# AI-enabled Healthcare CDT

## MRes Dissertation

## An Automated Approach for Identifying and Categorising Tables in Pharmacokinetic Literature

**Word Count: 15,607**

**October 27, 2021**

## Abstract

Pharmacokinetic-pharmacodynamic (PKPD) models estimate parameters to quantify the time course and effect of drugs. PKPD models are used to establish which drug candidates and dosing schedules will yield optimum efficacy and safety, throughout clinical trials and in the clinic. Prior information from similar drug compounds is used to make preclinical PKPD parameters predictions. However, large amounts of high-quality prior data are locked away in tables in the literature. Despite the rapid growth in PKPD literature, there is no publicly available, centralised database which collates information from these studies, nor a publicly available tool for filtering this information. Automated information extraction from tables has the potential to resolve this. This research focuses on the initial stage of automated table data extraction, classifying tables reported in PK literature by the parameters and covariates they contain. Table text is encoded into embedding matrices to provide input to a supervised machine learning pipeline. Different pipeline configurations are assessed with the final best model being a Convolutional Neural Network, which achieved a Micro-F1 score of 0.78, a Macro-F1 score on 0.66 and a Weighted-F1 score of 0.76 and outperformed a single expert annotator in terms of F1-score on four out of nine classes. This pipeline will aid researchers to efficiently filter usable PK estimates and provides the initial step towards automated table data extraction.

**Keywords:** Pharmacokinetics, Pharmacometrics, Natural Language Processing, Machine Learning, Information Extraction, Table Mining

# Contents

# Abbreviations

| | |
|---|---|
| **ADME** | Administration, Distribution, Metabolism and Elimination |
| **ANN** | Artificial Neural Network |
| **BoW** | Bag of Words |
| **CNN** | Convolutional Neural Network |
| **DDI** | Drug-Drug Interaction |
| **IE** | Information Extraction |
| **ML** | Machine Learning |
| **MLP** | Multilayer Perceptron |
| **NCA** | Non-Compartmental Analyses |
| **NLP** | Natural Language Processing |
| **PBPK** | Physiologically-based Pharmacokinetic |
| **PD** | Pharmacodynamic |
| **PK** | Pharmacokinetic |
| **PKPD** | Pharmacokinetic-Pharmacodynamic |
| **PMC** | PubMed Central |
| **PMOA** | PubMed Open Access |
| **QSAR** | Quantitative Structure-Activity Relationship |
| **RNN** | Recursive Neural Network |

# 1 Background

The expense of conducting clinical trials in order to bring new drugs to market, through high regulatory scrutiny, is a primary concern for pharmaceutical companies. The risk associated with bringing new drugs to market is also high, with failure rates of more than 85% at Phase I Clinical Trials.[1] The study of Pharmacokinetics and Pharmacodynamics (PKPD), contributes to selecting the best drug candidates in pre-clinical drug development, reducing attrition at clinical trials as a result of lack of efficacy or poor safety profiles.[2] Mathematical modelling approaches have become increasingly valuable in making data-driven decisions in drug discovery and development. However, despite extensive research in this area, the lack of centralised data repositories limits learning from similar drug compounds that can be brought to bear in the selection of new drug candidates.

## 1.1 PKPD Modelling: Approaches, Applications and Limitations

In order to understand the limitations of PKPD modelling, it is necessary to understand the different modelling approaches. The study of drug absorption, distribution, metabolism and excretion (ADME) processes is referred to as Pharmacokinetics (PK) and is described as "what the body does to the drug", whereas pharmacodynamics (PD) studies the intensity of a drug's effect in the body and is described as "what the drug does to the body". Variability in drug exposure (PK) and response (PD) can be quantified and explained using statistical and mathematical modelling. In PK models, ADME properties are inferred from modelling the relationships between dose and concentration in the body over time. In PD models, drug efficacy and toxicity are estimated by modelling response endpoints such as disease score or bacterial counts. These two modelling approaches are often integrated (PKPD modelling) to quantify the full dose–concentration–effect relationship for pre-clinical and clinical applications.[3] PKPD modelling involves a statistical aspect which tries to quantify variability in the data, as there is always an element of ‘noise’ when fitting a PKPD model.[4]

