, the higher the year number, the higher the multiplier. We divide the Y by 2, to lower the range of the data.

Pre-processing

As the time of the writing, Prophet only supports univariate time series. Although each store-product combination behaves differently, we assume that there is no correlation between products. To make our dataset into a univariate problem, we need to train each store-product combination separately.

We start by creating a key column, consists of a combination between store ID and product ID. Prophet model takes only 2 columns with a pre-determined names. The date column should be named as *ds* and the value column should be named as y. We rename the both columns to the required naming convention before sending the data to the training phase.

Model Training and Evaluation

One of the key feature of the Prophet model is its simplicity. It does not require a lot of feature engineering or parameter tuning for the model to run well. The goal is to build a system that can predict the number of sales of the following year for each store-product combination, using 3 years worth of data. The output can then be used to create a good inventory planning strategy.

To evaluate the quality of the forecasting, we compare our data for the last year with the forecasting result using Mean Absolute Percentage Error (MAPE) as the metric to determine the error rate. We choose to use MAPE as the forecasting metric following the Kaggle competition, where UC3 is inspired on [51].

6 Evaluation

In this section, we present results of an experimental evaluation of the three use-cases introduced in Section 5. Following the evaluation methodology described in ADABench [79]. We prepare three dimensions for the evaluations of the implementations, namely hardware, software, and problem size.

As discussed in Section 2, the main performance metric is throughput in samples per second while defining quality metric threshold for each use-cases as discussed in Section 5.

6.1 System Environment

Hardware is one of the dimension that we want to explore in this thesis. Two different compute clusters of following specifications are provided:

Property	**Big Node**	**Small Node**
CPU	2x Intel Xeon E5-2650 v2 @2.6 GHz	2x AMD Opteron 6376 @2.3 GHz
Core	32	32
Memory	128 GB	64 GB
GPU	Nvidia Tesla K40M 12GB DDR5	Nvidia GeForce GTX 980 4GB DDR5

Table 6.1: Cluster specification

Table 6.1 lists down the specification of each cluster, where we run our experiments. The GPU is used where it is applicable, such as by UC2. As Big Node provides a better specification, it is expected that workloads that run on Big Node perform better than Small node.

All use-cases are implemented in two different flavors, namely, Python and Spark. They are chosen due to their wide-spread use in the field of data processing and data science. Python implementations are based on *pandas*, *numpy*, *scikit-learn*, and *tensorflow*. These are prominent libraries often used in ML pipelines. Spark is designed for large scale data processing. It allows us to write in different languages, such as Scala and Python. While Python implementations are a single core implementation, Spark implementations utilizes multiple cores to run a number of process in parallel by representing the data as RDDs.

The scale factor determines the file size for each dataset, with bigger dataset size means higher amount of data. All use-cases generate different amount of data as explained in Section 5. Table 6.2 illustrates the size range of the data generated by PDGF for each dataset.

Scale Factor	Estimated Dataset Size
1	10 – 50 MB
5	50 – 100 MB
10	100 – 500 MB
50	0.5 – 1 GB
100	1 – 5 GB
500	5 GB+

Table 6.2: Estimated Dataset Size

Three synthetic datasets are generated for each run: training, serving, and scoring datasets. Training dataset is used to train the model and it contains all dataset features and label. The serving dataset is used to simulate the inference on a typical ML pipeline and to measure the throughput. And the scoring dataset is used to measure model quality based on each individual quality metric.

6.2 Results

As discussed in Section 2.1, throughput in samples per second is the main performance measure evaluated in ADABench. It is then followed by a combined benchmark metric *ADASps@SF*. ADABench describes this metric as a geometric mean of all use case metrics, where each use-case's metric is the geometric mean across the training and serving throughput. Using this metric, we can observe the performance on different scale factors closely.

Throughput measured for the training and serving stage at different scale factors. The Python-based implementation is shown in blue and red when run on a small or big node respectively. Analogous the Spark-based implementations are shown in yellow and green.

Use-Case 1: Customer Review Analysis

In Figure 6.1, we can observe that in general Spark achieves a higher throughput. As the size of the data grows, the advantages of using Spark becomes apparent. UC1 pre-processes the text by removing noises, stemming, and tokenizing to create a feature vector before training the model. Spark parallelization capability speeds up the pre-processing and training significantly as the dataset size grows. It results in a higher throughput for both training and serving stages for Spark implementation. On the other hand, Python implementation runs on a single core sequentially. Because of the sequential run, there is no significant throughput difference in the performance between running the Python implementation on a big node and small node.

At scale factor 1, we can observe that the Python implementation has a higher output. This happens because the size of data is small enough, that the overhead of starting spark jobs affects the runtime. At higher scale factor, this overhead becomes negligible as we process and train the data in parallel, allowing us to finish the pipeline faster.

