UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMATÍCA

Automatic task discovery: Towards full automation of the machine learning lifecycle

Jonathan Gehmayr

Mestrado em Ciência de Dados

Relatório de Estágio orientado por:
Prof. Doutor Cátia Luísa Santana Calisto Pesquita

2023

Acknowledgments

Firstly, I would like to thank my manager Mario Gonçalves and my colleagues at Siemens for the inspiration and support provided throughout the development of this work. I would also like to thank Professora Cátia Pesquita for the great scientific supervision. Additionally, I would like to thank my friends and family for their love and encouraging words.

Resumo

A procura por cientistas de dados aumentou significativamente na última década, e as ofertas de emprego para esses profissionais aumentaram 256% em comparação com 2018 [29]. O Bureau of Labor Statistics dos EUA [71] prevê que, até 2026, o número de empregos que exigem capacidades tipícas de um cientista de dados irá aumentar em 27,9%. O crescimento é atribuído à economia de custos em poder de computação e armazenamento de big data, bem como ao excelente desempenho de algoritmos de Aprendizagem de Máquina (AM) em tarefas como classificação de imagens [49, 37], processamento de linguagem natural [33, 20] e desafios generativos não supervisionados [73].

Uma vez que a procura de cientistas de dados não pode ser satisfeita através dos profissionais já formados, surgiu a direção de investigação da Aprendizagem Automática de Máquinas (AutoAM). O paradigma da AutoAM tem como objetivo automatizar o ciclo de vida da AM do início ao fim, de modo a que seja necessário apenas um mínimo esforço humano para criar modelos de AM. O AutoAM é descrito [105] como uma ferramenta que reduz a procura de cientistas de dados e democratiza a possibilidade dos peritos no domínio aplicarem algoritmos de AM sem haver a necessidade de um conhecimento profundo dos algoritmos em si, e suas estatísticas subjacentes. Avanços em AutoAM resultam numa necessidade menor de mão de obra humana, sendo assim, sendo por isso o desenvolvimento de soluções de AutoAM um tópico de grande interesse para as empresas [38].

Ao criar uma aplicação de AM, os cientistas de dados seguem o ciclo de vida de AM que consiste num ciclo de vida de AM interno e externo. O ciclo de vida interno do AM consiste em tarefas como a preparação de dados, a engenharia de características e a criação de modelos [1, 12, 32]. Estas tarefas consomem muito tempo [65], sendo que os cientistas de dados gastam 80% do seu tempo em tarefas de baixo nível, como melhorar a qualidade dos dados ou encontrar configurações óptimas de algoritmos. Para reduzir este tempo, foram desenvolvidas várias estruturas AutoAM, quer de código aberto [75, 85, 27, 74] quer comerciais [67, 50, 35, 23, 46]. O ciclo de vida externo do AM consiste em competências transversais, como a construção da confiança do cliente e a orientação no ambiente do cliente, e a formulação do enquadramento da tarefa de AM [44, 3, 91]. A fase de enquadramento das tarefas de AM é crucial para colmatar o fosso de conhecimentos

entre os cientistas de dados e as empresas. Os inquéritos mostram que nem as estruturas de código aberto nem as comerciais abordam a automatização do enquadramento de tarefas, enquanto tanto as empresas como os cientistas de dados desejam uma maior automatização desta fase [91]. Ao expandir a funcionalidade das estruturas AutoAM para descobrir automaticamente os quadros de tarefas de AM, a ciência dos dados seria ainda mais democratizada e os indivíduos sem conhecimentos técnicos seriam capazes de aproveitar o poder do AM nos seus domínios [44]. Assim, o objetivo deste trabalho é desenvolver uma abordagem para descobrir automaticamente quadros de tarefas de AM supervisionados. Para isso, é criado um corpus de dados de conjuntos de dados rotulados com quadros de tarefas de AM. O corpus de dados é utilizado para afinar as estruturas de anotação de tipos de colunas e detetar quadros de tarefas de AM supervisionados nesses conjuntos de dados.

A recolha do corpus de dados consiste em três etapas: 1. encontrar tabelas de dados open-source, 2. filtrar os conjuntos de dados e 3. rotular os conjuntos de dados com as tarefas de AM adequadas. A biblioteca de dados open-source do Kaggle foi consultada para obter conjuntos de dados Comma Separated Values (CSV) e metadados, que formaram o corpus de dados em bruto. De seguida, filtros foram aplicados para excluir os dados que não podem ser comercializados bem como para garantir que as tabelas de dados podem ser processadas com sucesso (filtros para limitar o tamanho do armazenamento, o número de colunas, etc.). De seguida, o corpus de dados foi etiquetado com quadros de tarefas de AM. A etiquetagem de cada conjunto de dados exigiu uma linguagem formal para representar os quadros. Assim, foi criada a linguagem de representação de quadros de tarefas de AM: MATA. A MATA organiza uma estrutura de tarefas de AM supervisionada com uma entidade, um alvo, propriedades de tipo AM e operações de filtro, agregação de colunas e transformação de linhas. Durante um mês, 256 conjuntos de dados foram rotulados através da MATA. Os conjuntos de dados etiquetados constituíram a entrada para a modelação da descoberta automática de tarefas.

A rotulagem e a exploração do corpus de dados revelaram que a maioria dos quadros de tarefas são tarefas de regressão de séries temporais. Além disso, descobriu-se que as operações de filtragem, agregação de colunas e transformação de linhas são demasiado subjectivas, ou demasiado raras para construir uma base fundamental sólida, pelo que estas propriedades MATA não podem ser aprendidas por um modelo AM. Isto significa que podemos formular o problema de descoberta automática de tarefas como uma cadeia de anotação de tipos de coluna, o que significa que é treinado um modelo de AM de classificação multiclasse, que considera uma coluna da tabela como um único registo, para o qual é prevista a classe correcta (entidade, alvo (classificação), alvo (regressão), outra). Para isso, as estruturas de anotação de tipo de coluna pré-treinadas TURL [24] e Sato [102] foram otimizados. O TURL baseia-se em transformadores e combina de forma inteligente dados tabulares com metadados de tabelas para prever rótulos semânticas para

colunas de tabelas. Sato é uma rede neuronal que considera apenas dados tabulares simples e necessita de um processo de engenharia de características manual. Neste trabalho, é proposto um pipeline de AM que concatena metadados de tabelas com dados tabulares, de modo a que cada registo dos dados de entrada represente uma coluna de tabela com os respectivos metadados. Os dados concatenados são introduzidos numa estrutura de anotação do tipo de coluna, que infere a etiqueta para o registo. De seguida, as entidades e as colunas de destino são combinadas para construir quadros de tarefas de AM.

Para avaliar os desempenhos nas experiências, foi aplicada a validação cruzada 5 vezes. As experiências realizadas incluem a comparação entre o TURL e o Sato, a otimização do desempenho do modelo com métodos básicos de aumento de dados e de otimização de hiperparâmetros e a análise dos erros do modelo. A comparação entre o TURL e o Sato exigiu que o Sato também fosse capaz de processar metadados de tabelas. O Turl e o Sato foram ambos treinados utilizando sete partições de dados diferentes de um conjunto de dados (apenas dados tabulares, dados tabulares + metadados, etc.). O resultado desta experiência mostrou que o TURL supera o Sato em todas as partições de dados, exceto na utilização de apenas dados tabulares, em que o Sato teve um desempenho ligeiramente superior. Por conseguinte, apenas o TURL foi utilizado nas experiências sucessivas em que o desempenho do modelo foi optimizado. Uma vez que o corpus de dados criado é pequeno, foi aplicada uma reamostragem dos dados tabulares para aumentar o número de amostras de treino. Através da reamostragem, o desempenho do modelo pode ser aumentado em 20%. De seguida, procedeu-se À otimização do parâmetro de aprendizagem do TURL, o que levou a um desempenho final do modelo de F1=0.863, F1 STD=0.009 no conjunto de validação e F1=0.847 no conjunto de teste. A análise de erros mostrou que, apesar de haver desequilíbrio entre as classes, o modelo consegue aprender as classes minoritárias. A análise qualitativa das estruturas de tarefas de AM resultantes dos conjuntos de dados de teste confirma que o modelo é capaz de prever estruturas de tarefas de AM significativas se o conjunto de dados de entrada for semelhante aos dados de treino.

Para concluir, com o MATA foi criada uma linguagem de representação de tarefas de AM que pode ser utilizada para enquadrar uma vasta gama de diferentes tarefas de AM supervisionadas. O corpus de dados rotulado com o MATA é pequeno, mas já capta uma vasta gama de diferentes enquadramentos de tarefas permitindo que os modelos de AM aprendam os padrões subjacentes aos enquadramentos de tarefas de AM. No entanto, uma vez que os conjuntos de dados se revelem menos semelhantes ao corpus de treino nem sempre são classificados corretamente, o aumento do corpus de dados atual resultaria numa melhor generalização do modelo. No entanto, o atual corpus de dados pode ser utilizado em investigação futura como um conjunto de dados de referência para a descoberta automática de tarefas. Os resultados estão de acordo com investigações anteriores que afirmam o desempenho superior das arquitecturas baseadas em transformadores em

relação às abordagens de engenharia de características e que o aumento dos dados através da reamostragem é eficaz para aumentar o desempenho das estruturas de anotação de tipos de colunas. Com este trabalho, foi criada a primeira abordagem para automatizar totalmente o enquadramento de tarefas de AM, que atinge um elevado desempenho, de modo a que possam ser efectuados os primeiros testes de integração em estruturas AutoAM existentes. Deste modo, será mais fácil para os especialistas do domínio entrarem no mundo do AM e tornarem o seu trabalho mais fácil e mais eficiente.

Palavras-chave: Aprendizagem automática de máquinas, aprendizagem de máquina, processamento de linguagem natural, anotação de tipo de coluna, transformador

Abstract

Our world generates a vast amount of data daily, leading to a steady increase in the demand for skilled data scientists. However, employers struggle to meet this demand for data scientists. Thus, the field of Automated Machine Learning (AutoML) has emerged, focusing on automating the stages of the Machine Learning (ML) lifecycle. While many stages of the ML lifecycle have already been fully automated, limited research has been directed toward automating the stage of *task framing*.

This work introduces the first data-driven approach for Automatic Task Discovery (ATD) of supervised ML tasks. The ML task frame representation language MATA is proposed, which is used to label a data corpus comprising 256 datasets with ML task frames. ATD is formulated as a multi-class classification task, for which the column-type annotation frameworks TURL [24] and Sato [102] are fine-tuned and compared. TURL achieves superior performance when compared to Sato and is further enhanced through data augmentation and hyperparameter optimization.

The resulting model achieves an impressive F1-score of 0.847 on the test set. Additionally, a qualitative analysis of the generated task frames demonstrates the model's ability to discover meaningful ML task frames. With this work, a novel data-driven approach for automatic discovery of supervised ML tasks is presented, ready for integration into existing AutoML frameworks.

Keywords: AutoML, machine learning, natural language processing, column type annotation, transformer

Contents

List of Figures

List of Tables

Acronyms

AM Aprendizagem de Máquina. iii, iv, v, vi

ANN Artificial Neural Network. 12, 13, 14

ATD Automatic Task Discovery. viii, xiii, xv, 3, 4, 12, 18, 23, 25, 33, 35, 36, 37, 38, 40, 41, 42, 44, 45, 49, 51

AUC Area Under the Curve. 11

AutoAM Aprendizagem Automática de Máquinas. iii, iv, vi

AutoML Automated Machine Learning. viii, 1, 2, 5, 52

CNN Convolutional Neural Network. 14, 16

CRF Conditional Random Fields. 19

CSV Comma Separated Values. iv, xiii, 3, 25, 26, 27, 28, 33, 35, 37, 38, 42

DL Deep Learning. 7, 8, 9, 16, 27

EDA Exploratory Data Analysis. 35

JSON JavaScript Object Notation. 19, 26

KG Knowledge Graph. 35

LDA Latent Dirichlet Allocation. 19

LSTM Long-Short Term Memory. 14

MAE Mean Absolute Error. 11

ML Machine Learning. viii, xi, xiii, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 33, 35, 36, 37, 38, 40, 42, 44, 51, 52

MLP Multi Layer Perceptron. 13, 14

MSE Mean Squared Error. 11, 13

NAS Neural Architecture Search. 10

NER Named Entity Recognition. 12

NLP Natural Language Processing. 3, 5, 11, 12

RNN Recurrent Neural Network. 14, 16

SGD Stochastic Gradient Descent. 14

SLP Single Layer Perceptron. xiii, 13, 15

STD Standard Deviation. 28, 33, 39

TB Terrabyte. 42

TFRL Task Frame Representation Language. 19, 20, 21, 22, 28, 51

Chapter 1

Introduction

1.1 Motivation

Over the last decade, the demand for data scientists has risen tremendously in various domains. According to statistics from Indeed [29], job advertisements for data scientists increased by 256% compared to 2018. This surging demand for data scientists is not only evident on job platforms like Indeed but also supported by the U.S. Bureau of Labor Statistics [71], which predicts that by the year 2026, the number of jobs requiring data science skills will expand by 27.9%. Several factors contribute to this growth, including the decreasing cost of computing power and big data storage. Furthermore, machine learning ML algorithms have achieved outstanding performance in tasks such as image classification [49, 37], natural language processing [33, 20], and unsupervised generative challenges [73].

Due to the insatiable demand for data scientists, the research direction of AutoML has emerged. The AutoML paradigm is focused on automating the entire end-to-end ML lifecycle, minimizing the need for human intervention in creating ML models. Zoeller and Huber [105] define AutoML as a tool aimed at reducing the demand for data scientists and democratizing access to ML for domain experts who may not possess in-depth knowledge of the underlying algorithms and statistics. Because advancements in AutoML contribute to reducing the workforce demand, businesses are interested in the development of AutoML solutions [38].

The traditional workflow of a data scientist in creating machine learning ML models consists of the stages of data preparation, feature engineering, and model generation. This ML pipeline is depicted similarly in various literature sources [1, 12, 32]. The process of producing a ML model follows a certain structure and it is reported [65] that data scientists spend 80% of their time with time-consuming low-level activities like enhancing data quality or finding the best configurations for a given algorithm. To alleviate this workload, various AutoML frameworks have been developed, which are available as open-source tools [75, 85, 27, 74] or as commercial solutions [67, 50, 35, 23, 46].

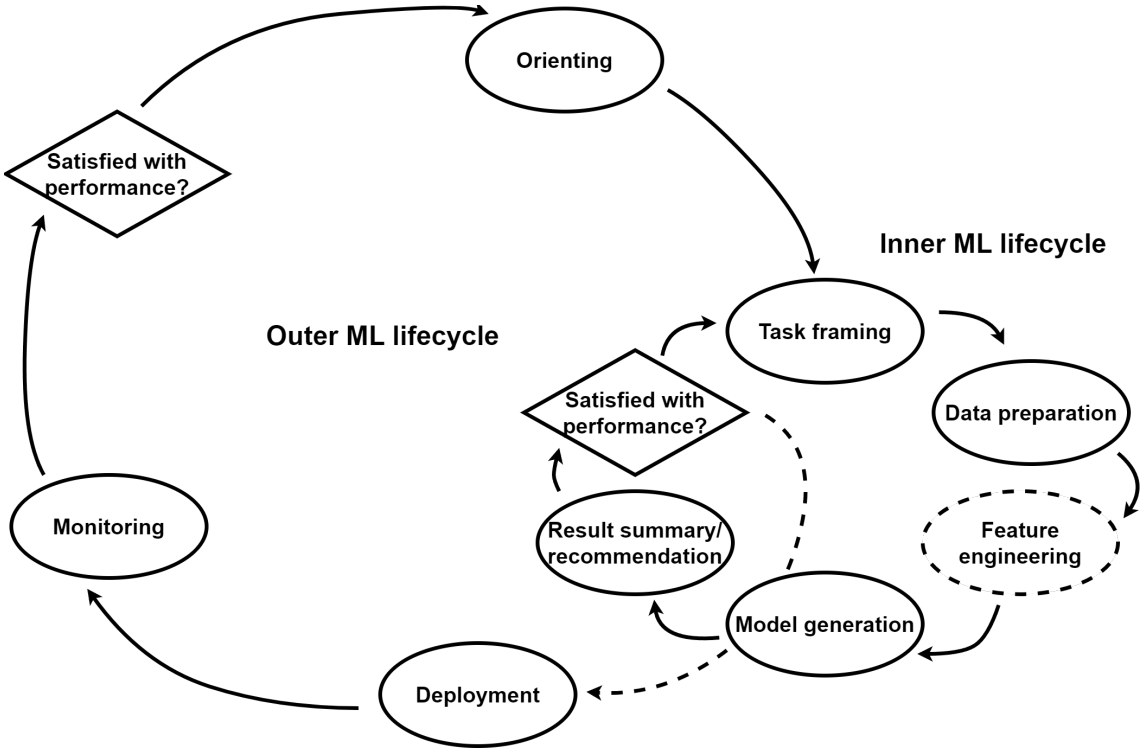

Figure 1.1: Diagram illustrating the outer- and inner ML lifecycle. The outer ML lifecycle focuses rather on soft skills like orienting in the customer's environment, while the inner ML lifecycle focuses on the technical aspects of ML model creation.

Other literature [44, 3, 91] suggests extending the horizon of AutoML and states that the ML lifecycle consists of both an outer cycle and an inner cycle. The outer cycle encompasses soft skills such as building trust with the customer, familiarizing with the customer's environment, and deploying and monitoring the designed ML model. The inner cycle consists of stages including task framing, data preparation, feature engineering, model generation, and result summary/recommendation. This approach of structuring the ML lifecycle is illustrated in Figure 1.1. Especially, the task of *task framing* is crucial to bridge the knowledge gap between data scientists and business stakeholders.