PK models relate known covariates (dosing time, dose, sampling time) with observed measurements (usually plasma concentration) to quantitatively represent the system, allowing PK parameters to be derived. PK parameters refer to a set of numerical factors which describe the relationship between drug concentration and pharmacological effect. A summary of PK parameters can be found in Table 1. Estimations of PK parameter are strongly influenced by covariates such as individual characteristics (age, weight, renal impairment, demographics) and study design (administration route, dose, dosing schedule).[3] Observed drug concentration measurements over time in plasma are required for a PK analysis, forming the basis of the PK concentration-time plot (Figure 1) from which absorption and elimination characteristics can be studied. To estimate PK parameters there are a number of modelling strategies.[5]

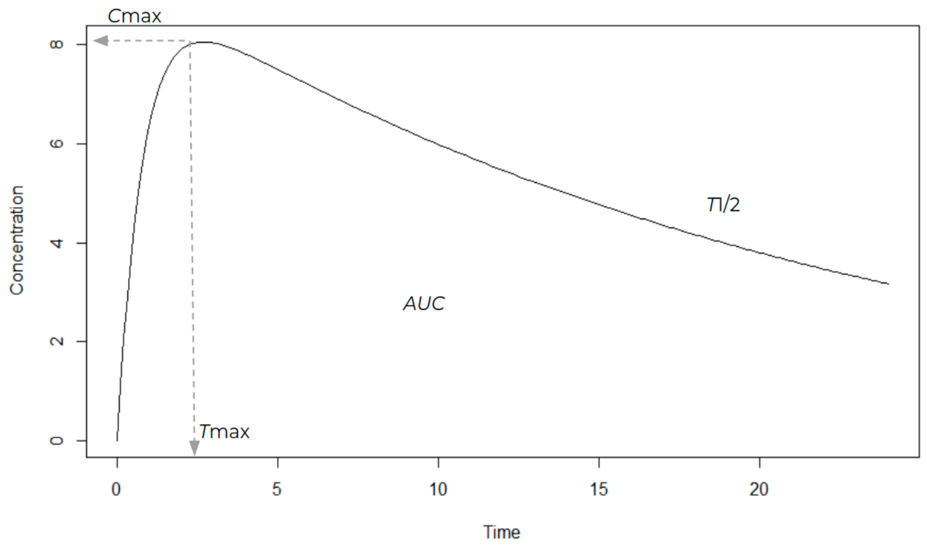


Figure 1: PK Concentration Time Curve. In a NCA this curve is used to calculate $AUC_{0-t}$ using the trapezoidal rule and then $AUC_{t-\infty}$ is calculated using $AUC_{t-\infty} = \frac{C_{last}}{k}$ where $C_{last}$ is the concentration of the last data point on the curve and k the elimination rate constant. $k = \frac{ln2}{t_{1/2}}$ and $t_{1/2}$ is determined using the last 3-4 observations from the terminal slope. Finally Clearance is calculated with the equation $CL = \frac{dose}{AUC_{0-\infty}}$, where $AUC_{0-\infty} = AUC_{0-t} + AUC_{t-\infty}$

**Table 1: Commonly Reported Pharmacokinetic Parameters**

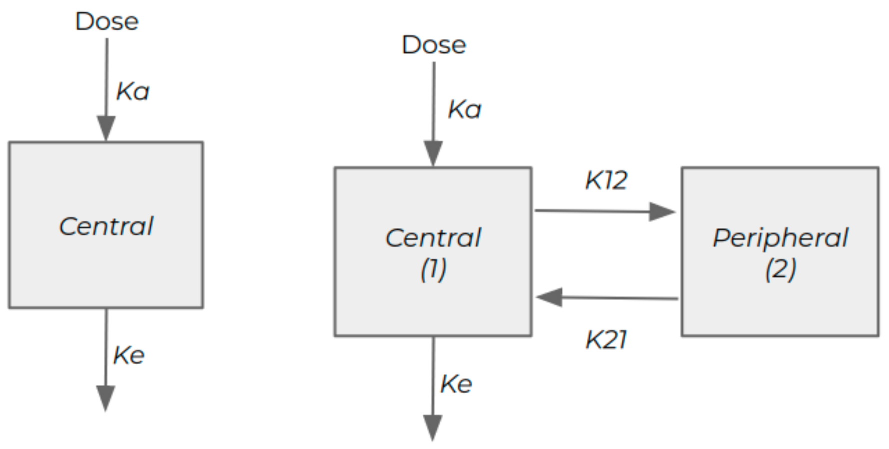
| **Parameter** | **Units** | **Description** |
|---|---|---|
| $C_{max}$ | mass/volume | Maximum Concentration: directly obtained as the peak concentration reached by the concentration-time curve |
| $t_{max}$ | time | Time to reach $C_{max}$: directly obtained as time at peak concentration reached by concentration-time curve |
| $t_{1/2}$ | time | Elimination Half-life: time taken for a 50% reduction in drug concentration in plasma |
| $AUC$ | mass·time/volume | Area under the concentration-time curve: the integral below the concentration-time curve is computed and either reported from administration to time of final measurement ($AUC_{0-t}$) or extrapolated to infinity ($AUC_{0-\infty}$) |
| $F$ | no units | Bioavailability: proportion of administered dose which reaches the systemic circulation, 1 for Intravenously administered drugs but otherwise estimated in PK analysis using AUC of both intravenous administration and extra-vascular administration route and the dose administered |
| $V_d$ | volume | Volume of distribution: the necessary volume to achieve a certain concentration of drug measured in plasma, given a dose |
| $Cl$ | volume/time | Clearance: the rate of elimination of a particular substance divided by its concentration in plasma (by various routes e.g. hepatic, renal etc.) |