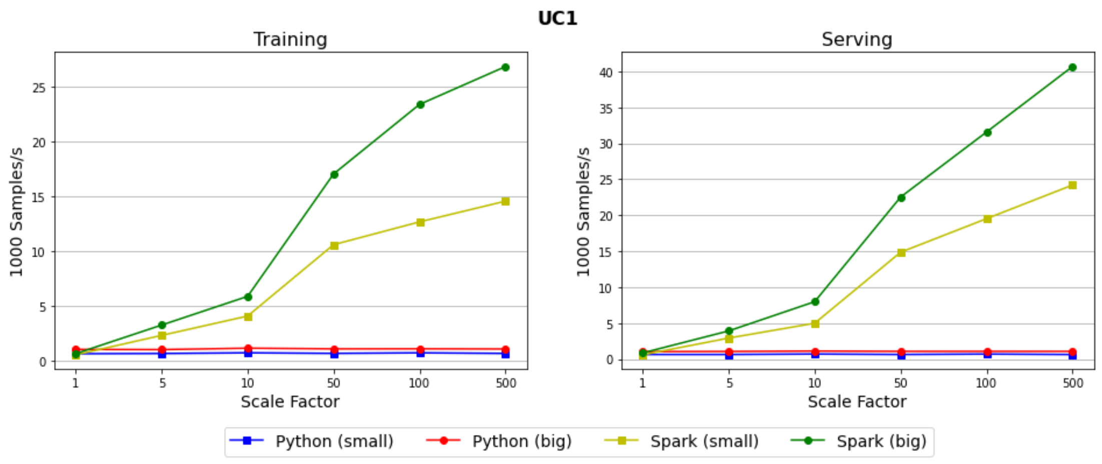


Figure 6.1: Throughput measurement for UC1

UC1 uses a Naïve Bayes solution. Although it is one of the simplest probabilistic classifier model. In practice it often competes well with more sophisticated models [83]. Both Python and Spark implementations make a single pass to the training data and pre-process it before feeding it to the model for training. Table 6.3 shows the quality metric that are achieved at each scale factor. At higher scale factor, we can observe an increase in the f1-score. Although, both implementations have the same pre-processing steps and are trained using the same model, Spark achieves higher metric and stops increasing at scale factor 50.

Scale Factor	**Python (small)**	**Python (big)**	**Spark (small)**	**Spark (big)**
1	0.871	0.872	0.946	0.946
5	0.935	0.935	0.971	0.971
10	0.944	0.944	0.973	0.973
50	0.955	0.955	0.976	0.976
100	0.957	0.957	0.976	0.976
500	0.958	0.959	0.976	0.976

Table 6.3: UC1 Quality Metric (F1-Score)

Use-Case 2: Ticket Classification

Our Bi-LSTM model is built using tensorflow, as it allows us to train on the GPU. However, as we tried to run our Spark-tensorflow implementation, we notice that tensorflow is not designed to take Spark RDDs as input. Unfortunately, we ran out of time to fix this issue and opted out to do the data cleaning with Spark RDD approach and text tokenizing with Python pandas, followed by training in Python tensorflow for the Spark implementation.

The throughput for UC2, illustrated in Figure 6.2, shows no significant difference in training throughput across all configurations. We attribute this to the fact that all trainings are done through a Python approach. In the serving, although Spark implementation has higher throughput, the difference is considerably small if we compare it to the performance difference in UC1. This behavior occurs due to the fact that we still run our tokenizer with Python.

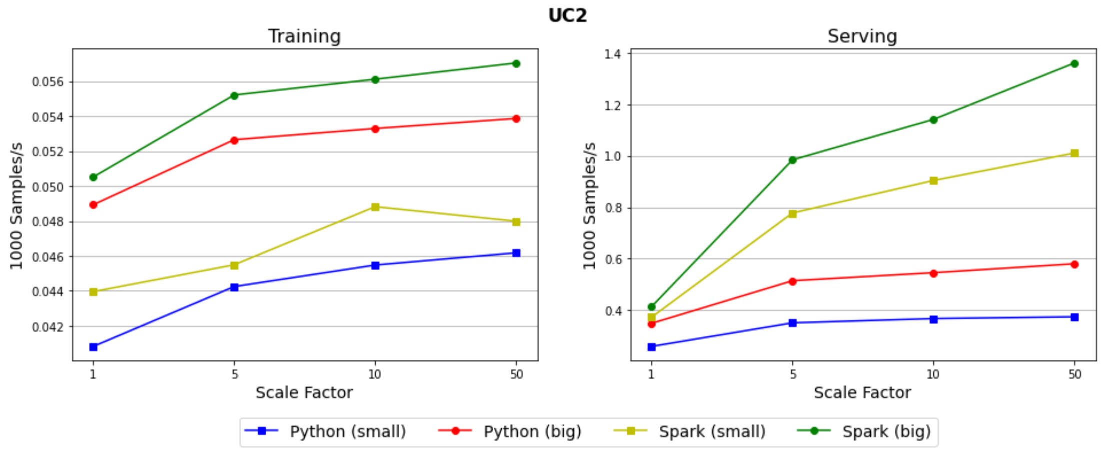


Figure 6.2: Throughput measurement for UC2

We train our Bi-LSTM model on 2 different GPUs that have different GPU memory size. To maximize the GPU utilization, we set the GPU memory allocation for the training according to the respective GPU's specification. However, due to the different GPU memory size, our experiments on big and small node differs in the batch size. The values of batch size on the big node and small node are 512 and 128 respectively. The difference in batch size means different gradients are calculated in each step. Lower batch size increases the training time, but increases the training iterations. This leads to the higher accuracy value that we get on the small node [81, 44, 46].

Scale Factor	Python (small)	Python (big)	Spark (small)	Spark (big)
1	0.798	0.746	0.681	0.739
5	0.910	0.866	0.905	0.860
10	0.932	0.906	0.955	0.916
50	0.987	0.982	0.989	0.985

Table 6.4: UC2 Quality Metric (Accuracy)

Use-Case 3: Inventory Demand Forecasting

We utilize Facebook Prophet forecasting model for UC3. Unfortunately, there is limited support for Prophet in Spark at the time this thesis is written. Facebook Prophet requires a two-column pandas data frame as input. This affects our pipeline, because now we need to transform our Spark data frame back to pandas before training them. Not only transforming the data frame to pandas is time consuming, but it is also computationally expensive and limited to the available memory on the system.