However, none of the aforementioned frameworks, whether open-source or commercial, currently addresses the automation of the *task framing* stage. Surveys conducted by Xin et al. [97] and Wang [91] show that data scientists are not using any framework that automates the *task framing* stage. Wang [91] also demonstrates that both business stakeholders and data scientists express an urge for further automation of the *task framing* stage. Makinen [59] underlines the importance of automating the entire ML lifecycle. By expanding the functionality of AutoML frameworks to include *task framing*, data science can be democratized further, enabling individuals with no technical expertise to harness the power of ML in their respective domains [44].

1.2 Objective

Research tackling the challenge of automated task framing is very limited. Schreck and Veeramachaneni [78] propose Trane, a formal language to structure ML tasks. Xu [98] builds on Trane and implemented an iterative system to recommend ML tasks for a given dataset. The system of Xu [98] is not able to independently discover ML tasks and is computationally inefficient. Thus, the objective of this work is to develop the first data-driven approach to discover supervised ML tasks in a dataset. Given a dataset in CSV format, it is aimed to frame all supervised ML task frames, that a data scientist would frame for the dataset. By treating a dataset as a document, Natural Language Processing (NLP) methods can be applied, such as column type annotation frameworks which are multiclass classification models. The task of this work is referred to as *modeling of ATD*, which is defined with the following task frame:

> *ATD is a column-type annotation problem, meaning that a multiclass classification ML model is trained, which treats each table column as an individual record. The model's objective is to predict the correct class for each column, which can belong to one of the following categories: entity, target (classification), target (regression), or other.*

1.3 Contributions

The contributions of this work are summarized in the following:

- **Definition of the task framing language MATA**: In this work, MATA is introduced, a ML task framing language that offers greater versatility compared to Trane [78]. MATA removes the restriction of being applicable exclusively to time-series data, making it suitable for a wide range of data types.
- **Creation of a data corpus for automated task discovery**: A data corpus comprising 256 labeled datasets was created. Leveraging the MATA language, each dataset is labeled with potential ML task frames that can be generated for that dataset.
- **Development of a data-driven system for automated task discovery**: For prediction of supervised ML tasks a ML approach based on the column type annotation framework TURL [24] is developed. The model achieves a macro F1-score of 0.847 on the test set.

1.4 Structure of the document

The document is structured with the following chapters:

- **Chapter 2 - Background:** In this chapter an overview is given about the concepts necessary to understand the approach presented in the rest of the work.
- **Chapter 3 - Structured representation of ML tasks:** In this chapter MATA is presented and compared to Trane [78].
- **Chapter 4 - Constructing a data corpus for automatic task discovery:** This chapter describes the process of creating the data corpus for automated task discovery and presents the results of it.
- **Chapter 5 - Modeling of automatic task discovery** This chapter describes the methods that are used to develop the data-driven system for automated task discovery and presents the conducted experiments and results of modeling ATD.
- **Chapter 6 - Conclusions:** Finally, the main findings are summarized together with their limitations and future work is outlined.

Chapter 2

Background

In the following sections, the foundational concepts that are used in the methods of this work are presented. An in-depth explanation of the entire ML-lifecycle offers a comprehensive overview of the field of application of this work. Additionally, since this work is based on NLP, an introduction of NLP and its most prominent deep learning architectures and the applied column type annotation frameworks is given. Finally, a summary of related work is presented.

2.1 Stages of the ML-lifecycle

AutoML aims to automate the inner ML lifecycle, which is illustrated alongside the outer ML lifecycle in Figure 1.1. In this section, explanations for the individual stages and sub-stages are given and relevant automation approaches are discussed.

2.1.1 Task framing

Data scientists are required to gain a deep understanding of the specific problem at hand by engaging in discussions with experts within their respective fields. Once the data is gathered, the crucial step is to frame a task that can be effectively addressed through ML techniques. The process of task framing often involves extensive iterations and collaboration between data scientists and domain experts, as they must explore various possibilities and ensure that the data requirements are consistently met before reaching a decision. Currently, this process is largely manual and lacks a clearly defined endpoint. Due to potential communication gaps and differing expectations between data scientists and domain experts, this iterative process can become labor-intensive [44].

2.1.2 Data preparation

Data preparation constitutes the second stage in the ML lifecycle. This stage is further divided into three parts: *data collection, data cleaning*, and *data augmentation*.

Data collection

Creating a new dataset or expanding an existing one often starts with the step of data collection. The most intuitive approach involves conducting web searches to find relevant datasets. Existing dataset repositories such as Kaggle[1], Elsevier Data Search[2], and Google Dataset Search[3] provide valuable resources for this purpose. However, it is common for existing datasets to fall short of meeting the specific needs of data scientists, hence requiring enhancements. This can be achieved through techniques like web scraping, as demonstrated by Chen et al. [17] and Fleet [96], who improved their image datasets using this approach.

Searching the web, however, can present challenges, such as encountering non-exact matches that require further data filtering. Moreover, data obtained from the web is often unlabeled, so either automatic or manual labeling efforts [38] are needed.

Enhancing a dataset is not limited to web searches; data synthesis is another option. One approach involves the use of generative adversarial networks, which have been employed to generate various types of data, including tabular data [73], language [26], and images [34].

Data cleaning

Collected data often contains noise, which can impact the training of ML models, so data cleaning [18] must be applied. While data cleaning was traditionally a manual process, there has been a shift towards automation in recent times. Data cleaning frequently requires domain-specific knowledge, making the involvement of domain experts essential. However, finding domain experts for this task can be challenging, leading to the development of *Katara* [19], a crowd-powered data cleaning system that relies on knowledge-based approaches. *Katara* was further enhanced by Krishnan et al. [47], who cleaned only a subset of the original dataset and defined a logic for the cleaning process of the entire dataset. This adaptation maintained performance comparable to the previous approach.

Another innovation from the same authors, *Alphaclean* [48], transforms automatic data cleaning into a hyperparameter optimization problem, reducing its dependence on human efforts. *Alphaclean* constructs a search space with various data cleaning methods, which are then evaluated and combined into a final cleaning pipeline. These techniques are particularly valuable when dealing with datasets of fixed sizes. However, in dynamic business environments where new data is continuously added, data cleaning algorithms must also adapt to evolving needs. To address this challenge, Mahdavi et al. [58] have developed a data cleaning workflow orchestrator. This tool learns from past tasks and recommends new cleaning workflows that align with emerging requirements.

[1] https://www.kaggle.com
[2] https://www.datasearch.elsevier.com/
[3] https://datasetsearch.research.google.com

Data augmentation

As previously discussed in Section 2.1.2, data augmentation was presented as a means of data collection. However, it is also employed as a regularizer to mitigate overfitting during the training of ML models. In this section, data augmentation techniques are inspected in greater detail. A useful way to categorize data augmentation techniques is by the data format they are designed for, which can be tabular, text, or image data.
A fundamental technique for tabular data augmentation is SMOTE [16], which uses a nearest-neighbor logic to either oversample minority classes or undersample majority classes. A more recent approach to tabular data augmentation is DeltaPy [82], which applies a workflow for tabular data augmentation involving transformations, interactions, mappings, extractions, and synthesis of data. This approach aims to cover a wide search space. Additionally, autoencoders are leveraged to generate new data. Demir et al. [22] utilized this technique to synthesize time series data, while Islam et al. [42] simulated crash data using Deep Learning (DL) methods.
Image data augmentation encompasses a wide range of techniques for artificially modifying original images. Basic operations include scaling, reflection, rotation, and random cropping. Elastic modifications account for brightness shifts, contrast shifts, channel shuffling, and blurring, while more complex alterations involve image blending and random erasing. These image data augmentation operations are readily available in free software frameworks such as OpenCV [14], scikit-image [87], or torchvision [60].

Additionally, numerous DL approaches have been developed for image data augmentation. For instance, Mikolajczyk and Grochowski [68, 69] employ adversarial noise and neural style transfer methods, while Antoniou et al. [5] utilize generative adversarial networks.
Textual data augmentation shares similarities with image data augmentation and can be categorized into basic data augmentation operations and advanced data augmentation approaches. Wong et al. [95] introduced two methods for enhancing textual datasets. The first method, known as data warping, generates additional samples by applying transformations in the data space, while the second method, synthetic oversampling, accomplishes a similar goal but in the feature space. An intriguing approach was taken by Yu et al. [100], who applied a back-translation function to enhance reading comprehension. For basic text augmentation, an open-source library commonly used is NLPAug [56], which offers a wide range of different transformations suitable for textual data.

2.1.3 Feature Engineering

Data scientists understand that the highest possible performance of a ML model is determined by the underlying data, particularly the quality of the data and the features constructed from it. Karmaker et al. [44] define *feature engineering* as:

> *Feature engineering is the process of transforming raw data into features that can better represent an underlying problem for computational predictive models, resulting in improved model accuracy on unseen data.*

According to Motoda et al. [70], feature engineering consists of the subtasks *feature selection, feature extraction*, and *feature construction*. Their definition points out that *feature extraction* reduces the dimensionality of the original feature space by using mapping functions, *feature selection* drops redundant features for decorrelation purposes, and *feature construction* expands the feature set by applying use case-specific feature construction procedures. Automating feature engineering involves a mix of these three subtasks for classical ML. On the other side, DL models are capable of learning the feature engineering process by themselves, which is one of the reasons for their success in recent years [38].

Feature selection

Feature selection *selects* a subset of features. This leads to simplifying the model while increasing its explainability, avoiding model over-fitting, and possibly improving model performance. A good set of features is uncorrelated to itself but highly correlated to its original data values. Algorithms for creating feature subsets are clustered into random search, heuristic search, and complete search methods. The most common random search strategies are simulated annealing and genetic algorithms. Heuristic search includes bidirectional search, sequential backward selection, and sequential forward selection, whereas complete search includes branch and bound search, best-first search, beam search, and breadth-first search [21].

Feature construction

Feature construction is the heart of feature engineering as it embodies the process of remodeling the original data into a feature space that strengthens the robustness and generalizability of a ML model and increases the representative value of the original data. Creating an informative feature space was strongly bound to expert knowledge but has already become part of the automated repertoire of data science procedures. Basic operations of feature construction include data standardization and normalization, as well as feature discretization. Successive transformation operations are dependent on the data type of the original data. Common transformations for boolean features are conjunctions, disjunctions, or negation; for numerical features, aggregation with descriptive statistics like minimum, maximum, mean, mode, or standard deviation, as well as other operations like addition and subtraction, are often used [38]. Features that hold natural language can be made machine-understandable by applying natural language processing techniques

such as bag-of-words [79] or TF-IDF [43] representations. To automate the feature construction process, research [83, 30] has already been presented that achieves the same or better performance than humans.

Feature extraction

Feature extraction reduces the feature space by transforming the current feature space with a mapping function into a lower-dimensional feature space. This differentiates *feature extraction* from *feature selection*, where the latter does not modify the original feature space. Popular feature extraction techniques include principal component analysis, isomap, independent component analysis, linear dimensionality reduction, and nonlinear dimensionality reduction [70]. These traditional methods are extended with several feature extraction approaches using feed-forward neural networks. Principally, the output of a tailored neural network serves as a new set of features. Similarly, Meng et al. [66] used auto-encoder powered models that take into account the relationships of the original features.

2.1.4 Model generation

After the feature engineering stage, a suitable ML model is trained and evaluated. The process of ML model generation is divided into the sub-stages: machine learning, alternative models exploration, hyperparameter tuning, and evaluation.

Machine learning

The ML stage is the heart of each ML pipeline: A learning algorithm is applied to infer patterns that are present in the data. One distinguishes between traditional ML algorithms and DL algorithms. Among traditional ML algorithms are *decision trees, support vector machines, k-nearest neighbor*, or *naive Bayes*, to name a few.

In the last decade, deep neural networks have received enormous attention, primarily due to their peak performance in diverse classification and regression tasks. Subforms of neural networks that are applied in various applications include convolutional neural networks, long short-term memory networks, or transformer-based architectures [44].

Existing frameworks that offer implementations for ML and DL models are Scikit-learn [75], TensorFlow [61], or PyTorch [74].

Alternative models exploration

ML tasks are not limited to being solved by a single algorithm only. Instead, different algorithms are capable of solving the same task, but certain ones might perform better than others. The exploration of alternative models aims to find the ML algorithm that yields the highest performance on a given problem. Lippman [52] proposed *Data-Driven*

Discovery of Models (D3M), which is structured into two phases. During the first phase, the framework builds models on empirical problems for which the complete data plus the solution generated by experts are given. In the second phase, the framework is tested by finding appropriate models for unsolved ML tasks. Thornton et al. [85] combined alternative models exploration with hyperparameter tuning and used Bayesian optimization to find candidate models. They utilized the existing WEKA [36] ML framework to instantiate new models with the configurations that resulted from the discovery process. This work resulted in the creation of the AutoWEKA framework [85].

Neural networks have received significant attention in alternative models discovery [77, 7, 101]. Researchers are striving to find optimal architectures for neural networks, a process known as Neural Architecture Search (NAS). A prominent example is Zoph and Le [104], who trained a recurrent neural network to predict good architectures for convolutional neural networks. They fine-tuned the model using a policy gradient method and incorporated *set selection type attention* to enable branching layers or skipping connections.

Hyperparameter tuning

Hyperparameter tuning is one of the most challenging tasks for a data scientist. Accordingly, a vast body of research [9, 11, 10, 81, 57] has contributed to the automation of hyperparameter tuning. Neural networks involve numerous hyperparameters that require optimization. In the initial experiments of Bengio and Bergstra [10], a defined hyperparameter grid space with random search was applied to discover promising combinations. They soon realized that the search space was so large that it became impractical to explore every possible hyperparameter combination. Instead, by exploring a random sample of parameter combinations, they achieved performances that were suitable for most use cases while reducing computing time compared to pure grid search.

Subsequently, the same group of researchers [11] leveraged the increasingly affordable computing power and proposed a sequential model-based approach. This greedy algorithm enabled the execution of more trials and led to the discovery of better hyperparameter configurations than the random search approach. Snoek et al. [81] introduced a Bayesian optimization approach, demonstrating its effectiveness in finding appropriate hyperparameters for ML models. Bayesian optimization relies on two critical components: a surrogate model and an acquisition function that determines which hyperparameter configuration to explore next. Additionally, they argued that optimizing the underlying Gaussian process while treating it as fully Gaussian was advantageous.

Maclaurin et al. [57] employed stochastic gradient descent-based optimization with momentum to discover improved hyperparameter configurations. They managed to minimize computational expenses while obtaining state-of-the-art results.

Evaluation

Determining the performance of ML models involves training the model on a training dataset and subsequently evaluating the fitted model on a testing dataset. Model performance is typically assessed using various metrics such as the Area Under the Curve (AUC), accuracy, or F1-score for classification tasks, and metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or R^2 for regression tasks, among others.

To address a wide range of regression and classification tasks, Olsen et al. [72] curated a dataset collection comprising 162 datasets for classification and 255 for regression. They systematically evaluated 13 different machine learning models from scikit-learn [75] on this dataset collection. Additionally, they provided metadata regarding the predictive performance of each dataset and its associated meta-features. This dataset collection includes both toy datasets and real-world datasets, making it diverse and highly valuable for evaluation purposes.

Similarly, Duan et al. [28] established a benchmarking dataset collection for reinforcement learning. This collection encompasses datasets ranging from simple to highly challenging tasks; some include hierarchical structures or partial observability. State-of-the-art reinforcement algorithms were successful in identifying goal-leading policies in most tasks but faced difficulties when tasks exhibited higher hierarchies.

2.1.5 Result summary/recommendation

A data scientist must be able to bridgeing the knowledge gap between their technical expertise and domain knowledge [92]. This bridge is essential for evaluating a ML model in the context of a given business objective. To achieve this, a data scientist must understand both the capabilities and limitations of a model. A ML model can be assessed on various levels, including computational overhead, features, and the model itself. It falls upon the data scientist to extract meaningful insights from these assessments and communicate them effectively to the business side.

However, this step is currently purely manual work and there have not been any automation approaches proposed yet [44].

2.2 Natural language processing

NLP is an artificial intelligence field that specializes in the engagement of humans with computers through natural language. The main objectives of NLP are to model language generation, perception, and understanding. Natural language is a complex data source. The complexity of natural language is due to the ambiguity of words and phrases in different contexts, the usage of sarcasm and humor, and language alterations through dialects and slang. The fact that the meaning of words is strongly dependent on the surrounding

context and that through this the interpretation of word changes as well makes it a challenging task to model natural language [86].

In the last years, big advancements could be achieved in NLP because of the availability of large data corpora, the decrease in the price of high-power computing and enhancements in machine learning techniques like artificial neural networks and especially transformer architectures. Transformers are especially well suited for NLP due to their high ability to capture complex data patterns and long-range dependencies. With the pretraining-finetuning paradigm, open-source large language models such as BERT [25] or GPT [15] achieved a lot of state-of-the-art performances in different NLP tasks.

Some common NLP tasks include [86]:

- **Question Answering:** NLP models can be trained to answer questions based on specific contexts or document sets. This technology can be found in chatbots, virtual assistants, and search engines.
- **Named Entity Recognition (NER):** NER involves identifying and categorizing named entities in text, such as names, locations, organizations, and dates. It is often employed for data extraction and text mining.
- **Machine Translation:** NLP models can be trained to translate text from one language to another, requiring an understanding of the source language and the ability to produce an equivalent translation in the target language.
- **Text Classification:** Text classification involves categorizing text documents into predefined categories or classes. This task is commonly used for spam detection, sentiment analysis, and topic classification.
- **Text Summarization:** Text summarization is used to summarize long text documents concisely. It is frequently used for summarizing news articles, research papers, and legal documents.