**Notes:** Please note, additional PK parameters and units are found in scientific literature. The ones stated above represent the commonly used parameters.[6]

### 1.1.1 PKPD Modelling Approaches

Non-compartmental Analyses (NCA) are empirical models which assume a uniform distribution of administered dose throughout the body and use a rate constant to define drug elimination. In the concentration-time plot (Figure 1), the time taken for a drug to reach $C_{max}$ indicates the speed of absorption of a drug and the terminal slope provides information on the speed of drug elimination from the body. Looking at curves from a number of individuals allows determination of variability across the study population. $AUC_{0-t}$, representing total drug exposure, can be estimated directly from the concentration-time plot using the trapezoidal rule. The elimination rate constant (k) can be calculated using the last 3-4 observations from the terminal slope. Further PK parameters can subsequently be estimated using algebraic equations (Figure 1).[7] NCA is a simple and rapid approach to generate PK parameter estimates, in practise being partly automated by software packages such as NONMEM. However, NCA relies on rich sampling schedules, meaning there are measurements taken at many time points per individual, for use of the trapezoidal rule and it does not account for biological processes or physiological mechanisms which affect PK.[7] For this reason, NCA is most commonly used to determine drug exposure in a single study with rich sample data for example estimating preclinical PK parameters of a drug in a toxicology studies or in early phase clinical trials.

Compartmental Models assume that the body consists of a finite number of well-stirred, interconnected, kinetically homogeneous compartments for example one being the blood and another the tissues and organs. After intravenous administration of a drug, it enters the central compartment (the blood) and distribution and elimination occur. In a one-compartment model, the entire body (blood, tissues and organs) is assumed to act as a single uniform compartment (Figure 2a) and the model describes a mono-exponential decline. A two- or three-compartment model separates blood and well-perfused organs from tissues (Figure 2b), producing a PK curve with multi-exponential decay.[8] Parameters are estimated from the study population data based on nonlinear regression methods, to minimise the loss function and find the most appropriate equation to fit the data. Compartmental models are based on assumptions, introducing variability between analyses.



(a) One-Compartment Model (b) Two-Compartment Model

**Figure 2: Compartmental Models, where $Ke$ is the first-order elimination rate constant and $Ka$ is the first-order absorption rate constant, K12 is the first-order transfer rate constant from central to peripheral compartment and K21 is the first-order transfer rate constant from peripheral to central compartment.**

However, compartmental methods boast key advantages over NCA having the ability to cope with sparse data and to explore PK variability resulting from covariate factors (for example renal impairment, sex, age), inter-individual variability and inter-occasion variability (differences in the same subject from one dose to another).[8,9]

PD models describe pharmacological effect as a function of drug concentration and are also usually described by compartmental models. The delay observed between the drug reaching its maximum concentration and a therapeutic effect, is attributed to a delay from drug in the plasma being distributed to it's effect site and so compartmental modelling is used to model plasma and effect site compartments.[9] PD models are used to estimate PD parameters such as maximum effect ($E_{max}$), the drug concentration at which the maximum effect is achieved.