We also encountered an issue in our data generation for the validation part that we do not manage to fix. UC3 is designed to train its forecasting model from 3 years worth of data any predict the fourth year as explained in Section 5.3. We generate 3 years of data from 2017 for each store product combination. For the validation part, we then generate the data for the fourth year for each combination. The sale values that we generated in this part do not reflect the values that are supposed to be created by PDGF. Due to this problem, our forecasting models have a high MAPE score. Although our model does not fulfill the quality metric, we believe that it will still have an equivalent throughput if the quality metric were fulfilled. It is because the quality metric scoring is done through calculating an actual and predicted value. A better quality metric means that we have a predicted value that is close to actual value. Having a bad predicted value does not affect throughput at all at this phase.

If we look into ~~the~~ Figure 6.3, we can observe that all configurations follow a same pattern. We attribute this to the sub-optimal implementation on Spark. As there is minimal pre-processing involved and Spark needs to transform the data frame back to Python pandas data frame, all configurations behave similarly. The serving stage has a lower throughput due to the nature of our forecasting. We try to forecast 1 year worth of data, meaning that for each store-product combination, we try to predict 365 values. Prophet also provides a parameter *uncertainty_samples* for its prediction. It determines the number of Monte Carlo simulations done to estimate future trend uncertainty. While low *uncertainty_samples* value can speed up the forecast, ~~ti~~ will introduces more variance in the forecast data.

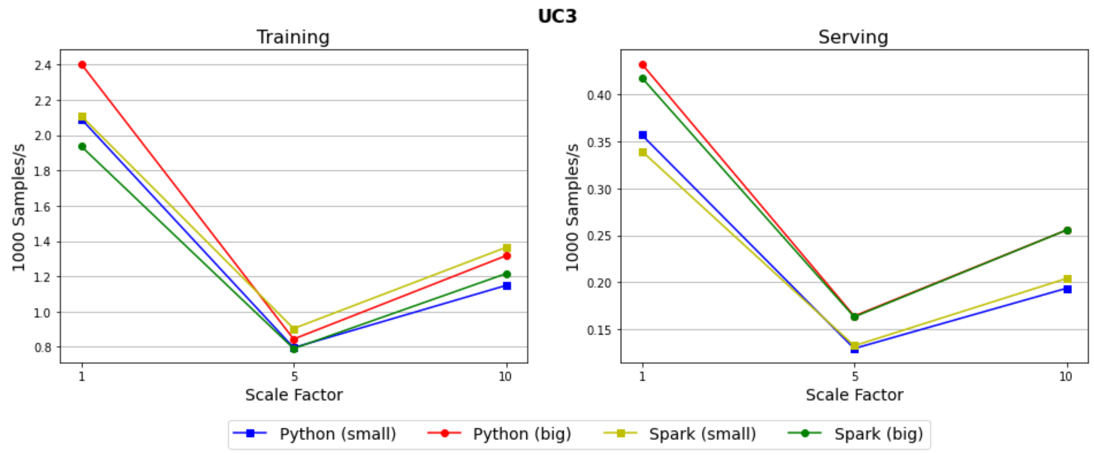


Figure 6.3: Throughput measurement for UC3

Facebook Prophet only supports univariate time series. This means, we need to train all store-product combinations separately. As the number of the combination grows, it becomes an incredibly time consuming to perform as sequential operation. In compare to other throughput measurements, Figure 6.3 shows a lower throughput at higher scale factor. This occurs due to the fact that our forecasting implementation trains all combinations sequentially. We believe that a Spark-based implementation that can utilize its parallelization aspect will improve the throughput significantly.

ADABench Metric

The main goal of ADABench is to compare different ML system for complete pipelines. It provides a combined benchmark metric: *ADASps@SF*. This metric is given by the geometric mean of all use-case metrics, where each use case's metric is the geometric mean across the training and serving throughput [79].

Figure 6.4 shows the the ADABench Metric for our reference implementation at different scale factors. It is important to mention that, we do not managed to run all use-cases at all scale factors. Table 6.5 shows a list at which scale factor that an experiment on the use-case is conducted.