NLP is also commonly used for table representation tasks like entity linking, column relation extraction and column type annotation [24, 41, 102, 90, 93]. Tables contain natural language and numerical data, so that NLP techniques can be applied to extract information from tables. In this work, ATD is formulated as a column type annotation task for which column type annotation frameworks are used that treat columns as natural language text.

2.3 Deep learning architectures

In this section the concepts of Artificial Neural Network (ANN) and transformers are explained, which are the basis working mechanisms of the column type annotation frame-

works TURL [24] and Sato [102].

2.3.1 Artificial neural networks

ANN are a computational abstraction inspired by the human brain. The first model of a neuron was introduced in 1943 by McCulloch and Pitts [63], laying the foundation for modern ANNs used in AI. In this section, the fundamental concepts of ANNs are explained, with the concepts provided by Aggarwal et al. [2].

The simplest form of an ANN is a single neuron, often referred to as a SLP. Figure 2.1 illustrates the operational mechanism of a SLP. An n-dimensional input vector X is input into the SLP, where it is multiplied by the weight vector W and then summed with the bias b. The resulting value is processed by an activation function φ, which generates the output y. A common choice for φ is the sigmoid function. The activation and output functions are defined as follows:

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

$$y = \phi(b + \sum_{i=1}^{k} W^T X) \tag{2.1}$$

The activation function is designed to be non-linear, enabling the model to learn complex patterns. However, the SLP remains a linear model and is capable of solving only linearly separable problems. Addressing non-linear problems requires the assembly of multiple SLPs to create a Multi Layer Perceptron (MLP). An illustrative example of an MLP is presented in Figure 2.2. A MLP has an input layer, one or more hidden layers, and an output layer. Each layer consists of multiple neurons, with every neuron connected to all neurons in the subsequent layer. When all neurons in a layer are connected to all neurons in the following layer, this layer is referred to as a fully connected layer.

To obtain an output from the network, a forward pass is executed. During the forward pass, the input vector X is transformed in each neuron of the first layer of the MLP, as described by Equation 2.1. The output of the first layer is then passed to the next layer, and this process continues until the output of the last layer is produced, serving as the MLP's final output.

To train the MLP, neuron weights are initially set randomly, and the output of the MLP is compared against the ground truth. The difference between the MLP's output and the ground truth is then used to calculate how to update the neuron weights, a process known as backpropagation. For backpropagation, a loss function is defined, which calculates the error between the MLP's output and the ground truth. A commonly used loss function for neural networks is the MSE, defined as:

$$L = MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

The goal is to adjust the weights W and biases b to minimize the loss between the output of the MLP and the ground truth. To achieve this, the gradient of the loss function for each weight and bias is calculated using the following equation:

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}}$$

Here, L represents the loss function, y_j denotes the output of the j-th neuron, z_j is the weighted sum of inputs to the j-th neuron, and w_{ij} represents the weight connecting the i-th neuron to the j-th neuron. By traversing backward through the MLP and calculating gradients for all parameters, the weights and biases of the network are updated using the following update equation:

$$w_{ij} = w_{ij} - \eta\frac{\partial L}{\partial w_{ij}}$$

In this equation, η represents the *learning rate* of the optimizer, determining the size of the weight and bias update steps. By iteratively executing the forward and backward passes for the records in the dataset, the MLP progressively converges towards a minimum of the loss function. This iterative process is commonly known as Stochastic Gradient Descent (SGD).

In addition to the described training process for MLPs, more complex neural networks have been developed and trained. Prominent derivatives of classical ANNs include Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long-Short Term Memory (LSTM), and transformer-based architectures.

CNNs find extensive use in image classification tasks, characterized by the presence of convolutional layers and pooling layers that enable them to process image data effectively.

RNNs, on the other hand, are well-suited for handling sequential data and are composed of recurrent layers designed for this purpose. LSTMs, a specialized type of RNN, are particularly effective in managing sequential data, offering improved performance in tasks that involve long-range dependencies.

Transformer-based architectures rely on self-attention mechanisms and are primarily employed in natural language processing tasks, where they have demonstrated remarkable effectiveness.

To date, various neural network approaches have achieved outstanding performance in diverse domains, such as image classification [49, 103], natural language processing [25, 20], and unsupervised generative challenges [73, 45].

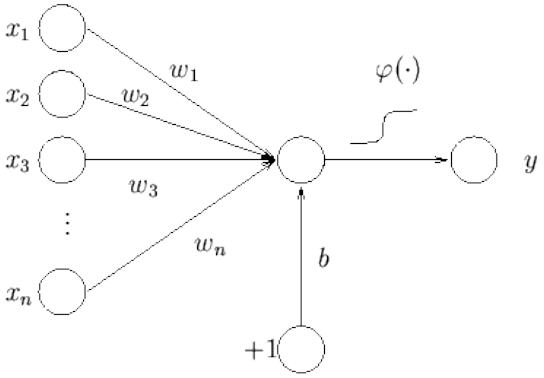

Figure 2.1: Illustration of a SLP. Source: [6]

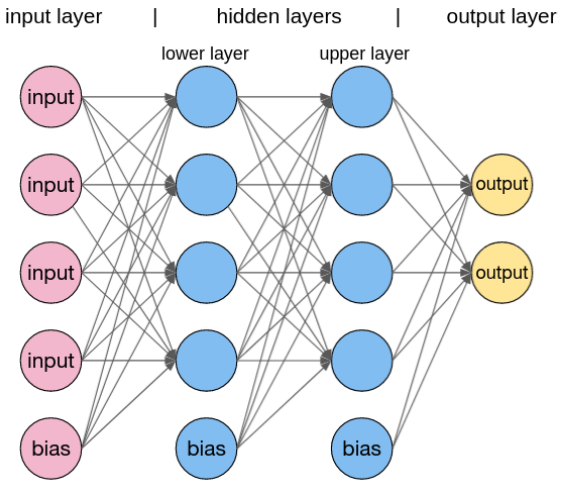

Figure 2.2: Illustration of a multi-layer perceptron. Source: [4]

2.3.2 Transformers

The concept of transformers was initially introduced by Google Research in 2017 [88]. Since then, extensive research efforts [25, 55, 15, 8] have shown the high potential of transformers across various domains. In this section, the fundamental mechanisms of transformer architectures are presented, as explained in the original paper [88].

With the advent of transformers, DL methods such as RNNs and CNNs have been overshadowed, primarily due to the significantly improved performance of transformers while being computationally more efficient. The core architecture of transformers consists of an encoder and decoder network, both employ multi-head self-attention blocks. This architecture is depicted in Figure 2.3.

The working mechanism of the self-attention blocks is based on computing the dot-product attention with the formula:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

In this formula, Q, K, and V denote the *query*, *key*, and *value*, respectively. Given a sentence, for each word (query), the dot product (similarity) is computed for all other words (*keys*) in the sentence. These similarity scores are then scaled by $\sqrt{d_k}$, where d_k is the dimensionality of the *keys* embedding, and are transformed into a weight vector using the $softmax$ function. The *values* embedding corresponds to a vector representation of all words in the given sentence. By multiplying the *values* by the previously calculated weights, each word in the sentence is assigned an attention score, indicating how much attention should be paid by the *query* to each word in the *values*. This process is illustrated using an example sentence in Figure 2.4.

In the multi-head attention block, h attention heads are used to generate h distinct linear projections. These linear projections are then concatenated and passed forward to a feed-forward network equipped with a single hidden layer. The original transformer architecture includes $N = 6$ of these blocks, each with $h = 8$ attention heads, in both the encoder and decoder components of the model.

To train the transformer for the task of natural language translation, where English sentences are translated into German, each record consists of a pair of English sentence and its corresponding German sentence. The English sentence is fed into the encoder, while the decoder is initialized with the German sentence. Leveraging the self-attention mechanism, the model learns which words in the English sentence hold high *attention* weights for specific words in the German sentence. The training process involves shifting the decoder input one word to the right iteratively, training the decoder to predict the next word in the German sentence at each step. This process continues until the decoder predicts the end-of-sentence token.

Transformer architectures are not limited to language translation, but can be applied

to a wide range of different tasks including question-answering, text summarization and image captioning. In this work, the transformer-based *column-type annotation framework* TURL [24] is used to learn table representations and is explained in Section 2.4.1.

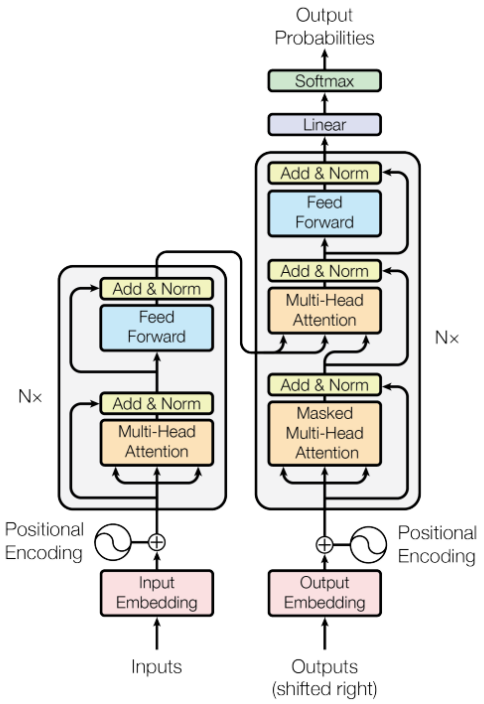

Figure 2.3: Original transformer architecture. Source: [88]

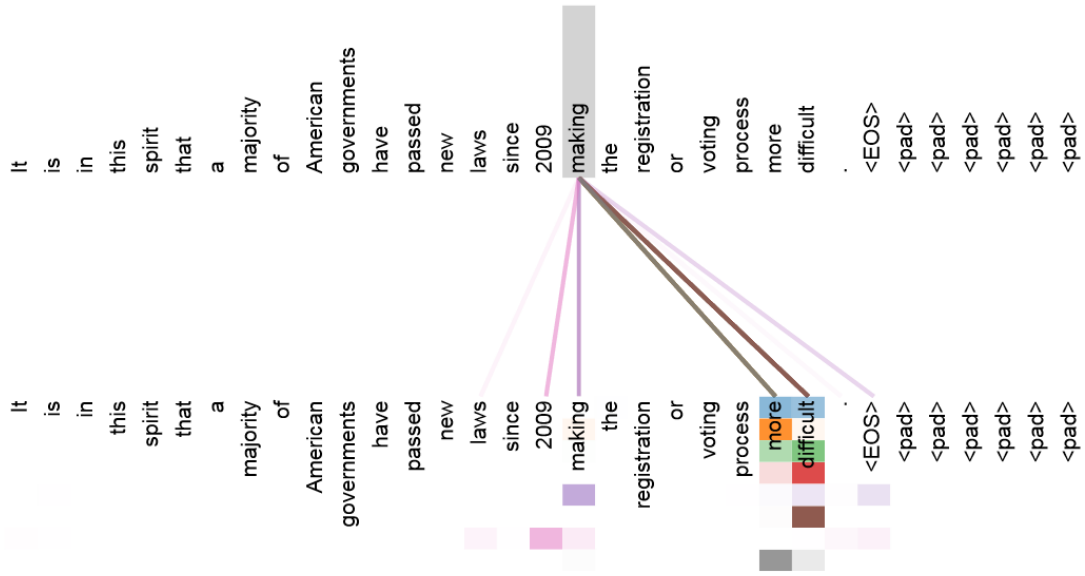

Figure 2.4: The self-attention mechanism detected, which words in the given sentence are dependent to the word *making*. Source: [88]

2.4 Column type annotation ML frameworks

In literature, several column annotation frameworks have been proposed, which build the state-of-the-art [84, 102, 41, 31, 90, 24, 93]. Many of these frameworks do not take into account table metadata for learning, which is expected to have potential value for ATD. TURL [24] stands out as the state-of-the-art approach among those that leverage metadata to enhance training data, making it the first candidate column annotation framework for ATD.

Sato [102], on the other hand, is a column annotation framework having a highly sophisticated feature set. The richness and expressiveness of Sato's features make it an interesting second candidate for ATD

2.4.1 TURL [24]

TURL is a transformer-based framework for table representation, pretrained on a dataset comprising 570,000 relational tables from Wikipedia. Its objective is to generate representative embeddings for both metadata and *semantic entities* to effectively model factual knowledge. The metadata of a table encompasses elements such as the *page title, section title, table caption*, and *column names*, while the semantic entities are real-world concepts mapped to a knowledge graph like Wikidata.

TURL introduces a novel pretraining objective namely *masked entity recovery* for table representation tasks. Given a table, random entities within the table are masked, and the framework is trained to predict these masked entities. The authors propose the use of a visibility matrix as a masking mechanism within the self-attention layer of the transformer architecture. This visibility matrix defines which parts of a table are visible to one another. For instance, a table cell is visible to its neighboring cells, but the table caption remains invisible to a table cell.

Following unsupervised pretraining with the masked entity recovery objective, the framework is fine-tuned on six downstream tasks. These tasks encompass table interpretation, including entity linking, column type annotation, and relation extraction, as well as table augmentation tasks like row population, cell filling, and schema augmentation. In the context of the column type annotation task, TURL has demonstrated superior performance compared to Sherlock [40], highlighting its exceptional capabilities. TURL is also available as open-source software[4]. .

2.4.2 Sato [102]

The second framework employed in this study is Sato [102], a hybrid neural network combined with structured learning designed to predict semantic types for table columns, with

[4] https://github.com/sunlab-osu/TURL

a focus solely on tabular data without utilizing any metadata. The authors of Sato trained their model using a dataset comprising 80,000 tables extracted from the WebTables corpus of VizNet [39].

The architecture of Sato comprises a single-column model, for which the authors utilize Sherlock [40] in combination with Conditional Random Fields (CRF) to incorporate contextual information from other columns into the predictions. Sherlock is a neural network-based model that relies on feature engineering, including features such as *character embeddings, word embeddings, paragraph embeddings*, and *column value statistics*. Sato extends Sherlock's feature set by introducing a *global context vector*, created using Latent Dirichlet Allocation (LDA) [13], which captures information from all table values.

The single-column model in Sato is capable of predicting the semantic type of a single column, without considering its neighboring columns. To address this limitation, the authors define column type annotation as a structured prediction problem and propose the use of a linear chain CRF to model the conditional co-occurrence of columns within a table. In this approach, only adjacent columns are taken into account, which means that the model is unable to capture longer-distance relations. The input for the CRF consists of the unary potentials for columns being certain semantic types, which are the direct outputs of the single-column model. The resulting semantic type for a column is determined as the semantic type with the highest probability, considering both the unary potentials and the pairwise potential functions.

Sato demonstrates superior performance compared to Sherlock, particularly excelling in predicting data types that occur less frequently within their dataset. Sato is also available as open-source software[5].

2.5 Related Work

Reviewing literature has shown that only two papers can be considered relevant to the objective of this work. Schreck and Veeramachaneni [78] introduced Trane, a language designed to frame ML tasks. Xu et al. [98] presented MLFriend, a machine learning task frame recommendation system.

2.5.1 Trane [78]

Schreck and Veeramachaneni were the first to introduce a ML Task Frame Representation Language (TFRL). They created Trane, which was particularly designed for handling time-series data. Trane enables data scientists to structure machine learning task frames in JavaScript Object Notation (JSON) format, with the properties of *entity*, *time-line segmentation*, *target*, as well as *column aggregations*, *row transformations*, and *filter* operations.

[5] https://github.com/megagonlabs/sato

The *entity* property defines the object represented by an individual record, while *timeline segmentation* captures aspects like *lead* and *lag* time periods, which are essential for prediction tasks with time-dependent data. The *target* refers to the dependent variable to be predicted. Additionally, *column aggregations* enable the grouping of columns based on discrete column values, followed by the application of aggregation functions. *Row transformations* provide the ability to transform one or multiple columns by applying specified functions, and *filters* are used to eliminate records that do not meet specific conditions of the task frame.

An interpreter, based on Pandas [64], is responsible for translating the task frame into actual data transformations and training of ML models. In this work, they used the Walmart Stores dataset[6] for demonstration and constructed a total of 1.077 ML task frames for this dataset, with only a small subset of them proving to be meaningful.

The major limitation of this work is that it can only be applied to time-series data. Thereby, a lot of ML task frames cannot be constructed. However, it serves as an inspiring foundation for the development of a more versatile and general-purpose TFRL.

2.5.2 MLFriend [98]

Building upon the foundations of Trane, MLFriend is the first approach in automating the ML stage of *task framing*. MLFriend operates as a human-in-the-loop ML task frame recommendation system. Data scientists can engage with MLFriend interactively, progressively narrowing down the most relevant tasks for a given dataset. During each iteration, data scientists rank the generated ML tasks, and MLFriend learns from these rankings to recommend tasks in the following iterations that are similar to the higher-ranked tasks from the prior iteration.

To evaluate the effectiveness of MLFriend, the authors employed datasets such as the Chicago Bicycle dataset[7], Flight Delay dataset[8], and YouTube Trending dataset[9]. MLFriend managed to discover all possible task frames within these datasets. However, only 23.6%, 20.0%, and 53.1% of these task frames proved to be meaningful. The determination of what constitutes a meaningful task frame was adjudicated by nine expert data scientists.

One shortcoming of MLFriend is, that only time-series task frames can be generated. Additionally, MLFriend is computationally very expensive since it generates evaluation scores of each recommended task frame by training ML models on the dataset.

[6] https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting
[7] https://www.kaggle.com/yingwurenjian/chicago-divvybicycle-sharing-data
[8] https://www.kaggle.com/usdot/flight-delays
[9] https://www.kaggle.com/datasnaek/youtube-new

Chapter 3

Structured representation of ML tasks

In this chapter, the TFRL MATA is introduced and compared to Trane [78].

3.1 Task frame representation language MATA

Assigning task frames to datasets requires a uniform machine learning TFRL with enough expressive power to construct a wide range of ML task frames. In 2016, Schreck and Veeramachaneni [78] introduced Trane, the pioneering approach to structurally representing ML task frames. However, Trane is limited solely to time-series data task frames. This constraints the creation of ML task frames that do not have a time dependency.

The present project builds upon the concepts of Trane and supplements them to create the supervised **MA**chine Learning **TA**sk representation language **MATA**. MATA structures supervised ML task frames using properties such as *entity*, *target*, and *ML-type*, along with operations such as *filtering*, *column aggregation*, and *row transformation*. To illustrate the concept of MATA, a hands-on translation of the following natural language task frame into MATA format is provided:

> The task is to build a *regression model*, to predict the *mean price per square-foot* for apartments in the San Francisco neighborhood *Russian Hill*, for *each day*.

The *entity*, or index, of a task frame, represents the concept of a single record. In the example task frame provided above, the entity corresponds to the *date* as the data is aggregated *for each day*. In cases where a task frame does not require aggregation of records, the entity is referred to as the *root entity*. Given the *Housing Prices in San Francisco* dataset, the root entity or in simpler terms one record of the dataset corresponds to *an ad of an apartment in San Francisco*.
The *target* of a task frame equals the dependent variable in statistical prediction tasks. Due to the transformations and aggregations applied within this task frame, the target variable is the *mean price per square foot*. MATA represents aggregated targets using the

symbol *AGG*. It's worth noting that targets may not always require modification through aggregation or transformation operations; in different scenarios, the target could simply be the *price* column itself.

Defining the *ML-type* of a task frame involves analyzing whether the target of the task frame comprises discrete or continuous values. If the target is discrete, the *ML-type* is categorized as *classification*; if it is continuous, it is labeled as *regression*. In the case of the target being *mean price per square foot*, which consists of continuous numerical values, the *ML-type* is *regression*.

Datasets often contain records that are irrelevant to a given task frame. Through the use of *filter operations*, records that do not meet the conditions specified in the *filter operations* are excluded from the dataset. MATA uses comparators such as $<, >, \leq, \geq, =$, logical operators $\|, \&, !$, and *NONE* in case no *filter operations* should be applied. In the provided example, the dataset is filtered based on the condition $hood = RussianHill$.

As previously mentioned, *column aggregation operations* are essential for aggregating a dataset based on its task frame entity. In the given task frame, the column aggregation operation is *MEAN*, and the *entity* being aggregated is each day (date). MATA provides a collection of *column aggregation operations*, including *NONE, MEAN, MIN, MAX, ALL, ANY, COUNT, SUM, ...*, to name a few. The complete list is shown in Appendix A.4 with explanations.

Row transformation operations are applied across a row of a tabular dataset and create a new column by using one or multiple columns as input. MATA encompasses basic mathematical operations such as $+, -, \times, \div$, as well as comparators like $<, >, =$, and *NONE*. Transforming the natural language task frame to MATA results in using the $\div$ operator on the columns *price & sqft*.

Furthermore, it is important to note that the first version of MATA does not support nested TFRL properties or operations. Consequently, *entities* that are constructed by chaining multiple columns are not supported. Additionally, a task frame can include a maximum of one filter, one column aggregation, and one row transformation operation. This limitation is because of the primary objective to maintain simplicity and broad applicability across a wide range of datasets.

The whole translation of the natural language task frame to MATA is listed in JSON format in Appendix A.3.

3.2 Comparison of Trane and MATA

When comparing Trane and MATA, some main differences can be observed. The most significant alteration is that MATA can also be applied to non-time-series data. In contrast, Trane is specifically designed for capturing time-series task frames, offering more granular prediction engineering features, such as the determination of lag and lead times.

Another difference is that MATA directly labels a task frame as either regression or classification ML-type, whereas Trane relies on row aggregations to transform the target column into discrete or continuous values. Real-world data is often noisy, which can hinder the direct application of row transformations. MATA addresses this challenge by accurately labeling noisy data with the correct ML-type.

Filter operations differ between MATA and Trane because MATA excludes time-dependent operations to ensure its applicability to both time-dependent and non-time-dependent data. In terms of column aggregation operations, MATA offers a broader range of statistical operations, and row transformation operations are extended to include simple mathematical operations while excluding time-dependent ones.

Furthermore, the first version of MATA is limited to not support nested properties and operations. Trane does support nesting by default, but in their experiments, only a version with the same nesting restrictions is applied. However, MATA can easily be adapted to support nesting.

In conclusion, MATA represents a more versatile framework for task frame representation compared to Trane. In the next chapter, MATA will be applied to construct a data corpus for ATD to demonstrate its high capability in framing various ML task frames.

Table 3.1: Comparison of the TFRLs Trane and MATA.

TFRL property	Trane [78] included	operations	MATA included	operations
entity	y	-	y	-
time-axis	y	time series segmentation ops.	n	-
target	y	-	y	-
ML-type	n	-	y	-
filter ops	y	$<, >, \leq, \geq, =, \|\|, \&, !$, DAY_OF_WEEK, HOUR, IDENTITY MIN, MAX, MEAN, ALL	y	$<, >, \leq, \geq, =, \|\|, \&, !$
col. agg. ops	y	FIRST, LAST, ANY, ALL, SUM, COUNT	y	ANY, ALL, SUM, COUNT, MEAN, MIN,MAX
row trans. ops	y	$<, >, =, !=$, POW, DIFF, IDENTITY_ORDER_BY	y	$<, >, =, !=, +, -, \times, \div$

Chapter 4

Constructing a data corpus for automatic task discovery

This chapter outlines the process of generating a data corpus for ATD. The created data corpus is empirically analyzed and summarized with descriptive statistics and visualizations. The results of this are presented and discussed.

4.1 Methods

The creation of a data corpus for ATD involves three steps: *data collection, data filtering*, and *data labeling*. The next sections provide detailed insights into each of these stages.

4.1.1 Data collection

ATD has not been addressed yet through a data-driven approach. Hence, the need arises to generate a diverse data corpus capable of training ML models for predicting supervised ML tasks. This data corpus should consist of tabular datasets in CSV format, because CSV is the most common tabular data format. Each of these datasets should be mapped to all potential supervised ML tasks that can be formulated for these datasets.

Several sources are available for generating this data corpus, including dataset collections such as Kaggle, Elsevier Data Search, and Google Dataset Search. Table 4.1 provides a comparison of these dataset collections, considering the number of available datasets and the availability of an API. According to this listing, Elsevier Data Search offers a limited number of datasets, therefore it is excluded from this work. Google Dataset Search offers a large number of datasets but lacks an API necessary for automating the data extraction process. Kaggle is the most prominent dataset collection for ML projects and offers a vast and diverse dataset repository along with a comprehensive API, enabling programmatic access to the data catalog. Therefore, Kaggle is selected as the dataset collection for creating the data corpus for this study.

Table 4.1: Listing of dataset collections and their properties.

Data catalogue	# of datasets	# of open source datasets	API (y/n)
Elsevier Data Search[1]	34	34	n
Google Dataset Search[2]	31 000 000	n.a	n
Kaggle[3]	230 000	40 000	y

The process of creating the data corpus begins by listing all available datasets on Kaggle. This yields high-level dataset metadata in JSON format, providing explicit details such as dataset IDs, names, storage sizes, and more. By iteratively traversing this list and utilizing the Kaggle API, metadata and CSV data for each dataset are downloaded. Sample metadata and CSV data responses are added in Appendix A.1 and A.2. This sample dataset is the *Housing Prices in San Francisco*[4] dataset, which will be used for demonstration purposes throughout this work.

After performing the steps above, metadata and tabular data have been gathered, which serves as raw data for the successive data filtering and data labeling.

4.1.2 Data filtering

After the creation of the raw data corpus, a diverse set of filters is applied to clean the data corpus by removing records that are not suitable for this project. The focus of these filters is on excluding datasets that are not commercially usable and ensuring that the datasets are easily processable. To achieve this, a filter is designed to retain datasets that include either an open-source license or an open-source with attribution license, thus authorizing commercial usage. Below is a list of all the licenses for which this filter is applied:

- CC0-1.0
- Attribution 4.0 International (CC BY 4.0)
- GPL-2.0
- U.S. Government Works
- ODC Public Domain Dedication and Licence (PDDL)
- Community Data License Agreement - Permissive - Version 1.0
- EU ODP Legal Notice
- Attribution 3.0 IGO (CC BY 3.0 IGO)
- MIT

[4] https://www.kaggle.com/datasets/thedevastator/scraping-apartments-off-of-craigslis

- Apache 2.0

Furthermore, the data corpus is filtered using the following custom filters to enable easier processing of the data corpus:

- **CSV Filter:** Since this work focuses on tabular data, it is necessary to filter out non-tabular data. Thus, only datasets are retained that include CSV files. Additionally, datasets that include CSV files only containing metadata, such as information about images (which serve as the primary data source), are also excluded from the filtered data corpus.
- **Storage Size Filter:** ML and especially DL require large datasets to capture data patterns. Given the need to process numerous datasets in this work, it is essential to establish a maximum dataset storage size limit. An analysis of the storage size distributions in the raw data corpus revealed that the mean storage volume was 24 MB, with the maximum storage size reaching 320 GB. Therefore, a dataset filtering threshold of 5 GB was set.
- **# of Columns Filter:** Similarly, it is necessary to impose a limit on the maximum number of columns. Certain datasets within the raw data corpus contain thousands of columns, making the labeling process unfeasible. An examination of the column distribution reveals that the average number of columns is 11. To address this issue, a threshold for the number of columns is set at 20, which lies outside two standard deviations from the mean value.
- **Multiple files filter:** In cases where a dataset consists of multiple files, a schema check is performed to determine if they share the same structure. If they do, the files are concatenated into a single dataset; otherwise, the dataset is excluded.
- **Usability filter:** Kaggle's metadata structure includes a *usabilityRating* field. Exploring datasets with both higher and lower *usabilityRating* values, it was observed that the data quality differs significantly. As a result, the threshold for the *usabilityRating* field was set to 0.9.
- **English filter:** Exploring the raw data corpus has revealed the presence of datasets in languages other than English. Non-English data could potentially have an impact on the model's performance; therefore, all non-English datasets are excluded.

4.1.3 Data labeling

Labeling the filtered data corpus involves the assignment of all potential supervised ML task frames for each dataset. These task frames can be transformed into actual ML models. Since no generally accepted definition of a supervised ML task frame could be found it is defined in this work in the following way:

> A supervised ML task frame provides a natural language description of the variable, which is the *target* of the prediction, the data structure, which forms a single record of the dataset (*entity*), whether the *ML-type* is regression or classification and which *filter, aggregation* and *transformation* operations are applied to the dataset.

An example of a natural language task frame is provided in Section 3.1. To ensure computer readability, a natural language task frame must adhere to a predefined structure. To achieve this, the novel TFRL MATA was created, which is introduced in Section 3.1. The labeling process of the filtered data corpus begins with an initial check to determine whether supervised ML can be applied. This check considers factors such as the number of rows, the semantic content, and the uniqueness of values in the dataset's columns. With these checks, a data scientist can confirm the suitability of applying supervised ML.

After that, ML task frames in the MATA format are assigned to each dataset. When formulating these task frames, all task frames are considered that predict a variable meeting the criteria of being *hard to measure*, *dependent* on other variables, and *adding value* through the automation of the prediction of this variable.

4.2 Results and discussion

The raw data corpus fetched from Kaggle initially contained 44,152 datasets. After applying the described filtering process, the original raw data corpus was reduced to a size of 2,687 datasets, which served as the input for the data labeling process. During one month, a total of 511 datasets were labeled, of which only 256 met the criteria for suitability of supervised ML. Consequently, these 256 datasets were further explored and processed for modeling.

After the labeling process, each *dataset* now comprises the *tabular data* in CSV format, a natural language description referred to as the *context*, and a list of task frames in MATA format. In the next sections, the results of the empirical analysis for each of these three components are presented and discussed.

The ***tabular data*** within the data corpus consists of 256 CSV tables, which are analyzed using descriptive statistics. Table 4.2 presents statistical measures, including the *mean*, *Standard Deviation (STD)*, *minimum*, and *maximum* values, for metrics such as *number of rows, number of columns, NaN ratio*, and *duplicate ratio*.

The number of rows exhibits significant variability across datasets, reflecting the strong natural diversity of datasets. On average a dataset has 9.448 columns, although this mean value is slightly influenced by the imposed maximum column threshold of 20 columns. Remarkably, missing values (NaNs) are infrequent within the data corpus. The

reason for this may be the preprocessing and cleaning of datasets that are publicly available on Kaggle. Similarly, duplicate values are rare within the datasets.

Table 4.2: Descriptive statistics of the data corpus.

	mean	std	min	max
# rows	23469.974	14752.624	251	48 027 403
# columns	9.448	3.853	2	20
# NaN ratio	0.013	0.002	0	0.348
# duplicate ratio	0.001	0.000	0	0.003

Likewise, the column data types are explored. Column data types are relevant to some properties of a task frame. For instance, a column with the data type *datetime* is highly likely to serve as an *entity* column within a task frame, while numerical data types like *integer* or *floating point* are more likely to be associated with the target column of a task frame. Table 4.3 presents the counts of each data type, along with the mean number of columns of a specific data type within a table. The most prevalent data type is *string*, followed by *floating point, integer, datetime*, and *boolean*. Summing up the counts of datatypes results in 2419, which is the number of all columns in the data corpus.

Table 4.3: Counts of the column datatypes of the tables in the data corpus in total and as mean per table.

data type	#	mean per table
string	914	3.570
floating point	691	2.699
integer	614	2.398
datetime	173	0.676
boolean	27	0.105
total	2419	9.448

The column names of a table serve the purpose of explaining the values contained in each column, holding valuable information for later modeling tasks. Figure 4.1 displays a bar chart showing the 40 most common words found in the column names of the data corpus. Analyzing this chart shows that the most frequently occurring words are often time-related. Despite there being a higher number of columns with *string* or *numerical* data types compared to *datetime*, it appears that the variability in column names for *datetime* columns is lower. This observation may stem from *datetime* columns not being recognized as such, causing time-related words to appear in columns of different data types.

The ***contexts*** datasets within the data corpus are natural language descriptions provided by the dataset authors, to provide additional information about the dataset. Among the 256 datasets in the corpus, 142 include a *context*. In Table 4.4, descriptive statistics

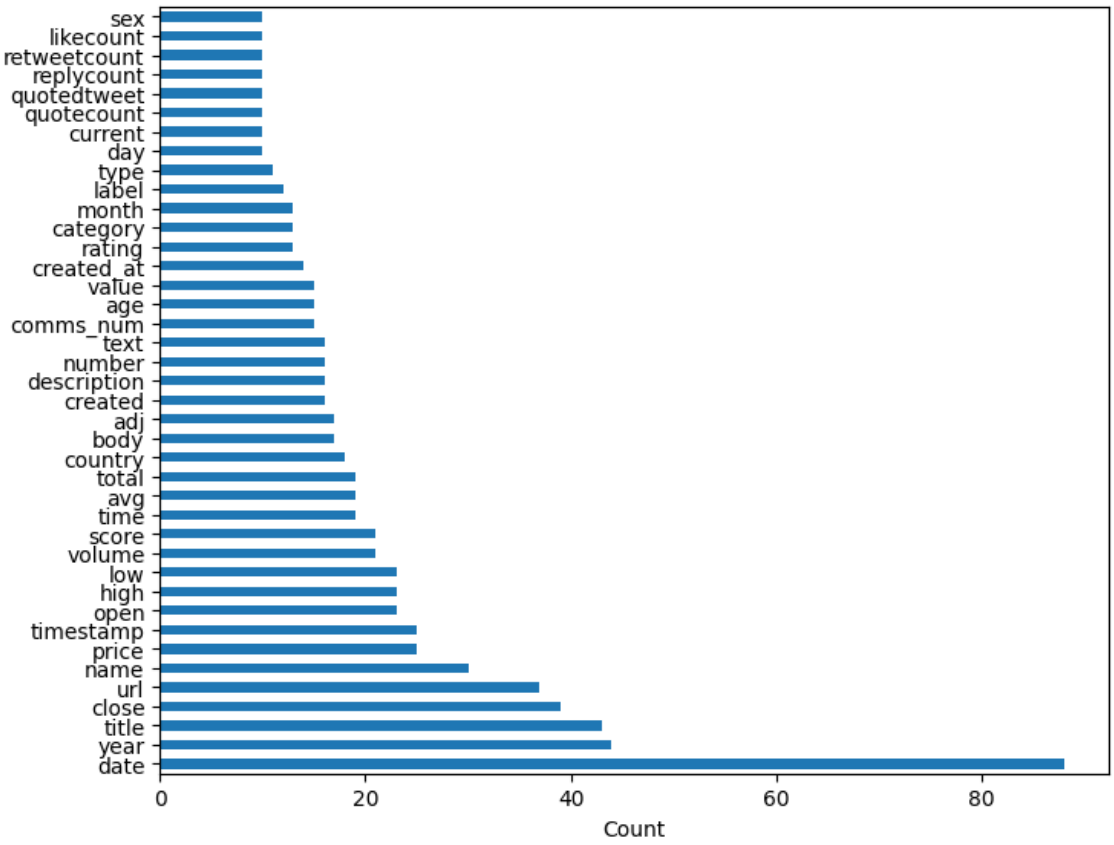

Figure 4.1: Bar chart displaying the counts of the 40 most common words in the column names of the tables in the data corpus.

summarize the character and word counts of contexts. The mean word count of 69.475 demonstrates that the majority of datasets have a sufficient document size for the application of natural language processing techniques.