Scale Factor	UC1	UC2	UC3
1	✓	✓	✓
5	✓	✓	✓
10	✓	✓	✓
50	✓	✓	
100	✓		
500	✓		

Table 6.5: Scale factor for use-cases

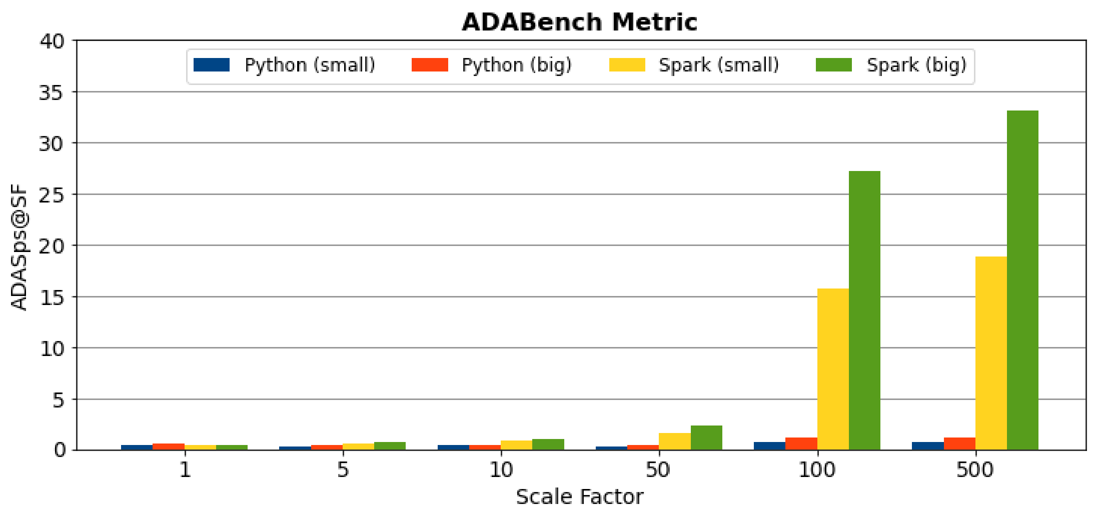


Figure 6.4: ADABench Metric - Combined metric of all stages and use cases per systems and scale factor (Full)

The metric in Figure 6.4 shows that a Spark-based implementation outperforms Python-based implementation at high scale factors. It is important to mention that, the high throughput that Spark-based implementation manages to achieve is due to the Spark runs processes in parallel. It means that Spark-based implementation utilizes more resources than Python-based implementation at the runtime. At a small scale factor, Python-based implementation has a comparable metric value. However, as the amount of data grows, so is the number of processing needed. And the disadvantage of Python single core implementation becomes more evident. This tells us that as long as the data fits into memory, Python-based implementation can have a higher throughput than Spark.

7 Related Work

Benchmarks have always been meant to compare multiple things on a similar problem as a basis for evaluation. In the ML space, this often means to compare the workloads on specific systems or to compare the performance of ML algorithms on a pre-determined problem. Raj Jain [20] explains in his book that benchmarks can be divided into three distinct types: *component-level benchmark*, *system-level benchmark*, and *algorithmic competition*.

Component-level benchmarks test and compare a system or application on the component level. In the ML space, it is hardware benchmarks in essence. It compares a specific application on different CPU or GPU in terms of training and/or inference performance. DeepBench [68] is a famous open source benchmarking tool that measures the performance of basic operations involved in training deep neural networks on different hardware platforms. DeepBench attempt to answer the question, "Which hardware provides the best performance on the basic operations used for training deep neural networks?". It does not work with deep learning frameworks or deep learning models built for applications. It cannot measure the time required to train an entire deep learning model.

System-level benchmarks test and compare complete ML system, which ADABench strives for. An ML algorithm is only a piece of the puzzle in a complete ML pipeline. It often consists of data provisioning, data cleaning, and feature engineering as well. Shah et al. [88] stated that surveys of data scientists show that ML data prep often dominates their time and effort, even up to 80%. For this reason, it is important to also consider these aspects in an ML benchmark. DAWNBench [27] serves as a good example. They benchmark training performance of systems. One of their main contribution is introducing a novel time-to-accuracy metric. In comparison to DAWNBench, ADABench measures end-to-end performance, starting from loading the data up until inference rather than evaluating only training performance.

Algorithmic competitions benchmarks compare different algorithms on a constrained problem. Netflix Prize [17], Mario AI [52], M4 Forecasting competition [7], and ML techniques comparison for phishing detection [8] are some examples. Such competition provide one or multiple representative datasets with a specific problem in mind, where where data scientists design and execute ML pipelines to the specific problem in order to achieve highest quality metric. ADABench, however, focuses on providing a realistic benchmarking system for industrial ML use-cases by generating arbitrary sized datasets that can be configured and offering representative end-to-end ML pipelines

Aside from the benchmark approach, another aspect that is commonly found in ML benchmarks is the use of toy dataset instead of real-world data. While they are practical to use, they do not reflect the data in the wild, where the data needs to be preprocessed. Recent works [73, 37, 35] start to address this problem by using an representative real-world dataset for their benchmark.

PMLB [73] states that numerous publicly available real-world and simulated benchmark datasets are inconsistent in terms of organization and adoption. This creates an unnecessary problem for data scientist at selecting datasets and curating specific benchmarks. PMLB introduces an accessible, curated, and developing public benchmark dataset to facilitate identification of the strengths and weaknesses of different ML methodologies. PMLB's data repository features meta-features among the benchmark datasets to characterize the diversity of available data. ADABench, on the other hand, provides an interface to generate our own data. Thanks to its flexibility, ADABench is able to build a realistic simulation of industrial data and quickly adapt to requirement changes in the data.

BigBench [37] can be seen as a basis for ADABench. It is an end-to-end big data benchmark that is based on real industry analytics use-cases. It provides a data generator that is capable of producing large amounts of data for an arbitrary schema. However, its focus lies on traditional data analytics use-cases. It compares end-to-end performance of analytic queries based on data types, query processing types, and analytic techniques. ADABench extends the scope of the use-cases to advance analytic use-cases.

BigDataBench [35] provides a scalable and unified big data and AI benchmark suite. They propose a data motif-based scalable benchmarking methodology that covers various application domains and workload types with representative real-world datasets. However, BigDataBench does not include dedicated end-to-end scenarios. ADABench, however, measures end-to-end ML pipelines that covers various dimensions.

8 Conclusions

An end-to-end machine learning benchmark allows us to have a more realistic insight on how the end-to-end pipeline will perform with the actual industrial data. ADABench is built to cover real business requirements and includes different scale ranges, which are mapped to problem sizes typically found in industry. Its benchmark model draws from industry standard as specified from the Transaction Processing Performance Council (TPC) and the Standard Performance Evaluation Corporation (SPEC). It addresses neglected challenges that often found in industry, such as the input data may vary in size across the pipeline, the input data may contain different kind of noises, the numerical representations may be sparse or dense.

In ADABench paper, 16 use-cases have been propose, which are derived from retail business applications and analytic tasks common in online bussinesses according to their market potential identified by McKinsey [25]. Six use-cases are already presented in the paper and this thesis extends them by three additional use-cases.

The primary contribution fo this thesis is the investigation of three additional industry-relevant use-cases, including their corresponding performance metrics. We analyze the importance of these use-cases to businesses and provide a reference implementation in a Python and Spark environment. In order to demonstrate the benchmark, we also provide data generation schema for all use-cases that reflect industrial data.

Our experiments are executed on two different nodes at various scale factors. We measure the runtime of the whole pipeline to calculate the throughput of each configuration on each use-case and compare them. Our evaluation shows the different trade-offs in terms of performance based on data size, choice of environment and complexity. We see that Python and Spark implementations behave differently in scaling. As the the data size grows, Spark outperforms Python thanks to the parallelization.

Future Works

Spark implementations on UC2 and UC3 are unfortunately suboptimal and can be further improved. Another approach that can be done to the UC2 is to utilize Horovod on Spark to enable multi-GPU training. Horovod is a distributed training framework that allows us to train directly on a Spark DataFrame [14].

Facebook Prophet only supports univariate time series. Currently, there is limited support for Prophet in Spark and it requires further work to enable the integration of Prophet and Spark. Although there is an attempt to run Facebook Prophet on Spark [71], it still needs to transform the data frame back to pandas data frame.

A List of UC1 Categories and Sub-Categories

Books		
Education & Reference	Literature & Fiction	New
Used & Rental Textbooks	NO_SUB_CAT	

Clothing & Accessories			
Novelty & Special Use	Accessories	Pants	Men
Tops & Tees	Active	Bras	Women
Dress Shirts	Intimates	NO_SUB_CAT	

Music			
Classical	Blues	Pop	Classic Rock
R&B	World Music	Jazz	Rock
Dance & Electronic	Miscellaneous	Rap & Hip-Hop	Country
Hard Rock & Metal	Latin Music	Alternative Rock	Folk

Home & Kitchen		
Kitchen & Dining	Furniture	Tabletop
Bedding	Kitchen Utensils & Gadgets	Home Decor

Tools & Home Improvement			
Lighting & Ceiling Fans	Hand Tools	Power Tools	Hardware
Power & Hand Tools	Power Tool Accessories	Electrical	

Electronics		
Portable Audio & Video	Accessories	Computer Accessories
Camera & Photo	Computer Components	Audio & Video Accessories
Computers & Accessories	Accessories & Supplies	

Toys & Games		
Hobbies	Figures	Games
Electronics for Kids	Dolls	Stuffed Animals & Plush
Vehicles & Remote-Control	Dolls & Accessories	Animals & Figures
Sports & Outdoor Play	Learning & Education	Board Games
Action & Toy Figures	NO_SUB_CAT	

Movies & TV	
Movies	TV

Sports & Outdoors		
Fan Shop	Outdoor Recreation	Team Sports
Boating & Water Sports	Camping & Hiking	Accessories
Clothing	Men	Hunting & Fishing
Hunting		

References

[1] Hellofresh capital markets day 2019. https://ir.hellofreshgroup.com/download/companies/hellofresh/Capital_Markets_Day/CMD_2019_WEBSITE.pdf. Accessed: 2021-05-23.

[2] Hellofresh capital markets day 2020. https://ir.hellofreshgroup.com/download/companies/hellofresh/Capital_Markets_Day/CMD_2020_Final_Presentation_10122020.pdf. Accessed: 2021-05-23.

[3] Hellofresh sustainability report 2020. https://ir.hellofreshgroup.com/download/companies/hellofresh/AnnualReports/HF_Sustainability-Report-2020_EN_FINAL.pdf. Accessed: 2021-05-23.

[4] How hellofresh became carbon neutral. https://www.newfoodmagazine.com/article/141758/sustainability-hellofresh/. Accessed: 2021-05-23.

[5] Prophet forecasting at scale. https://research.fb.com/blog/2017/02/prophet-forecasting-at-scale/. Accessed: 2021-05-23.

[6] Sentiment analysis explained. Tech. rep., Lexalytics.

[7] The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting 36*, 1 (2020), 54–74. M4 Competition.

[8] Abu-Nimeh, S., Nappa, D., Wang, X., and Nair, S. A comparison of machine learning techniques for phishing detection. In *Proceedings of the Anti-Phishing Working Groups 2nd Annual ECrime Researchers Summit* (New York, NY, USA, 2007), Association for Computing Machinery.

[9] Agarwal, R. Multiclass text classification - pytorch. https://www.kaggle.com/mlwhiz/multiclass-text-classification-pytorch, 2019. Accessed: 2021-04-01.