The distribution of word counts is visualized in Figure 4.2, revealing that only a small subset of datasets possess a *context* exceeding 200 words in length. To explore the content of the*contexts*, Figure 4.3 presents the 40 most frequently occurring words. As expected, words such as *dataset, data, contains*, and *column* dominate the list.

Table 4.4: Descriptive statistics about the distribution of the number of words and number of characters in the context property of the datasets in the data corpus.

	mean	std	min	max
# words	65.031	69.475	0	420
# characters	762.090	827.931	0	5175

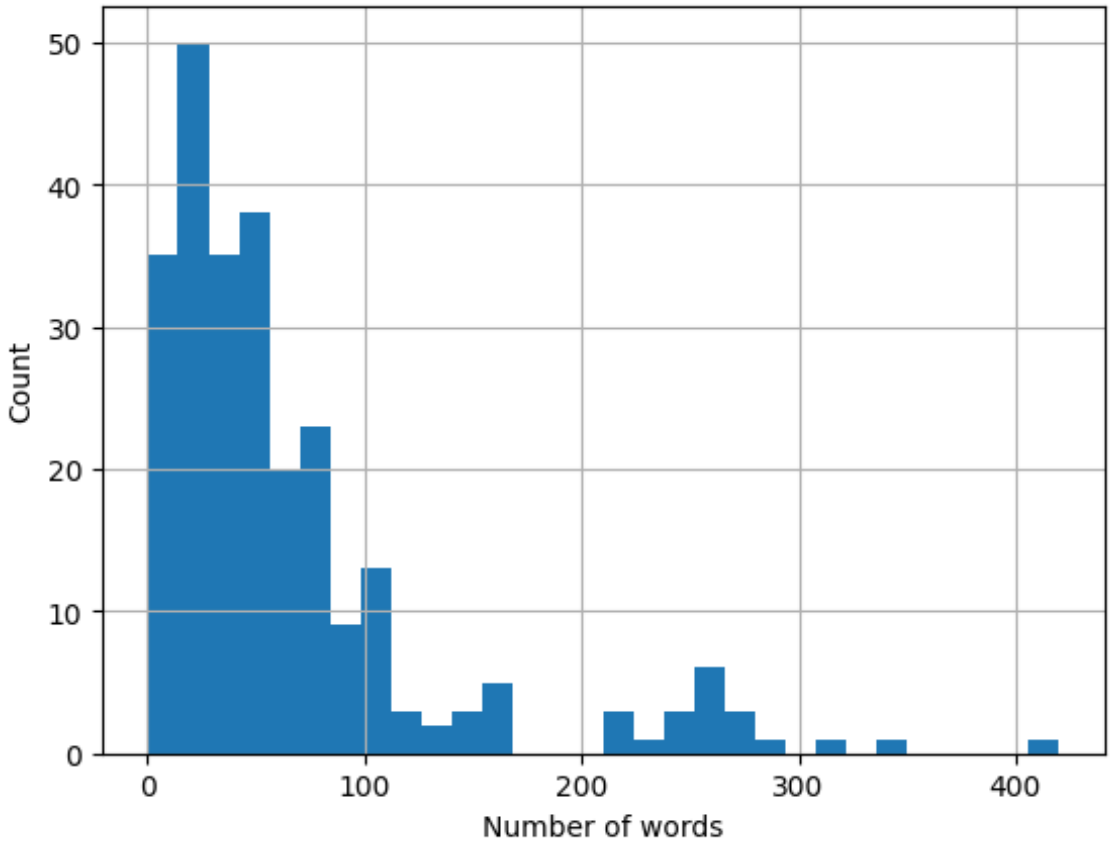

Figure 4.2: Distribution of the number of words per context of the datasets in the data corpus.

The following provides an overview of the counts of the most frequently assigned values for each property of a ***task frame*** in the MATA language. In total, 986 task frames

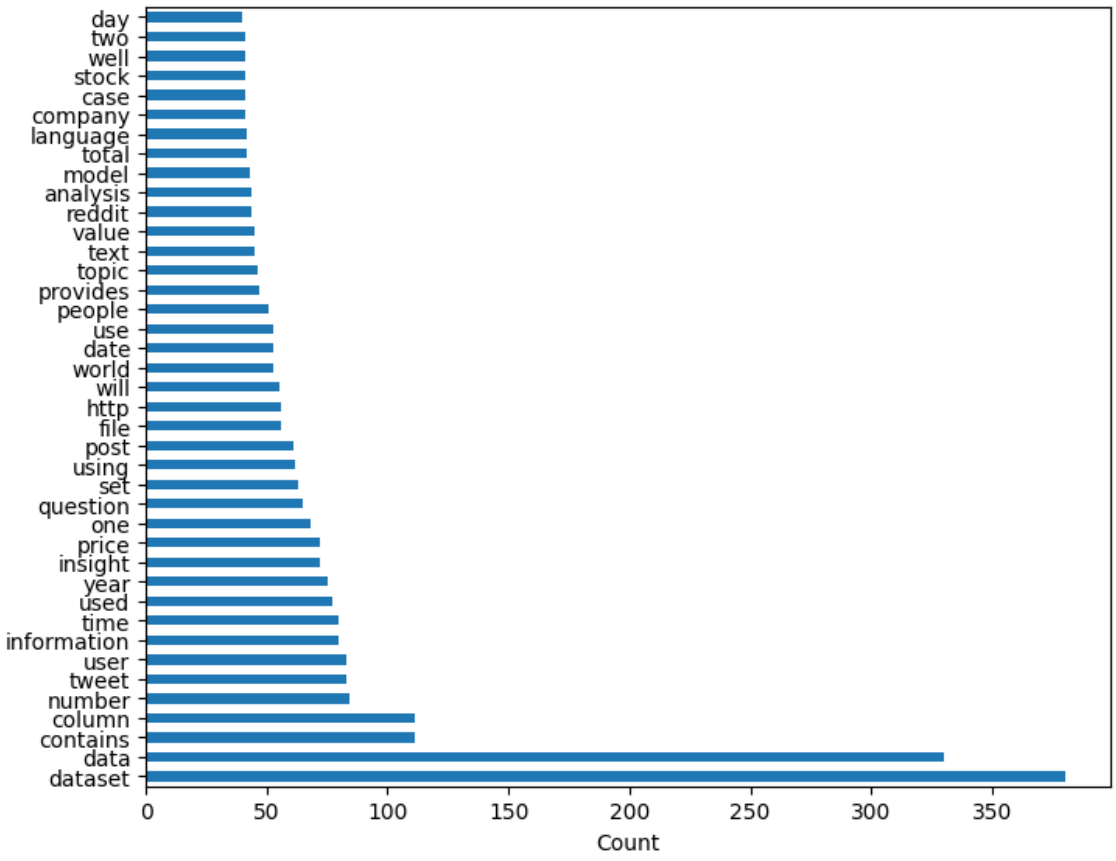

Figure 4.3: Bar chart of the 40 most common words in the context of the datasets in the data corpus.

were constructed and assigned to 256 datasets. The mean number of task frames per dataset is 2.206, with a STD of 3.103, and a range from 1 to 19 task frames per dataset.

Table 4.5 presents the counts of the most commonly occurring values for each property of a MATA task frame. Beginning with the *entity* property, the most frequent value is *root*, indicating that the root index of a CSV file is the most frequently assigned entity in the designed task frames. Apart from that, a high number of the most frequent *entities* suggest that time-related columns are often selected as *entities*.

Analyzing the *target* property reveals that *open, high, low, close*, and *volume* are the most frequently chosen target columns. These columns typically originate from stock data, suggesting that many datasets contain stock-related data.

Regarding the *ML-type* property, regression tasks are approximately seven times more common than classification tasks.

The *filter, column aggregation*, and *row transformation* operations share the characteristic that the full range of available operations within these MATA properties was not extensively applied during labeling. During labeling, it was observed that a lot of task frames do not need the use of any of these properties, with *row transformation operations* being particularly infrequent.

Table 4.5: Value counts of the task frames assigned in MATA language to the data corpus.

entity (#)	target (#)	ML-type (#)	filter ops. (#)	col. agg. ops. (#)	row trans. ops. (#)
root (165)	open (21)	regr. (866)	NONE (662)	MEAN (472)	NONE (982)
date (70)	high (21)	class. (120)	= (324)	NONE (452)	$\div$ (3)
id (37)	low (21)	-	$<$ (0)	COUNT (39)	$\times$ (1)
created (29)	close (20)	-	$>$ (0)	SUM (23)	+ (0)
timestamp (17)	volume (19)	-	$\leq$ (0)	FIRST (0)	- (0)
year (24)	score (19)	-	$\geq$ (0)	LAST (0)	$<$ (0)
started (8)	comms (15)	-	$\|\|$ (0)	ANY (0)	$>$ (0)
month (6)	price (11)	-	& (0)	ALL (0)	= (0)
ended (4)	label (11)	-	! (0)	MIN (0)	-
time (4)	value (9)	-	-	MAX (0)	-

In conclusion, the labeled datasets establish the first-ever data corpus for modeling ATD. However, it's important to note that the created data corpus is relatively small, consisting of only 256 labeled datasets. With 256 labeled datasets, only a small variety of possible datasets for supervised ML could be captured.

Furthermore, the labeling of the data has significant bias for two primary reasons. Firstly, the data corpus was labeled by a single data scientist, thus introducing subjectivity to the labels. Secondly, the labeling process comprises qualitative criteria for formulating task frames, including that a *target* variable should be *hard to measure* is *dependent* on other variables and *adds value* by automating its prediction. This qualitative evaluation is purely based on the data expertise of the data scientist. Given that data scientists may lack

domain-specific knowledge, the labeling process may introduce unsuitable task frames for very domain-specific datasets.

Chapter 5

Modeling automatic task discovery

In this chapter, the methods for modeling ATD are introduced. The methods are applied in the experiments, which are presented together with their results and discussion.

5.1 Methods

In the following sections, the reasoning behind the ML approach for ATD is demonstrated. Furthermore, the evaluation approach is outlined, which is used to compare the results of the experiments.

5.1.1 Column type annotation for automatic task discovery

The data labeling process and the Exploratory Data Analysis (EDA) of the labeled data corpus have led to the conclusion that ATD can be formulated as a downstream task of *column type annotation*. Column type annotation is a semantic table interpretation task that aims to label table columns with semantic types derived from Knowledge Graph (KG) [53]. The main challenge lies in identifying appropriate semantic types within a semantic type hierarchy that offers optimal granularity. For example, when considering the *hood* column of the CSV table in Appendix A.1, potential semantic types could include *geographic entity (Q27096213)* or *district (Q149621)*. While both annotations are correct, *district (Q149621)* provides a more fine-grained annotation than *geographic entity (Q27096213)*. In the context of ATD, the goal is not to determine the correct semantic type of a column, but rather to annotate table columns with ATD labels such as *entity, target (classification), target (regression)* and *other*. This formulation of ATD as a column type annotation task is supported by the following reasons:

- **Subjectivity of task frame operations:** In Section 4.2, the analysis of the constructed task frames in the data corpus is presented. Table 4.5 provides a list of usage counts for MATA properties during labeling. It is evident that *filter operations*, *column aggregation operations*, and *row transformation operations* have

been used only limited. The reason for this is that the usage of these operations is highly subjective. Given that a high number of task frames could be formulated for each dataset, determining the business value of each of them cannot be done very objectively. As a result, only the more general task frames were considered. For instance, in the case of the *Housing Prices in San Francisco* dataset, one could formulate the following task frame:

> The task is to build a *regression model*, to predict the ***maximum*** *price* for apartments in San Francisco for *each day*.

In this example, the *column aggregation operation* is MAX. The choice of using *column aggregation operations* is dependent on the specific business use case, which is formulated **subjectively** towards the business' interests. For a different stakeholder, it might be more beneficial to determine the minimum (MIN) price for apartments each day, leading them to opt for a different *column aggregation operation* in this task frame. Similarly, *filter operations* and *row transformation operations* are bound by the same subjective considerations. Therefore, it is very difficult to learn those operations, so they are excluded from being modeled in this work.

- **No observed combinatorial restrictions of entities and targets:** During the labeling of the data corpus, it was observed that there are no combinatorial restrictions between entities and targets. Given the sets of entities $E_i = \{e_1, e_2, \cdots .e_i\}, i = [1, \cdots, u]$, and the *targets* $T_j = \{t_1, t_2, \cdots .t_j\}, j = [1, \cdots, v]$ within a labeled dataset of the data corpus, it was observed that by combining each e_i with each t_j, a total of $u \times v$ meaningful task frames can be constructed. **This enables the formation of ATD as a multiclass classification task**. It is important to note that this observation of no combinatorial restrictions between entities and targets within the data corpus should be interpreted with caution. It does not imply that combinatorial restrictions do not exist in general, but rather that this assumption holds for the constructed data corpus and is used to simplify the modeling of ATD.

- ***ML-type* is only dependent on *target*:** The ML-type of a supervised ML model can be either regression or classification. Inferring the ML-type for a task frame depends solely on the characteristics of the target variable. If the target variable's values are continuous, the ML-type is regression; if the values are discrete, the ML-type is classification. In summary, when the target label is split into *target (classification)* and *target (regression)*, the ML-type can be directly predicted within the context of a multiclass classification task.

Based on the reasoning above, the MATA task frame operations are excluded due to subjectivity. The absence of restrictions on combinatorics allows for the simultaneous

labeling of table columns. Furthermore, the ML-type can be directly inferred from the target column. These prerequisites make column type annotation ML models applicable for ATD. Consequently, a column can be labeled to be whether of type *entity, target (classification), target (regression)*, or *feature*.

5.1.2 Automatic task discovery machine learning pipeline

The ML pipeline developed for ATD consists of four stages: *Input, Information Concatenation, Column Type Annotation Framework*, and *Task Frame Generation*, as presented in Figure 5.1.

The *Input* of the pipeline is a dataset in CSV format, comprising the *context, column names*, and *tabular data*. This data is then forwarded to the *Information Concatenation* stage, where the *context* and *column names* are joined with the *tabular data*. For the tabular data, each column is randomly sampled with $n = 10$ cell values, unless otherwise stated in the experiments. As described in Section 5.1.1, the ML methodology used for ATD is *column type annotation*. Hence, each record in the *Information Concatenation* stage must represent a single table column.

Subsequently, the concatenated data serves as input for the *Column Type Annotation Framework*. In this stage, frameworks such as TURL [24] and Sato [102], described in Section 2.4, are applied. Firstly, the output from the previous stage is vectorized using framework-specific embedding or feature extraction methods. Secondly, the vectorized data is fed into a ML model that predicts, for each record (column), the label *(entity, target (classification), target (regression), other)* with the highest probability.

In the final stage, *Task Frame Generation*, the combinatorial logic introduced in Section 5.1.1 is applied to form supervised ML task frames. If, for instance, u columns in a dataset are labeled as *entities* and v as *targets*, each *entity* is paired with each *target* to create a task frame, resulting in $u \times v$ task frames per dataset. Finally a tuple of *entity* and *target* can be translated into a natural language problem frame, as depicted in Figure 5.1. This translation step is not implemented in this work.

5.1.3 Evaluation

Cross-validation

To train and test the models in the experiments, a K-fold cross-validation with $K = 5$ is performed. The train-test split is applied to the number of datasets in such a way that the columns of a dataset are exclusively kept within the same fold as the dataset it originates from. While the splits are shuffled based on datasets, the index of columns remains unchanged.