[10] Alamoodi, A., Zaidan, B., Zaidan, A., Albahri, O., Mohammed, K., Malik, R., Almahdi, E., Chyad, M., Tareq, Z., Albahri, A., Hameed, H., and Alaa, M. Sentiment analysis and its applications in fighting covid-19 and infectious diseases: A systematic review. *Expert Systems with Applications* (2020), 114155.

[11] Alexandrov, A., Benidis, K., Bohlke-Schneider, M., Flunkert, V., Gasthaus, J., Januschowski, T., Maddix, D. C., Rangapuram, S., Salinas, D., Schulz, J., et al. Gluonts: Probabilistic time series models in python. *arXiv preprint arXiv:1906.05264* (2019).

[12] Altintas, M., and Tantug, A. C. Machine learning based ticket classification in issue tracking systems.

[13] Araque, O., Gatti, L., Staiano, J., and Guerini, M. Depechemood++: a bilingual emotion lexicon built through simple yet powerful techniques. *CoRR abs/1810.03660* (2018).

[14] Authors, T. H. Horovod on spark. https://horovod.readthedocs.io/en/stable/spark_include.html, 2019. Accessed: 2021-08-09.

[15] Basiri, M. E., Nemati, S., Abdar, M., Cambria, E., and Acharya, U. R. Abcdm: An attention-based bidirectional cnn-rnn deep model for sentiment analysis. *Future Generation Computer Systems 115* (2021), 279 – 294.

[16] Bengfort, B., Bilbro, R., and Ojeda, T. *Applied Text Analysis with Python: Enabling Language-Aware Data Products with Machine Learning*, 1st ed. O'Reilly Media, Inc., 2018.

[17] Bennett, J., Lanning, S., and Netflix, N. The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD* (2007).

[18] Boiy, E., and francine Moens, M. A machine learning approach to sentiment analysis in multilingual web texts. *Information Retrieval* (2009), 526–558.

[19] Boughorbel, S., Jarray, F., and El-Anbari, M. Optimal classifier for imbalanced data using matthews correlation coefficient metric. *PLOS ONE 12* (06 2017), e0177678.

[20] Bukh, P. N. D. The art of computer systems performance analysis, techniques for experimental design, measurement, simulation and modeling, 1992.

[21] Cecaj, A., Lippi, M., Mamei, M., and Zambonelli, F. Comparing deep learning and statistical methods in forecasting crowd distribution from aggregated mobile phone data. *Applied Sciences 10*, 18 (2020).

[22] Chatfield, C. The holt-winters forecasting procedure. *Journal of the Royal Statistical Society. Series C (Applied Statistics) 27*, 3 (1978), 264–279.

[23] Chatfield, C. The holt-winters forecasting procedure. *Journal of the Royal Statistical Society. Series C (Applied Statistics) 27*, 3 (1978), 264–279.

[24] Chicco, D., and Jurman, G. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics 21* (01 2020).

[25] Chui, M., Manyika, J., Miremadi, M., Henke, N., Chung, R., Nel, P., and Malhotra, S. Notes from the ai frontier - insights from hundreds of use cases. Tech. rep., McKinsey Global Institute, 4 2018.

[26] Colas, F., and Brazdil, P. Comparison of svm and some older classification algorithms in text classification tasks. vol. 217.

[27] Coleman, C., Kang, D., Narayanan, D., Nardi, L., Zhao, T., Zhang, J., Bailis, P., Olukotun, K., Ré, C., and Zaharia, M. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. *CoRR abs/1806.01427* (2018).

[28] Cryer, J. D., and Chan, K.-S. Time series analysis with applications in r. *Journal of Forecasting 2* (2008).

[29] Dai, W., and Berleant, D. Benchmarking deep learning hardware and frameworks: Qualitative metrics. *CoRR abs/1907.03626* (2019).

[30] Denecke, K. Using sentiwordnet for multilingual sentiment analysis. In *2008 IEEE 24th International Conference on Data Engineering Workshop* (2008), pp. 507–512.

[31] Domingos, P., and Pazzani, M. On the optimality of the simple bayesian classifier under zero-one loss. *Mach. Learn. 29* (Nov. 1997).

[32] Feldman, R. Techniques and applications for sentiment analysis. *Commun. ACM 56*, 4 (Apr. 2013).

[33] Filatova, E. Irony and sarcasm: Corpus generation and analysis using crowdsourcing. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)* (Istanbul, Turkey, May 2012), European Language Resources Association (ELRA), pp. 392–398.

[34] Gabernet, A. R., and Limburn, J. Breaking the 80/20 rule: How data catalogs transform data scientists' productivity. `https://www.ibm.com/cloud/blog/ibm-data-catalog-data-scientists-productivity`, 8 2017. Accessed: 2020-11-29.

[35] Gao, W., Zhan, J., Wang, L., Luo, C., Zheng, D., Ren, R., Zheng, C., Lu, G., Li, J., Cao, Z., Zhang, S., and Tang, H. BigDataBench: A Dwarf-based Big Data and AI Benchmark Suite.

[36] Ghassemi, M., Naumann, T., Schulam, P., Beam, A. L., and Ranganath, R. Opportunities in machine learning for healthcare. *CoRR abs/1806.00388* (2018).

[37] Ghazal, A., Rabl, T., Hu, M., Raab, F., Poess, M., Crolotte, A., and Jacobsen, H.-A. BigBench: Towards an industry standard benchmark for big data analytics. pp. 1197–1208.