The data corpus is initially divided into a training set and a test set with a ratio of 85% (218 datasets, 2085 columns) for training and 15% (38 datasets, 334 columns) for

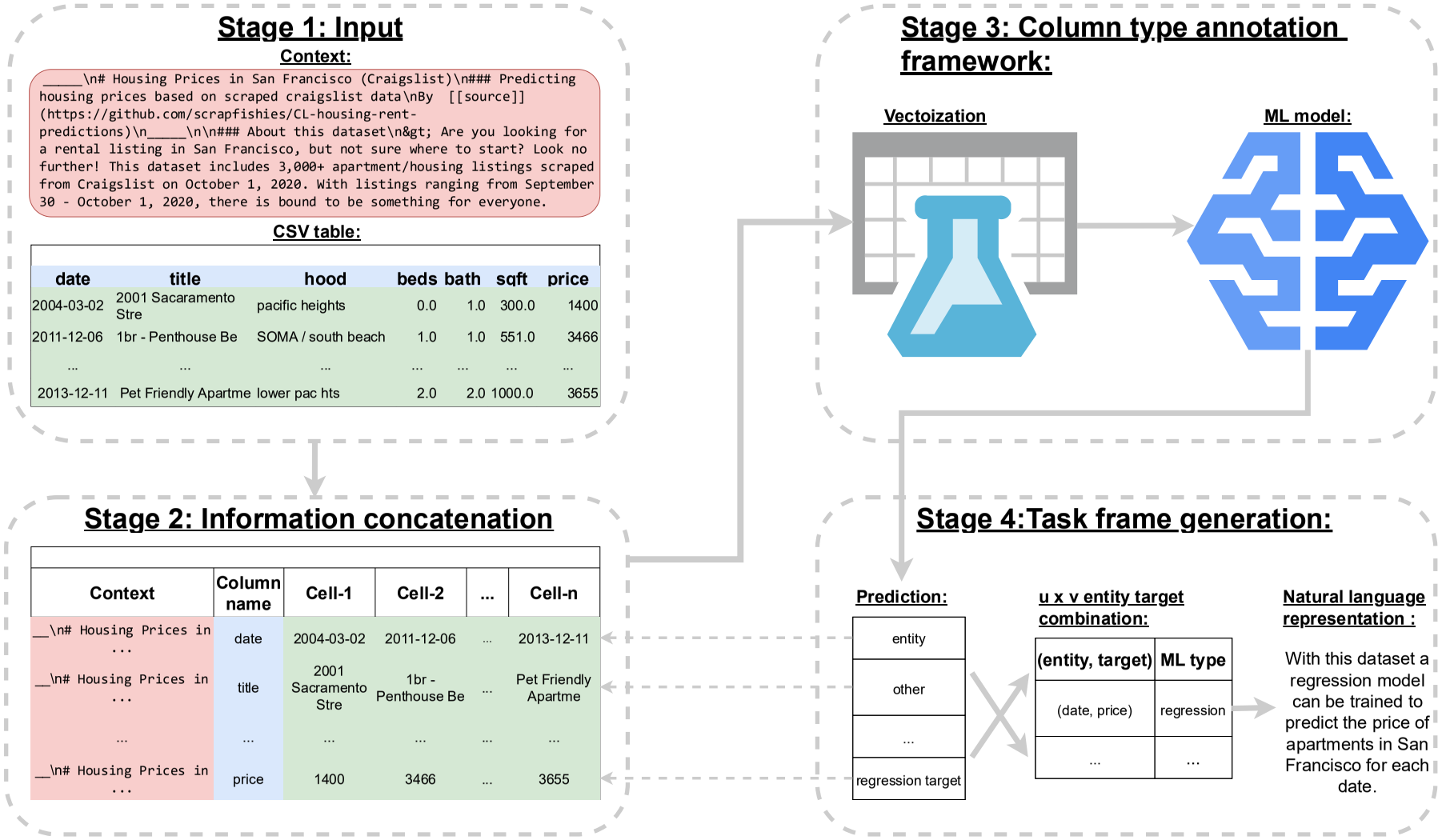

Figure 5.1: The ATD ML pipeline begins with a CSV table, its column names, and a natural language context description as input. In the first step, the data is concatenated so that each record represents a table column and contains the *context*, the *column name*, and the table values. This concatenated data is then passed to the *Column Type Annotation Framework*, which vectorizes the information and serves it as input for the ML model. The ML model's task is to predict the most probable label *(entity, target (regression), target (classification), other)* for each record (column). By assigning one of these labels to each column, it becomes possible to generate $u \times v$ task frames by combining u entities with v targets. Finally, for each combination of predicted *entities* and *targets*, a natural language task frame can be constructed.

testing. Subsequently, the training set is further divided for the 5-fold cross-validation into 5 folds. In each iteration, 4 folds are used for training, and 1 fold is used for validation. This process is repeated 5 times, with a different combination of folds for training and validation, and the mean score + STD of these iterations is reported.

The test set is held out and applied only after all training parameters are determined in Section 5.2.4.

Metrics

Comparing the results of the experiments requires standardizing the reported metrics. In each experiment, models are trained using a 5-fold cross-validation approach as described in Section 5.1.3. The metrics reported include the *macro-averaged F1-score* and its STD calculated over all folds. Additionally, for some experiments, *Precision* and *Recall* are presented. It's important to note that when calculating these metrics, one record represents one column and no aggregation per dataset is performed.

The formulas for calculation are defined as follows:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{Macro F1} = \frac{1}{C} \sum_{i=1}^{C} \text{F1}_i$$

$$\text{STD F1} = \sqrt{\frac{1}{K} \sum_{i=1}^{K} (\hat{\text{Macro}}\ \text{F1}_i - \text{Macro F1})^2}$$

In these equations, *True Positives* represent the number of correctly classified positive records, *False Positives* represents the number of negative records incorrectly classified as positive, *False Negatives* represent the number of positive instances incorrectly classified as negative, and C is the number of classes. *Precision* measures the accuracy of positive predictions, *Recall* measures the coverage of positive records, and the F1 score is the harmonic mean of precision and recall, providing a balanced measure of both metrics. The F1 score is calculated for each class separately and is then averaged over all classes to obtain the *Macro F1* score. *STD F1* represents the standard deviation of the F1 scores across all folds.

5.2 Experiments

In this section, the experimental setups used to model ATD are presented along with the outcomes and a discussion of the results.

5.2.1 E1: Comparison of TURL & Sato

In this experiment, the objective is to compare the *column type annotation frameworks* TURL [24] and Sato [102], introduced in Section 2.4. TURL and Sato differ primarily in their framework architecture. TURL utilizes the BERT-Tokenizer for input data vectorization and employs a BERT-based transformer model for machine learning ML. In contrast, Sato is an approach with an engineered feature set that utilizes four sub-networks to process distinct feature groups, with one main network to produce the outputs.

Secondly, TURL and Sato consider different parts of the input data by default. TURL incorporates all components of a dataset, including the *context, column names,* and *tabular data.* Sato exclusively focuses only on the *tabular data.* To conduct a meaningful comparison of both frameworks, it is necessary to modify the *Information concatenation* mechanisms in both frameworks to enable the activation and deactivation of different input data partitions.

TURL offers a masking option to conceal specific inputs of a dataset, requiring minimal adaptation for its application in this experiment. Conversely, as Sato originally processes only tabular data, the framework had to be enhanced to handle *context* and *column names* as well. By reusing the *paragraph vector* feature extractor from Sato, a *paragraph vector* is extracted from the context. Additionally, the *word embedding* feature extractor is employed to produce *word embeddings* for the column names. In alignment with TURL's approach, a masking mechanism was devised to utilize various data partitions during training.

With these modifications in place, the performance of both TURL and Sato is evaluated by training the frameworks with masking and combining different partitions of the datasets. The various dataset partitions that are analyzed in this experiment are listed below:

- Tabular data
- Colum names
- Context
- Tabular data + column names
- Tabular data + context
- Column names + context

- Tabular data + column names + context

Both frameworks are trained using their default configurations, incorporating the modifications as previously described.

The results of E1 are presented in Table 5.1. The F1 scores indicate that, in most experiments, TURL outperforms Sato. The exception is the experiment of the data partitioning with only tabular data, in which Sato performs slightly better. Generally, using more parts of a dataset as input leads to better model performance. The highest performance is achieved with the *Tabular data + Column names + Context* data partition for the TURL framework, while the lowest performance is observed when using only the *context* part. Furthermore, adding the *context* to other dataset parts only marginally improves performance.

When ranking the F1-scores of individual dataset parts, it becomes evident that the most crucial part of a dataset for ATD is the *tabular data* (F1 TURL = 0.381, F1 STD = 0.016), followed by *column names* (F1 TURL = 0.332, F1 STD = 0.019), and finally, *context* (F1 TURL = 0.024, F1 STD = 0.021).

Table 5.1: Comparison of Sato and Turl on their performance, by masking different parts of input data

	Sato		TURL	
	Macro F1	STD F1	Macro F1	STD F1
Tabular data	**0.396**	0.014	0.381	0.016
Column names	0.241	0.022	**0.332**	0.019
Context	NaN	NaN	**0.024**	0.021
Tabular data + column names	0.467	0.020	**0.588**	0.018
Tabular data + context	0.392	0.019	**0.396**	0.014
Column names + context	0.240	0.015	**0.331**	0.012
Tabular data + column names + context	0.483	0.018	**0.602**	0.017

Previous research [25, 55, 15] has demonstrated that transformer-based deep learning architectures outperform traditional feature engineering and embedding methods. Therefore, the expectation was that TURL would outperform Sato in most experiments. Particularly, the data partitions that included multiple dataset parts achieved significantly higher F1-scores compared to Sato. This result can be attributed to the fact that the default Sato framework was originally designed to process only tabular data but was modified to additionally handle *column names* and *context* using word- and paragraph embeddings. Given

that transformers have been proven to excel in various NLP tasks, this outcome aligns with expectations.

Notably, Sato exhibits slightly better performance when both frameworks are trained with only *tabular data*, highlighting that Sato is a high-performing column annotation framework for *tabular data* only. However, since most CSV files include *column names* and *tabular data*, TURL is the preferred choice for ATD. Consequently, only TURL will be used in the subsequent experiments, as it can effectively process all parts of a dataset and consistently achieves strong performance in this task.

5.2.2 E2: Sampling of n rows

Machine learning datasets can become extremely large, sometimes reaching storage sizes of multiple Terrabyte (TB). To facilitate the processing of such large datasets, the minimum sampling size (n) required to maintain representative power for table column values in ATD is investigated. In this experimental setup, a simple random sample of n cells per column is extracted. n is selected from a logarithmic scale with a base of 2, spanning the range from 2^0 to 2^{20}. Each column is sampled $m = 100$ times, with m held constant. TURL is trained using its default configurations, and all dataset parts are used as input.

The results of E2 are presented in a scatter plot shown in Figure 5.2. In this plot, the F1-score for different values of n in the range of 2^0 to 2^{20} reveals that the F1-score reaches a threshold of F1=0.670, F1 STD=0.012 around $n = 2^4$. Beyond this point, there is no significant increase in the F1-score. Therefore, it is argued that a sampling size of 8 is sufficient to represent the information provided by the values of a table column for the ATD. In the next experiments, models are trained using records in which 8 values per column are sampled.

The outcomes of this experiment align with expectations, as data scientists typically inspect only a small sample of cell values per column when formulating ML task frames. Therefore, it is not surprising that a small sampling size proves to be sufficient for ATD. These results are consistent with those of Suhara et al. [84], who also concluded that a sampling size of 8 captures the column's variance effectively. Moreover, another study [90] achieved a similar high performance with a sampling size of 10 and observed that performance plateaued at a sampling size of 20, showing no further improvement. In conclusion, the results of E2 are supported by existing literature and align with prior expectations.

5.2.3 E3: Sampling $m \times n$ rows

In the previous experiment, it was analyzed how many rows need to be sampled from a single column to capture its variance. In the current experiment, it is investigated how

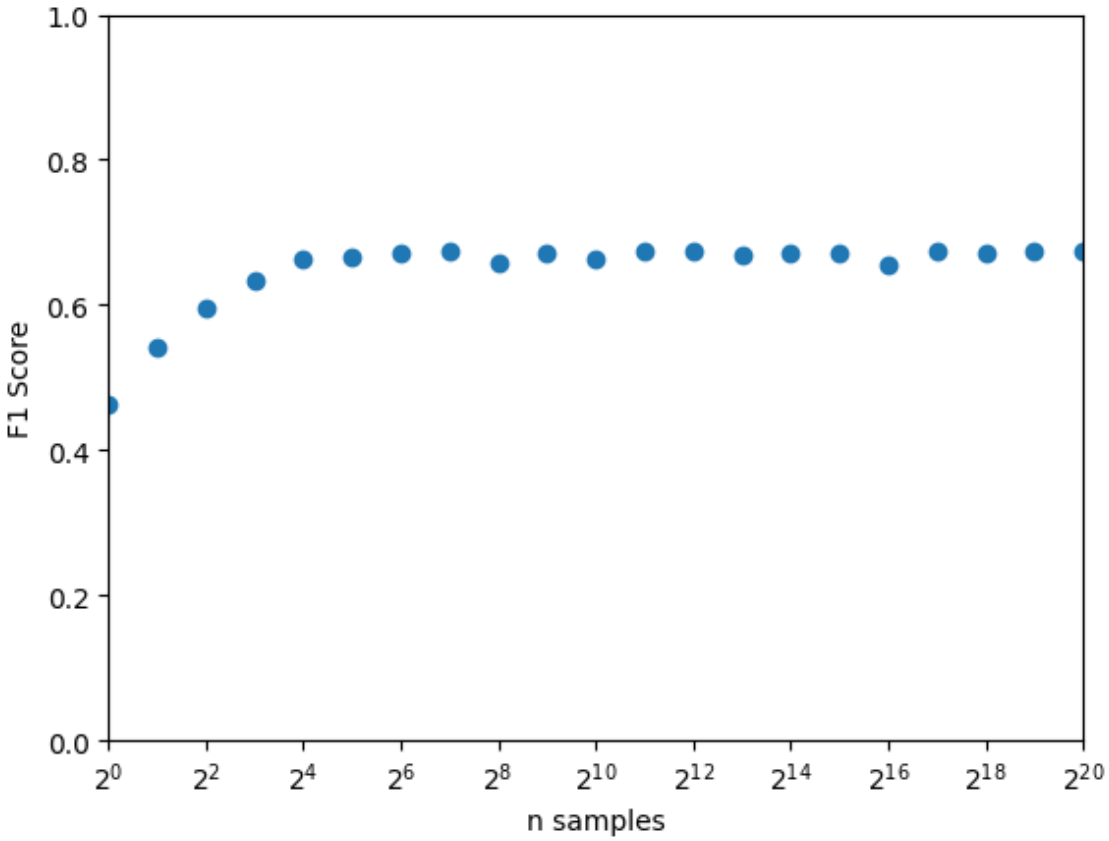

Figure 5.2: Scatter plot displaying the change of the F1-score of TURL while changing the number of sampled rows n of a column.

many times a column can be sampled to increase the variance in the training data and produce a better generalizing ML model. The sampling size n is fixed at 8, while the number of times a column is sampled, denoted as m, varies across the range of 2^0 to 2^{20}. The sampling is performed without replacement, ensuring that each column value occurs only once in the training data. When the product of the number of sampled values $m \times n$ exceeds the number of values in a column, the column is sampled as many times as possible.

Figure 5.3 illustrates the change in the F1-score achieved by TURL when a column is sampled m times with a constant n rows. The F1-score increases steadily until it reaches its peak value of F1=0.804, F1 STD=0.008 at 2^{15}. This indicates that sampling a column from a dataset multiple times significantly improves the performance of a column annotation model. In the subsequent experiments, the results of this experiment are applied, ensuring that each column is sampled $m = 2^{15}$ times with $n = 8$ rows unless fewer rows are available.

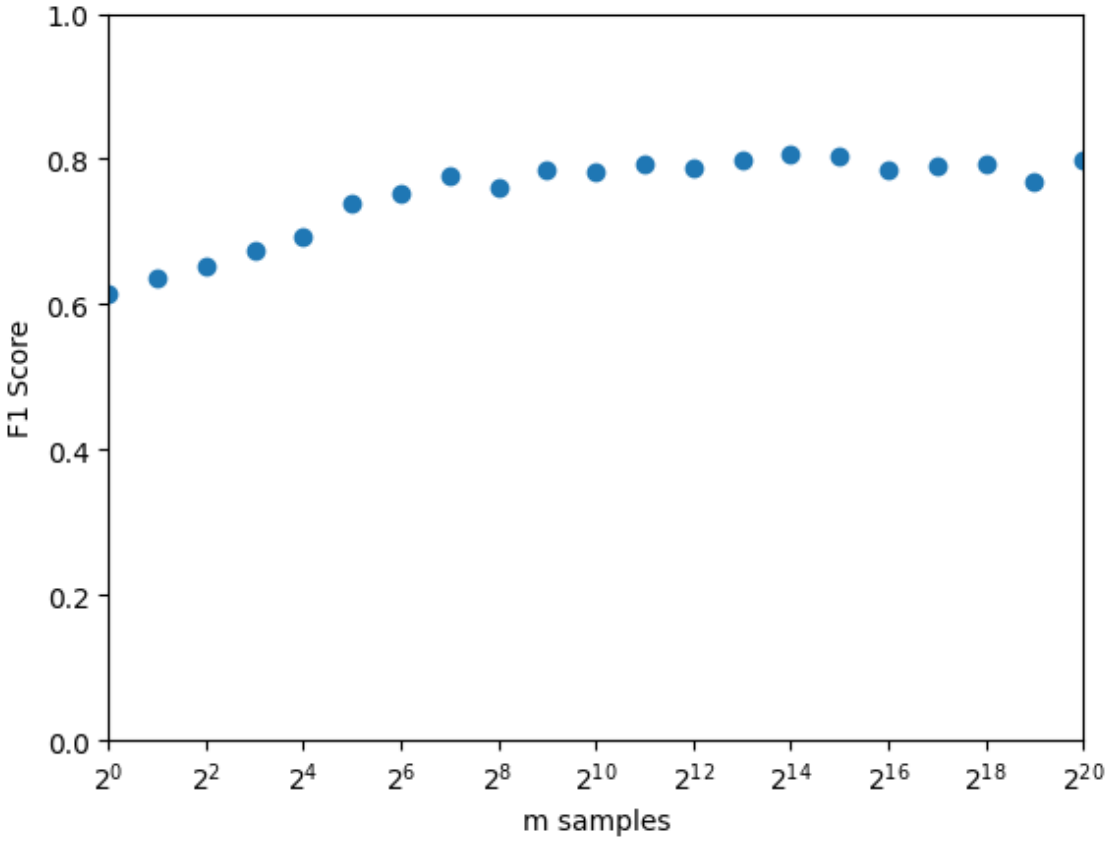

Figure 5.3: Scatter plot displaying the change of the F1-score of TURL while sampling m times n rows of a column.

Data augmentation through the resampling of column values for ATD significantly enhances model performance and generalizability. It's worth noting that literature focusing on column-type annotation typically has access to large amounts of data, making

data augmentation less common and restricting direct comparisons to similar research. In contrast, data augmentation techniques are well-established in other domains, such as NLP [94] and computer vision [80], where they have consistently led to significant performance improvements.

Additionally, it's important to consider that the numerical reasoning capabilities of transformers are known to have limitations [89]. Given that Table 4.3 indicates the presence of numerous numerical data types in the data corpus, resampling columns in the data corpus m times can contribute to enhancing the model's numerical stability.

Despite the positive impact on overall model performance, this experiment does not investigate whether over- or undersampling the classes of ATD would further enhance performance.

5.2.4 E4: Hyperparameter tuning

Deep learning models have numerous hyperparameters that can be fine-tuned for optimal performance. In this study, the transformer-based column type annotation framework is applied. To maximize TURL's effectiveness in learning the downstream task of ATD, it's crucial to tune the learning parameters of this model.