[38] Graves, A. *Supervised Sequence Labelling with Recurrent Neural Networks*. Studies in computational intelligence. Springer, Berlin, 2012.

[39] Graves, A., and Schmidhuber, J. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks 18*, 5 (2005), 602–610. IJCNN 2005.

[40] Hartmann, J., Huppertz, J., Schamp, C., and Heitmann, M. Comparing automated text classification methods. *International Journal of Research in Marketing 36* (10 2018).

[41] Harvey, A., and Peters, S. Estimation procedures for structural time series models. *Journal of Forecasting 9* (1990), 89–108.

[42] Hastie, T., and Tibshirani, R. Generalized additive models; some applications. In *Generalized Linear Models* (New York, NY, 1985), R. Gilchrist, B. Francis, and J. Whittaker, Eds., Springer US, pp. 66–81.

[43] Hastie, T., Tibshirani, R., and Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009.

[44] He, F., Liu, T., and Tao, D. Control batch size and learning rate to generalize well: Theoretical and empirical evidence. In *NeurIPS* (2019).

[45] Holt, C. C. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting 20*, 1 (2004), 5–10.

[46] Hui, C. Effect of batch size and number of gpus on model accuracy. `https://ai.stackexchange.com/questions/17424/effect-of-batch-size-and-number-of-gpus-on-model-accuracy`, 2020. Accessed: 2021-08-09.

[47] HUSSEIN, D. M. E.-D. M. A survey on sentiment analysis challenges. *Journal of King Saud University - Engineering Sciences 30*, 4 (2018), 330 – 338.

[48] HYNDMAN, R. J., AND ATHANASOPOULOS, G. *Forecasting: principles and practice*. OTexts, 2018.

[49] JAGANNATHAN, R. White paper âĂŞsentimentsanalysis. Tech. rep., Usha Martin Technologies, 7 2018.

[50] JURAFSKY, D., AND MARTIN, J. *Speech and Language Processing*, vol. 3. 12 2020.

[51] KAGGLE. Store item demand forecasting challenge. `https://www.kaggle.com/c/demand-forecasting-kernels-only/overview`, 2018. Accessed: 2020-10-01.

[52] KARAKOVSKIY, S., AND TOGELIUS, J. The mario ai benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games 4*, 1 (2012), 55–67.

[53] KIBRIYA, A. M., FRANK, E., PFAHRINGER, B., AND HOLMES, G. Multinomial naive bayes for text categorization revisited. In *Australian Conference on Artificial Intelligence* (2004).

[54] LI, S. Multi-class text classification with scikit-learn. `https://towardsdatascience.com/multi-class-text-classification-with-scikit-learn-12f1e60e0a9f`, 2018. Accessed: 2021-04-01.

[55] LIAKOS, K. G., BUSATO, P., MOSHOU, D., PEARSON, S., AND BOCHTIS, D. Machine learning in agriculture: A review. *Sensors 18*, 8 (2018), 2674.

[56] LIU, G., AND GUO, J. Bidirectional lstm with attention mechanism and convolutional layer for text classification. *Neurocomputing 337* (2019), 325–338.

[57] MAKRIDAKIS, S., SPILIOTIS, E., AND ASSIMAKOPOULOS, V. Statistical and machine learning forecasting methods: Concerns and ways forward. *PLOS ONE 13*, 3 (03 2018), 1–26.

[58] MALIK, U. Python for nlp: Movie sentiment analysis using deep learning in keras. `https://stackabuse.com/python-for-nlp-movie-sentiment-analysis-using-deep-learning-in-keras`, 2019. Accessed: 2021-04-01.

[59] MANNING, C. D., RAGHAVAN, P., AND SCHÃIJTZE, H. *Scoring, term weighting, and the vector space model*. Cambridge University Press, 2008, p. 100âĂŞ123.

[60] MANNING, C. D., RAGHAVAN, P., AND SCHÜTZE, H. *Introduction to Information Retrieval*. Cambridge University Press, USA, 2008.

[61] MARSLAND, S. *Machine Learning: An Algorithmic Perspective, Second Edition*, 2nd ed. Chapman and Hall/CRC, 2014.

[62] MATHUR, M. Sentiment analysis on amazon musical instruments reviews. `https://www.kaggle.com/madz2000/sentiment-analysis-89-accuracy`, 2019. Accessed: 2021-04-01.

[63] MOENS, M.-F. *Information Extraction: Algorithms and Prospects in a Retrieval Context*, vol. 21. 01 2006.

[64] MOHAN, K. Sentiment analysis: Universal sentence encoder 91%. `https://www.kaggle.com/kshitijmohan/sentiment-analysis-universal-sentence-encoder-91`, 2019. Accessed: 2021-04-01.

[65] Monostori, L. Ai and machine learning techniques for managing complexity, changes and uncertainties in manufacturing. *Engineering applications of artificial intelligence 16*, 4 (2003), 277–291.

[66] Mozaffari, A., Emami, M., and Fathi, A. A comprehensive investigation into the performance, robustness, scalability and convergence of chaos-enhanced evolutionary algorithms with boundary constraints. *Artificial Intelligence Review* (2018), 1–62.

[67] Murphy, K. P. *Machine Learning: A Probabilistic Perspective.* The MIT Press, 2012.

[68] Narang, S., and Diamos, G. Deepbench, 2016.

[69] Ni, J., Li, J., and McAuley, J. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (Hong Kong, China, Nov. 2019), Association for Computational Linguistics, pp. 188–197.