The hyperparameter search space, as outlined in Table 5.2, includes parameters such as *Batch size, Learning rate, Learning rate schedule, Epochs* and *Weight decay*. The chosen value ranges for these hyperparameters align with the recommendations provided by the authors of TURL [24]. To systematically explore this hyperparameter space, the distributed hyperparameter tuning framework, Ray Tune [51] is utilized to conduct a random grid search comprising 100 trials. Subsequently, the best-performing model is selected and evaluated on the test set.

Table 5.2: Hyperparameter search space to tune the learning parameters of TURL

Parameter	Value Space
Batch size	{5, 10, 20 }
Learning rate	$\{1^{-6}, 5^{-6}, 1^{-5}, 5^{-5}, 1^{-4}, 5^{-4}\}$
Learning rate schedule	{constant, linear decay, linear decay + warmup}
Epochs	{5, 10, 20}
Weight decay	$\{1^{-2}, 1^{-1}\}$

Through the optimization of TURL's learning hyperparameters, the performance on ATD was further enhanced, resulting in F1=0.863, F1 STD = 0.009 on the validation set and F1=**0.847** on the test set. The hyperparameter configuration that led to this result is provided in Table 5.3.

Table 5.3 reveals that the best-performing hyperparameters include a small batch size (5) and a high learning rate (1^{-4}), combined with a relatively small number of

Table 5.3: Hyperparameter configuration resulting in the highest performance.

Parameter	Value
Batch size	5
Learning rate	1^{-4}
Learning rate schedule	constant
Epochs	5
Weight decay	1^{-1}

epochs (5). This finding aligns with the observations made by Masters and Luschi [62], who reported that small batch sizes and high learning rates often lead to faster convergence and improved generalization, which is consistent with the found parameter choice.

Interestingly, the chosen learning rate schedule in this configuration is *constant*, which is unexpected. In the literature, linearly decreasing learning rates are often favored as they tend to contribute to better generalization [76, 99]. Additionally, *warmup* is known to stabilize model training and accelerate convergence [54, 55]. However, in this experiment, the *warmup* strategy did not prove to be beneficial.

5.2.5 E5: Error analysis

To analyze the errors in the model's predictions, this experiment quantitatively captures the number of correctly and incorrectly classified examples by the tuned model on the test set. As a result, class-wise *Precision, Recall* and *F1-score* are calculated. Additionally, a confusion matrix is plotted to visualize whether specific classes are more frequently misclassified than others.

Table 5.4 shows the performance metrics, including *Precision, Recall,* and *F1-score*, for each class. It is evident that the *entity* class achieves the highest F1-score, while the *target (class.)* class has the lowest F1-score.

To gain deeper insights into classification errors, the confusion matrix in Figure 5.4 is examined. The *entity* class, which achieves the highest F1-score, is rarely confused with other classes. Conversely, the *other* class, which has the largest number of samples, exhibits the highest number of absolute misclassified records, including 9 as *entity*, 7 as *target (class.)*, and 6 as *target (regr.)*. The *target (regr.)* and *target (class.)* classes are exclusively misclassified as *other*.

Empirical analysis has revealed that columns labeled as *entity* often exhibit low diversity. A high proportion of columns marked as *entity* contain *datetime* data or follow an incrementing index pattern. On the other hand, the lower F1-score observed for the *target (class.)* class can be attributed to the high class imbalance.

To potentially enhance the performance of the *target (class.)* class, one approach

Table 5.4: Class specific precision, recall and F1-score of the test set.

Class	Precision	Recall	F1-score	Support
entity	0.855	0.946	0.898	56
other	0.905	0.880	0.893	184
target (class.)	0.650	0.812	0.722	16
target (regr.)	0.905	0.848	0.876	79
macro avg	**0.829**	**0.872**	**0.847**	**335**

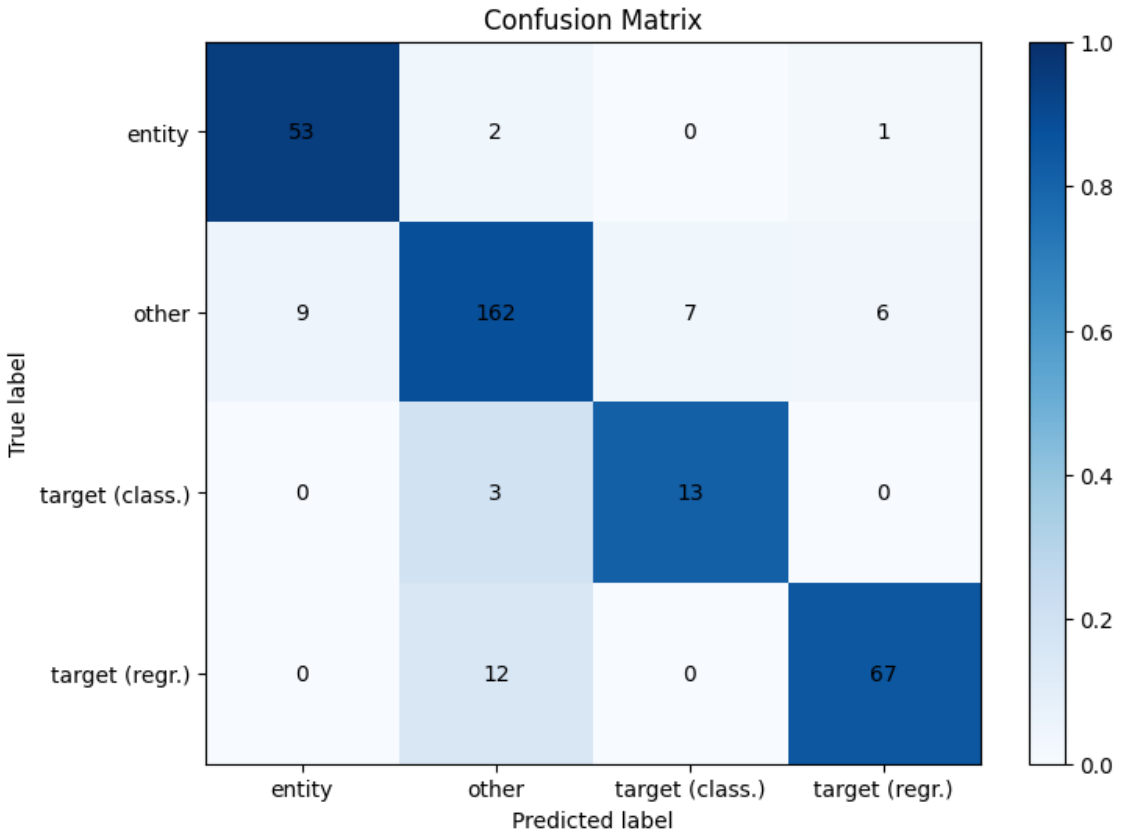

Figure 5.4: Confusion matrix, showing the absolute numbers of correctly and incorrectly classified examples of the test set.

could involve resampling columns from this class to generate more training examples. However, implementing such a resampling strategy requires careful consideration, as column annotation frameworks take into account the *column context* within a dataset. Altering the combination of columns in a dataset would require a more sophisticated column resampling approach.

5.2.6 E6: Qualitative task frame analysis

In this experiment, the optimized column-type annotation framework from Section 5.2.4 is applied to infer task frames for a given dataset, which are afterward analyzed qualitatively. The objective is to determine whether the predicted column labels are accurate and whether the generated task frames hold meaningful insights.

For this experiment, two datasets are selected: one for which the model performs poorly (*Satellite Data on Australia Fires*[1]) and another one for which the model performs well (*NVIDIA stock price*[2]).

Inferring labels for each column of the *Satellite Data on Australia Fires* dataset yielded suboptimal results. The annotation of this dataset is presented in Table 5.5. Herein, two columns were misclassified. The column *brightness* was misclassified as *target (regression)* instead of its correct label, which should be *other*. Utilizing the *root entity satellite measurement* and the target *brightness* would result in the following natural language task frame:

> *The task is to build a regression model to predict the brightness of each satellite measurement.*

Since *brightness* is a main feature in satellite measurements, constructing a task frame for its prediction would not be meaningful. Similarly, the label for column *daynight*was incorrectly predicted as *target (classification)* instead of *other*. Predicting whether it is day or night is also not meaningful, as this is a fundamental characteristic of satellite measurements rather than a target variable. The reason for this is that the training data does not contain any datasets that are very similar to the *Satellite Data on Australia Fires* dataset. The model is therefore not able to generalize well on this dataset.
Inspecting the predictions for the *NVIDIA stock price* dataset in Table 5.6 shows, that the model could achieve this task without any errors. The *Date* column was correctly identified as an entity, while the columns *Open, High, Low, Close* and *Adj. Close* were identified as regression targets. A meaningful example task frame could be built like the following:

[1] https://www.kaggle.com/datasets/gabrielbgutierrez/satellite-data-on-australia-fires

[2] https://www.kaggle.com/datasets/kannan1314/nvidia-stock-price-all-time

The task is to build a regression model to predict the High value of the NVIDIA stock price for each day.

Since a lot of datasets in the data corpus are stock data, the model is trained on similar data and it is easy for the model to predict the correct labels.

Table 5.5: ATD predictions for the *Satellite Data on Australia Fires* dataset, with incorrectly predicted labels.

Context:	There isn't a lot of datasets that provide historical data about fires in Australia. So I compiled data from NASA's MODIS satellite MODIS/Aqua + Terra Thermal Anomalies/Fire locations that provides data from hotspots/fires on Australia since 2000.					
Column names:	latitude	brightness	acq_date	satellite	daynight	type
Tabular data:	-25.138	331.1	2002-10-09	Terra	N	0
	-14.975	340.2	2002-11-05	Terra	D	0
	-34.33	308.2	2004-06-08	Aqua	D	0
	...	...	...	...	...	...
Predictions:	other	target (regr.)	entity	other	target (class.)	target (class.)
True labels:	other	other	entity	other	other	target (class.)

Table 5.6: ATD predictions for the *Nvidia stock price* dataset, with correctçly predicted labels.

Context:	NVIDIA Corp. engages in the design and manufacture of computer graphics processors, chipsets, and related multimedia software. It operates through the following segments: Graphics Processing Unit (GPU), Tegra Processor, and All Other. The GPU ...					
Column names:	Date	Open	High	Low	Close	Adj. Close
Tabular data:	2000-05-05	1.827	1.964	1.802	1.919	1.763
	2004-02-03	1.879	1.912	1.853	1.877	1.725
	2004-07-07	1.53	1.562	1.511	1.518	1.395
	...	...	...	...	...	...
Predictions:	entity	target (regr.)	target (regr.)	target (regr.)	target (regr.)	target (regr.)
True labels:	entity	target (regr.)	target (regr.)	target (regr.)	target (regr.)	target (regr.)

Chapter 6

Conclusions

In this chapter, the primary findings of this work are summarized. Additionally, the limitations of this study and avenues for future research are presented and discussed.

6.1 Summary of main contributions

In this work, three main scientific contributions were established. Firstly, the TFRL MATA was created, offering improved generalizability compared to Trane, as it is not limited to time-series data only. The versatility of MATA was directly demonstrated through the labeling of 256 datasets with 986 task frames, forming a data corpus that supports data-driven modeling of ATD. This data corpus is the first creation of a collection of datasets that structurally maps supervised ML task frames to datasets and can be used as a benchmark dataset in future research, which is the second scientific contribution of this work.

Thirdly, a novel ML approach for modeling ATD was developed using the column type annotation framework TURL. Through the fine-tuning of TURL, a high-performing model was created, capable of predicting supervised ML task frames.

In summary, this work illustrates the feasibility of automating supervised ML task discovery through a data-driven approach and extends the automation of the ML lifecycle.

6.2 Limitations

Although the results are very promising, it is essential to transparently present the limitations of this study. The main limitations center around the simplifications made to ensure feasibility. In the design of MATA, it was assumed that the nesting of the properties and operations of MATA is not supported. This limitation can pose challenges when working with datasets having complex relationships between columns, so that MATA's current syntax may be insufficient.

Another limitation arises from the relatively small and biased data corpus. Given that

data can be sourced from various domains, a corpus comprising only 256 datasets cannot be considered representative. Consequently, the designed model's generalizability is constrained, particularly when applied to highly specific datasets. Furthermore, bias is introduced during the labeling process due to the qualitative nature of the rules for formulating task frames. Labeling by a single data scientist increases this bias additionally, and it remains untested how significant this bias might be when the model is applied to different data sources.

Lastly, the introduction of the unrestricted combinatorial logic for *entities* and *targets* was necessary to apply column type annotation frameworks. While through this work it was not observed that this logic leads to meaningless task frames, it is speculated that for more complex datasets, the ability to formulate meaningful task frames may be limited by this simplification.

6.3 Future work

This work establishes the first approach to fully automate the ML stage of task framing for supervised learning. Consequently, it opens several potentials for future research. The first step involves expanding the current data corpus by incorporating data from multiple sources and engaging multiple labelers for a comprehensive review, to reduce the current bias and enhance the diversity of the corpus. This expansion will contribute to the development of a more robust model with improved generalization.

As a next step, it would be interesting to conduct integration tests within existing AutoML tools like AutoWeka [85] or Auto-Sklearn [75]. Such tests would determine the feasibility of generating meaningful ML models completely automatically. Through that, it can be tested if this extension of AutoML would help non-data scientists generate their own ML models.

Supplementary, there is potential to enhance this work by introducing automatic task discovery for unsupervised ML. Unsupervised ML is very diverse, therefore it is difficult to discover useful unsupervised learning tasks for datasets. With this extension, this process would be accelerated.

Appendix A

A.1 San Francisco house pricing metadata


```
{
  "id": "thedevastator/scraping-apartments-off-of-
     craigslist-in-san-fra",
  "datasetId": 2677154,
  "ownerUser": "thedevastator",
  "usabilityRating": 1.0,
  "totalViews": 1784,
  "totalVotes": 9,
  "totalDownloads": 284,
  "title": "Housing Prices in San Francisco (Craigslist)",
  "subtitle": "Predicting housing prices based on scraped
     craigslist data",
  "description": "_____\n# Housing Prices in San Francisco
     (Craigslist)\n### Predicting housing prices based on
     scraped craigslist data\nBy [[source]](https://github
     .com/scrapfishies/CL-housing-rent-predictions)\n_____\
     n\n### About this dataset\n&gt; Are you looking for a
     rental listing in San Francisco, but not sure where to
      start? Look no further! This dataset includes 3,000+
     apartment/housing listings scraped from Craigslist on
     October 1, 2020. With listings ranging from September
     30 - October 1, 2020, there is bound to be something
     for everyone.",
  "licenses": [
    {
      "nameNullable": "CC0-1.0",
      "name": "CC0-1.0",
      "hasName": true
    }
  ]
}
```

A.2 San Francisco house pricing tabular data

Table A.1: Sample of the San Francisco house pricing CSV file.

date	title	hood	beds	bath	sqft	price
2004-03-02	2001 Sacramento Stre	pacific heights	0.0	1.0	300.0	1400
2011-12-06	1br - Penthouse 1 Be	SOMA / south beach	1.0	1.0	551.0	3466
2016-02-11	Pet friendly Studio	downtown / civic / van ness	2.0	2.0	1479.0	4375
2015-11-03	Perfect 2bd on Chest	russian hill	1.0	1.0	784.0	2995
2019-04-01	2 months free! Updat	mission district	0.0	1.0	400.0	1600
2018-05-29	BIG VIEW! Penthouse	SOMA / south beach	4.0	2.0	1800.0	3995
2003-07-14	Move now, pay rent l	mission district	2.0	1.5	840.0	2040
2018-03-21	Fully Remodeled Jr.	pacific heights	2.0	2.0	916.0	3412
2018-11-15	XL 1 BR 1/2 Block to	marina / cow hollow	1.0	1.0	916.0	3200
2013-12-11	Pet Friendly Apartme	lower pac hts	2.0	2.0	1000.0	3655

A.3 San Francisco house pricing problem frame

```
{
    "entity": "date",
    "target": "AGG",
    "filter_by": {
        "hood":"russian hill"
    },
    "filter_operation": {
        "hood":"="
    },
    "column_aggregate_by": "date",
    "column_aggregate_operation": "mean",
    "row_aggregate_by": [
        "price",
        "sqft"
    ],
    "row_operation": "/",
    "ml_type": "regression"
}
```

A.4 Column aggregation operations in MATA

Table A.2: Listing of all column aggregation operations of MATA. Adapted from [78]

Aggregation operation	Input type	Output type	Explanation
IDENTITY	any	any	Does nothing, returns the input as is.
LAST	any	same	Returns the last value.
FIRST	any	same	Returns the first value.
LAST_MINUS_FIRST	numeric	numeric	Returns the result of substracting the last minus the first value.
ANY	Boolean	Boolean	Given an Boolean column as input, returns True if any of the values is True.
ALL	Boolean	Boolean	Given an Boolean column as input, returns True if any of the values is True.
SUM	Numeric	Numeric	Returns the sum of the values that serve as input
COUNT	Any	Integer	Returns the count of values given as input
MEAN	Numeric	Numeric	Returns the mean of values given as input
MIN	Numeric	Numeric	Returns the minimum of values given as input
MAX	Numeric	Numeric	Returns the maximum of values given as input

Bibliography

[1] Charu Aggarwal, Djallel Bouneffouf, Horst Samulowitz, Beat Buesser, Thanh Hoang, Udayan Khurana, Sijia Liu, Tejaswini Pedapati, Parikshit Ram, Ambrish Rawat, Martin Wistuba, and Alexander Gray. How can AI Automate End-to-End Data Science?, October 2019. arXiv:1910.14436 [cs].