[70] Nielsen, M. A. Neural networks and deep learning, 2018.

[71] Obeidat, B., Smith, B., Heintz, B., and O'Malley, K. Fine-grained time series forecasting at scale with facebook prophet and apache spark: Updated for spark 3. `https://databricks.com/blog/2021/04/06/fine-grained-time-series-forecasting-at-scale-with-facebook-prophet-and-apache-spark-updated-for-spark-3.html`, 2021. Accessed: 2021-08-09.

[72] Olah, C. Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015. Accessed: 2021-02-14.

[73] Olson, R., La Cava, W., Orzechowski, P., Urbanowicz, R., and Moore, J. Pmlb: A large benchmark suite for machine learning evaluation and comparison. *BioData Mining 10* (12 2017).

[74] Papastefanopoulos, V., Linardatos, P., and Kotsiantis, S. Covid-19: A comparison of time series methods to forecast percentage of active cases per population. *Applied Sciences 10* (06 2020), 3880.

[75] Picazo-Vela, S. The effect of online reviews on customer satisfaction: An expectation disconfirmation approach. *Dissertations 1* (01 2011).

[76] Powers, D. Evaluation: From precision, recall and f-factor to roc, informedness, markedness & correlation. *Mach. Learn. Technol. 2* (01 2008).

[77] ProgrammerSought. Keras examples-imdb_bidirectional_lstm [using bi-lstm to implement sentiment classification]. `https://www.programmersought.com/article/84693910701/`. Accessed: 2020-12-01.

[78] Qamili, R., Shabani, S., and Schneider, J. An intelligent framework for issue ticketing system based on machine learning. pp. 79–86.

[79] Rabl, T., Brücke, C., Härtling, P., Stars, S., Palacios, E., Patel, H., Srivastava, S., Boden, C., Meiners, J., and Schelter, S. ADABench - Towards an Industry Standard Benchmark for Advanced Analytics.

[80] Rabl, T., Frank, M., Mousselly Sergieh, H., and Kosch, H. A Data Generator for Cloud-Scale Benchmarking. vol. 6417, pp. 41–56.

[81] Radiuk, P. M., et al. Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. *Information Technology and Management Science 20*, 1 (2017), 20–24.

[82] Reyes, S. Multi-class text classification (tfidf). `https://www.kaggle.com/selener/multi-class-text-classification-tfidf`, 2018. Accessed: 2021-04-01.

[83] Rish, I. An empirical study of the naive bayes classifier. *IJCAI 2001 Work Empir Methods Artif Intell 3* (01 2001).

[84] Salinas, D., Flunkert, V., Gasthaus, J., and Januschowski, T. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting 36*, 3 (2020), 1181–1191.

[85] Schafer, J. B., Konstan, J., and Riedl, J. Recommender systems in e-commerce. In *Proceedings of the 1st ACM Conference on Electronic Commerce* (New York, NY, USA, 1999), EC '99, Association for Computing Machinery, p. 158âĂŞ166.

[86] Schiermeier, Q. And now for the energy forecast: Germany works to predict wind and solar power generation. *Nature 535*, 7611 (2016).

[87] Sebastiani, F. Machine learning in automated text categorization. *ACM Comput. Surv. 34*, 1 (Mar. 2002).

[88] Shah, V., and Kumar, A. The ml data prep zoo: Towards semi-automatic data preparation for ml. In *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning* (2019), pp. 1–4.

[89] Taboada, M., Brooke, J., Tofiloski, M., Voll, K., and Stede, M. Lexicon-based methods for sentiment analysis. *Computational Linguistics 37* (06 2011), 267–307.

[90] Taylor, S., and Letham, B. Forecasting at scale. *The American Statistician 72* (09 2017).

[91] Urbanowicz, R., Kiralis, J., Sinnott-Armstrong, N., Heberling, T., Fisher, J., and Moore, J. Gametes: A fast, direct algorithm for generating pure, strict, epistatic models with random architectures. *BioData mining 5* (10 2012), 16.

[92] Wilson, G. T. Time Series Analysis: Forecasting and Control, 5th Edition , by George E. P. Box , Gwilym M. Jenkins , Gregory C. Reinsel and Greta M. Ljung , 2015 . Published by John Wiley and Sons Inc. , Hoboken, N. *Journal of Time Series Analysis 37*, 5 (September 2016), 709–711.

[93] Witten, I., and Bell, T. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory 37* (07 1991), 1085–1094.

[94] Wood, S. *Generalized Additive Models: An Introduction With R*, vol. 66. 01 2006.

[95] Zhai, Y. Time series forecasting competition among three sophisticated paradigms.

[96] Zhang, L., and Liu, B. *Sentiment Analysis and Opinion Mining*. Springer US, Boston, MA, 2017, pp. 1152–1161.

[97] Zhang, L., Wang, S., and Liu, B. Deep learning for sentiment analysis : A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8* (01 2018).

[98] Zhou, C., Sun, C., Liu, Z., and Lau, F. C. M. A c-lstm neural network for text classification, 2015.

[99] Zhou, P., Qi, Z., Zheng, S., Xu, J., Bao, H., and Xu, B. Text classification improved by integrating bidirectional lstm with two-dimensional max pooling, 2016.

[100] Zubrinic, K., Milicevic, M., and Zakarija, I. Comparison of naive bayes and svm classifiers in categorization of concept maps. *International journal of computers 7* (01 2013), 109–116.