[2] Charu C. Aggarwal. *Neural networks and deep learning: a textbook*. Springer, Cham, Switzerland, 2018.

[3] Ahmad Alsharef, Karan Aggarwal, Sonia, Manoj Kumar, and Ashutosh Mishra. Review of ML and AutoML Solutions to Forecast Time-Series Data. *Archives of Computational Methods in Engineering*, 29(7):5297–5311, November 2022. Number: 7.

[4] [Figure reference] An SangGyu. *Introduction to how an Multilayer Perceptron works but without complicated math*. https://medium.com/codex/introduction-to-how-an-multilayer-perceptron-works-but-without-complicated-math-a423979897ac, Accessed: 2023-08-30 14:53:51, October 2021.

[5] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data Augmentation Generative Adversarial Networks, March 2018. arXiv:1711.04340 [cs, stat].

[6] [Figure reference] Antti Honkela. *Multilayer perceptrons*. https://users.ics.aalto.fi/ahonkela/dippa/node41.html, Accessed: 2023-08-30 14:43:59, May 2001.

[7] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing Neural Network Architectures using Reinforcement Learning. In *International Conference on Learning Representations*, 2017.

[8] Arian Bakhtiarnia, Qi Zhang, and Alexandros Iosifidis. Multi-Exit Vision Transformer for Dynamic Inference. 2021. Publisher: arXiv Version Number: 3.

[9] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for Hyper-Parameter Optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc. event-place: Granada, Spain.

[10] James Bergstra and Yoshua Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012. Type: misc.

[11] James Bergstra, Daniel Yamins, and David D. Cox. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *International Conference on Machine Learning*, 2013.

[12] Francine Berman, Rob Rutenbar, Brent Hailpern, Henrik Christensen, Susan Davidson, Deborah Estrin, Michael Franklin, Margaret Martonosi, Padma Raghavan, Victoria Stodden, and Alexander S. Szalay. Realizing the potential of data science. *Communications of the ACM*, 61(4):67–72, March 2018.

[13] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3(null):993–1022, March 2003.

[14] G. Bradski. *The OpenCV Library*. Dr. Dobb's Journal of Software Tools, http://docs.opencv.org/, Accessed: 2023-08-30 11:33:21, 2000.

[15] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. 2020. Publisher: arXiv Version Number: 4.

[16] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002. arXiv:1106.1813 [cs].

[17] Xinlei Chen, Abhinav Shrivastava, and Abhinav Gupta. NEIL: Extracting Visual Knowledge from Web Data. In *2013 IEEE International Conference on Computer Vision*, pages 1409–1416, 2013.

[18] Xu Chu, Ihab F. Ilyas, Sanjay Krishnan, and Jiannan Wang. Data Cleaning: Overview and Emerging Challenges. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 2201–2206, New York, NY, USA, 2016. Association for Computing Machinery. event-place: San Francisco, California, USA.

[19] Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. KATARA: A Data Cleaning System Powered by Knowledge Bases and

Crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1247–1261, Melbourne Victoria Australia, May 2015. ACM.

[20] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy, 2019. Association for Computational Linguistics.

[21] M Dash and H Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(1-4):131–156, 1997.

[22] Sumeyra Demir, Krystof Mincev, Koen Kok, and Nikolaos G. Paterakis. Data augmentation for time series regression: Applying transformations, autoencoders and adversarial networks to electricity price forecasting. *Applied Energy*, 304:117695, December 2021.

[23] Janez Demšar, Tomaž Curk, Aleš Erjavec, Črt Gorup, Tomaž Hočevar, Mitar Milutinovič, Martin Možina, Matija Polajnar, Marko Toplak, Anže Starič, Miha Štajdohar, Lan Umek, Lan Žagar, Jure Žbontar, Marinka Žitnik, and Blaž Zupan. Orange: Data Mining Toolbox in Python. *Journal of Machine Learning Research*, 14:2349–2353, 2013.

[24] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. TURL: Table Understanding through Representation Learning. *ACM SIGMOD Record*, 51(1):33–40, May 2022.

[25] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2018. Publisher: arXiv Version Number: 2.

[26] David Donahue and Anna Rumshisky. Adversarial Text Generation Without Reinforcement Learning, January 2019. arXiv:1810.06640 [cs, stat].

[27] Iddo Drori, Yamuna Krishnamurthy, Remi Rampin, Raoni de Paula Lourenco, Jorge Piazentin Ono, Kyunghyun Cho, Claudio Silva, and Juliana Freire. AlphaD3M: Machine Learning Pipeline Synthesis. 2021. Publisher: arXiv Version Number: 1.

[28] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International*

Conference on Machine Learning, volume 48 of *Proceedings of Machine Learning Research*, pages 1329–1338, New York, New York, USA, June 2016. PMLR.

[29] Andrew Flowers. *Data Scientist: A Hot Job That Pays Well*. https://www.hiringlab.org/2019/01/17/data-scientist-job-outlook/, Accessed: 2022-12-09 17:14:51, January 2019.

[30] João Gama. Functional Trees. *Machine Learning*, 55(3):219–250, June 2004.

[31] Majid Ghasemi-Gol and Pedro Szekely. TabVec: Table Vectors for Classification of Web Tables, February 2018. arXiv:1802.06290 [cs].

[32] Yolanda Gil, James Honaker, Shikhar Gupta, Yibo Ma, Vito D'Orazio, Daniel Garijo, Shruti Gadewar, Qifan Yang, and Neda Jahanshad. Towards human-guided machine learning. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pages 614–624, Marina del Ray California, March 2019. ACM.

[33] Chengyue Gong, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tie-Yan Liu. FRAGE: Frequency-Agnostic Word Representation. 2018. Publisher: arXiv Version Number: 2.

[34] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[35] Google. *Google Cloud AI platform*. https://cloud.google.com/ai-platform/docs/technical-overview, Accessed: 2022-12-12 10:13:29, 2022.

[36] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, November 2009.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, June 2016. IEEE.

[38] Xin He, Kaiyong Zhao, and Xiaowen Chu. AutoML: A Survey of the State-of-the-Art. *Knowledge-Based Systems*, 212:106622, January 2021. arXiv:1908.00709 [cs, stat].

[39] Kevin Hu, Neil Gaikwad, Michiel Bakker, Madelon Hulsebos, Emanuel Zgraggen, César Hidalgo, Tim Kraska, Guoliang Li, Arvind Satyanarayan, and Çağatay Demiralp. VizNet: Towards A Large-Scale Visualization Learning and Benchmarking Repository. 2019. Publisher: arXiv Version Number: 1.

[40] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çağatay Demiralp, and César Hidalgo. Sherlock: A Deep Learning Approach to Semantic Data Type Detection, May 2019. arXiv:1905.10688 [cs, stat].

[41] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. TABBIE: Pretrained Representations of Tabular Data. 2021. Publisher: arXiv Version Number: 1.

[42] Zubayer Islam, Mohamed Abdel-Aty, Qing Cai, and Jinghui Yuan. Crash data augmentation using variational autoencoder. *Accident Analysis & Prevention*, 151:105950, March 2021.

[43] Thorsten Joachims. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. Technical report, Carnegie-mellon univ pittsburgh pa dept of computer science, 1996.

[44] Shubhra Kanti Karmaker ("Santu"), Md. Mahadi Hassan, Micah J. Smith, Lei Xu, Chengxiang Zhai, and Kalyan Veeramachaneni. AutoML to Date and Beyond: Challenges and Opportunities. *ACM Computing Surveys*, 54(8):1–36, November 2022. Number: 8.

[45] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. 2018. Publisher: arXiv Version Number: 3.

[46] Scott Klein. Azure Machine Learning. In *IoT Solutions in Microsoft's Azure IoT Suite*, pages 227–252. Apress, Berkeley, CA, 2017.

[47] Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, Jiannan Wang, and Eugene Wu. ActiveClean: An Interactive Data Cleaning Framework For Modern Machine Learning. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2117–2120, San Francisco California USA, June 2016. ACM.

[48] Sanjay Krishnan and Eugene Wu. AlphaClean: Automatic Generation of Data Cleaning Pipelines, May 2019. arXiv:1904.11827 [cs].

[49] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, May 2017.

[50] Karim Lakhani, Shane Greenstein, and Kerry Herman. AWS and Amazon SageMaker (A): The Commercialization of Machine Learning Services. Harvard Business School Case 622-060, 2022.

[51] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez, and Ion Stoica. Tune: A Research Platform for Distributed Model Selection and Training, July 2018. arXiv:1807.05118 [cs, stat].

[52] Richard Lippmann, William Campbell, and Joseph Campbell. An overview of the DARPA data driven discovery of models (D3M) program. *In Proceedings of the NIPS Workshop on Artificial Intelligence for Data Science.*, 2016.

[53] Jixiong Liu, Yoan Chabot, Raphaël Troncy, Viet-Phi Huynh, Thomas Labbé, and Pierre Monnin. From tabular data to knowledge graphs: A survey of semantic table interpretation tasks and methods. *Journal of Web Semantics*, 76:100761, April 2023.

[54] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the Variance of the Adaptive Learning Rate and Beyond. 2019. Publisher: arXiv Version Number: 4.

[55] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. 2019. Publisher: arXiv Version Number: 1.

[56] Edward Ma. *NLP Augmentation*. https://github.com/makcedward/nlpaug, Accessed: 2022-12-13 12:30:08, 2019.

[57] Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based Hyperparameter Optimization through Reversible Learning. 2015. Publisher: arXiv Version Number: 3.

[58] Mohammad Mahdavi, Felix Neutatz, Larysa Visengeriyeva, and Ziawasch Abedjan. Towards Automated Data Cleaning Workflows. September 2019.

[59] Sasu Makinen, Henrik Skogstrom, Eero Laaksonen, and Tommi Mikkonen. Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help? In *2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN)*, pages 109–112, Madrid, Spain, May 2021. IEEE.

[60] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the international conference on Multimedia - MM '10*, page 1485, Firenze, Italy, 2010. ACM Press.

[61] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015.

[62] Dominic Masters and Carlo Luschi. Revisiting Small Batch Training for Deep Neural Networks. 2018. Publisher: arXiv Version Number: 1.

[63] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, December 1943.

[64] Wes McKinney. Data Structures for Statistical Computing in Python. pages 56–61, Austin, Texas, 2010.

[65] Wes McKinney. *Python for data analysis: data wrangling with pandas, NumPy, and IPython*. O'Reilly Media, Inc, Sebastopol, California, second edition edition, 2018. OCLC: ocn959595088.

[66] Qinxue Meng, Daniel Catchpoole, David Skillicom, and Paul J. Kennedy. Relational autoencoder for feature extraction. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 364–371, 2017.

[67] Ingo Mierswa and Ralph Klinkenberg. *RapidMiner Studio*. https://my.rapidminer.com/nexus/account/index.html, Accessed: 2022-12-12 10:13:29, 2018.

[68] Agnieszka Mikołajczyk and Michał Grochowski. Data augmentation for improving deep learning in image classification problem. In *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, pages 117–122, 2018.

[69] Agnieszka Mikołajczyk and Michał Grochowski. Style transfer-based image synthesis as an efficient regularization technique in deep learning, May 2019. arXiv:1905.10974 [cs, eess].

[70] Hiroshi Motoda and Huan Liu. Feature Selection Extraction and Construction. New York, NY, 2002. Springer.

[71] U.S. Bureau of Labor Statistics. *OEWS Data*. https://www.bls.gov/oes/tables.htm#nat, Accessed: 2022-12-09 17:17:22, May 2021.

[72] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 485–492, Denver Colorado USA, July 2016. ACM.

[73] Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. Data Synthesis based on Generative Adversarial Networks. 2018. Publisher: arXiv Version Number: 5.

[74] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. 2019. Publisher: arXiv Version Number: 1.

[75] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python, June 2018. arXiv:1201.0490 [cs].

[76] Biserka Petrovska, Tatjana Atanasova-Pacemska, Roberto Corizzo, Paolo Mignone, Petre Lameski, and Eftim Zdravevski. Aerial Scene Classification through Fine-Tuning with Adaptive Learning Rates and Label Smoothing. *Applied Sciences*, 10(17):5792, August 2020.

[77] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient Neural Architecture Search via Parameters Sharing. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104. PMLR, July 2018.

[78] Benjamin Schreck and Kalyan Veeramachaneni. What Would a Data Scientist Ask? Automatically Formulating and Solving Predictive Problems. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 440–451, Montreal, QC, Canada, October 2016. IEEE.

[79] Sam Scott and Stan Matwin. Feature engineering for text classification. In *ICML*, volume 99, pages 379–388, 1999.

[80] Connor Shorten and Taghi M. Khoshgoftaar. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1):60, December 2019.

[81] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pages 2951–2959, Red Hook, NY, USA, 2012. Curran Associates Inc. event-place: Lake Tahoe, Nevada.

[82] Derek Snow. DeltaPy: A Framework for Tabular Data Augmentation in Python. *SSRN Electronic Journal*, 2020.

[83] Parikshit Sondhi. Feature Construction Methods : A Survey. https://api.semanticscholar.org/CorpusID:15010720, Accessed: 2022-12-20 12:16:05, 2009.

[84] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. Annotating Columns with Pre-trained Language Models. In *Proceedings of the 2022 International Conference on Management of Data*, pages 1493–1503, Philadelphia PA USA, June 2022. ACM.

[85] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. 2012. Publisher: arXiv Version Number: 2.

[86] Lewis Tunstall, Leandro von Werra, and Thomas Wolf. *Natural language processing with transformers: building language applications with Hugging Face*. O'Reilly Media, Sebastopol, CA, first edition edition, 2022. OCLC: on1266359932.

[87] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in Python. *PeerJ*, 2:e453, 2014. Publisher: PeerJ Inc.

[88] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. 2017. Publisher: arXiv Version Number: 7.

[89] Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. Do NLP Models Know Numbers? Probing Numeracy in Embeddings, September 2019. arXiv:1909.07940 [cs].

[90] Daheng Wang, Prashant Shiralkar, Colin Lockard, Binxuan Huang, Xin Luna Dong, and Meng Jiang. TCN: Table Convolutional Network for Web Table Interpretation. In *Proceedings of the Web Conference 2021*, pages 4020–4032, Ljubljana Slovenia, April 2021. ACM.

[91] Dakuo Wang, Q. Vera Liao, Yunfeng Zhang, Udayan Khurana, Horst Samulowitz, Soya Park, Michael Muller, and Lisa Amini. How Much Automation Does a Data Scientist Want?, January 2021. Issue: arXiv:2101.03970 Number: arXiv:2101.03970 arXiv:2101.03970 [cs].

[92] Dakuo Wang, Justin D. Weisz, Michael Muller, Parikshit Ram, Werner Geyer, Casey Dugan, Yla Tausczik, Horst Samulowitz, and Alexander Gray. Human-AI Collaboration in Data Science: Exploring Data Scientists' Perceptions of Automated AI. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–24, November 2019. Number: CSCW.

[93] Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. TUTA: Tree-based Transformers for Generally Structured Table Pre-training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1780–1790, Virtual Event Singapore, August 2021. ACM.

[94] Jason Wei and Kai Zou. EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. 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)*, pages 6381–6387, Hong Kong, China, 2019. Association for Computational Linguistics.

[95] Sebastien C. Wong, Adam Gatt, Victor Stamatescu, and Mark D. McDonnell. Understanding data augmentation for classification: when to warp?, November 2016. arXiv:1609.08764 [cs].

[96] Yan Xia, Xudong Cao, Fang Wen, and Jian Sun. Well Begun Is Half Done: Generating High-Quality Seeds for Automatic Image Dataset Construction from Web. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, volume 8692, pages 387–400. Springer International Publishing, Cham, 2014. Series Title: Lecture Notes in Computer Science.

[97] Doris Xin, Eva Yiwei Wu, Doris Jung-Lin Lee, Niloufar Salehi, and Aditya Parameswaran. Whither AutoML? Understanding the Role of Automation in Machine Learning Workflows. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–16, Yokohama Japan, May 2021. ACM.

[98] Lei Xu, Shubhra Kanti Karmaker Santu, and Kalyan Veeramachaneni. ML-Friend: Interactive Prediction Task Recommendation for Event-Driven Time-Series Data, June 2019. Issue: arXiv:1906.12348 Number: arXiv:1906.12348 arXiv:1906.12348 [cs, stat].

[99] Kaichao You, Mingsheng Long, Jianmin Wang, and Michael I. Jordan. How Does Learning Rate Decay Help Modern Neural Networks? 2019. Publisher: arXiv Version Number: 2.

[100] Adams Wei Yu, David Dohan, Quoc Le, Thang Luong, Rui Zhao, and Kai Chen. Fast and Accurate Reading Comprehension by Combining Self-Attention and Convolution. In *International Conference on Learning Representations*, 2018.

[101] Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards Automated Deep Learning: Efficient Joint Neural Architecture and Hyperparameter Search. 2018. Publisher: arXiv Version Number: 1.

[102] Dan Zhang, Madelon Hulsebos, Yoshihiko Suhara, Çağatay Demiralp, Jinfeng Li, and Wang-Chiew Tan. Sato: contextual semantic type detection in tables. *Proceedings of the VLDB Endowment*, 13(12):1835–1848, August 2020.

[103] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. 2017. Publisher: arXiv Version Number: 2.

[104] Barret Zoph and Quoc V. Le. Neural Architecture Search with Reinforcement Learning, February 2017. arXiv:1611.01578 [cs].

[105] Marc-André Zöller and Marco F. Huber. Benchmark and Survey of Automated Machine Learning Frameworks, January 2021. Issue: arXiv:1904.12054 Number: arXiv:1904.12054 arXiv:1904.12054 [cs, stat].