Physiologically-based PK (PBPK) models build further upon compartmental models, incorporating detailed knowledge on blood flow, tissue composition of organs and drug properties to simulate drug exposure and response in different contexts. This approach using biological prior information can even predict drug PK parameters before conducting any *in vivo* studies by changing the physiological parameters of the model depending on species and disease status.[10] In fact, Regulatory Agencies, in some cases, now accept

PBPK modelling alone in place human studies, for example in lieu of special population studies (e.g. paediatric) or drug-drug interaction studies.[11]

### 1.1.2 PKPD Modelling Applications

In drug development, PK modelling is used to select for drugs with oral bio-availability (for orally delivered therapies) and half-lives which can reach pharmacological effect with a reasonable dosing schedule, whilst minimising DDI potential. PK modelling is employed at various phases to understand the dose-concentration relationship across preclinical studies, for establishing a safe first-in-man dose, to select the safest doses in Phase I/II trials and finally in Phase III trials, to optimise the dose for efficacy in the target population. Furthermore, in a clinical setting, PK modelling is used to determine off-label dosing (unapproved dosing of an approved drug) for example to individualise doses based on specific biomarkers or for special populations (e.g. paediatrics).[3] An iterative approach is employed in PK model building to find the most appropriate model. Preclinically, when there is no human data available, prior mechanistic knowledge from similar compounds are often used to project anticipated response 'in man' and expected therapeutic range. Consequently, the confidence of preclinical predictions depends to a considerable extent on availability and quality of prior PK information from related compounds. This information is largely gathered from the scientific literature and manually curated databases.[12]

### 1.1.3 PKPD Modelling Limitations

Sub-optimal PK properties are one of the contributors to attrition (failure) of potential new drugs early in clinical trials.[13] Additionally, PK model building is a laborious iterative process which requires expert input and is also computationally intensive.[14,15] Attempts to automate parts of this process, such as employing search algorithms to find optimal covariates and model configurations, has only served to increase computational burden further. To reduce drug attrition and the associated cost, there is rising interest in *in silico* approaches which allow virtual PK screening of compounds before their chemical synthesis.[15,16] Quantitative Structure-Activity Relationship (QSAR) modelling is one of these approaches, using machine learning (ML) to map structural and physiological properties of a compound to its biological activity, for example to predict a drug's binding affinity for it's biological target.[17] However, more recently, QSAR approaches have been applied to predicting in human PK profiles of new compounds given their molecular properties.[14, 18, 19] Developing robust QSAR models is a key development area to successfully predict PK properties before human trials. However, performance of PK predictive models is highly dependent on the availability of large databases with information on drug chemical properties and clinically derived PK parameters on which a model can be developed. However, currently there is no centralised PK database providing the quality or quantity of data required for this approach.

### 1.1.4 PKPD Data Sources

The availability of extensive and standardised data is a key barrier to improving *in vivo* PK predictions of new potential therapies. There are a number of public databases containing information on various molecular properties and ADME data for numerous compounds, including but not limited to DrugBank,[20] PubChem,[21] PharmGKB[22] and ChEMBL.[23] However, these databases often lack ADME data with enough detail to directly integrate into PK parameter prediction.[24] A study of 31 ADME databases found the majority reported *in vitro* PK properties, and only a few reported human PK parameters. For those reporting human PK information, data was often sparse, meaning there are few measurements per individual, and rarely specified information on modelling approaches and study design,[25] which forms the basis of interpreting PK estimates.[6] Manually curated databases specifically reporting human ADME information, such as PK-DB[26] or the data set curated by Lombardo *et al.*,[27] have limited numbers of studies, drugs and administration routes included and once again lack information on potential sources of variability such as modelling approaches, covariate information and study design,[26, 27] making interpretation and reuse of results difficult. In addition, most pharmacological databases are maintained manually, which cannot meet the demands of the constantly growing PK literature.

More than 121K relevant PK publications were identified on PubMed in early 2021, using a previously developed PK document classifier,[28] demonstrating the challenges of a manual approach.[6,29] Consequently, for a specific PK parameter prediction task, information is usually extracted manually from a small number of papers and pooled with any data available in publicly available databases or in-house data sets. The pooling alone is a highly laborious and time-consuming task for Pharmacokineticists to perform, limiting the quantity and quality of data they can collect.[14] This is largely due to the unstructured nature of scientific literature in general, specifically PK prior information, frequently presented in tables in the literature and in a format that is not machine-readable. Hence, despite the potential PK data contained in scientific articles, exploiting this source of information is still a major challenge for drug development. Automated information extraction holds potential to aid researchers to retrieve PK information reported in PK literature with greater speed and ease in order to generate larger data sets for more predictive PKPD modelling.

## 1.2 Automated Information Extraction

Text mining approaches have been used widely on free-text literature to retrieve and classify scientific information. However, extraction of information presented in tables in the literature, in which rich PK data are found, has, to date, been largely overlooked.[30] Text mining and table mining aim to derive high-quality information from text and tables respectively. One of the principal methodologies behind text mining is Natural Language Processing (NLP), a subfield of ML which is concerned with representing the meaning of text with computational techniques. Text mining can be split into two main steps: (1) identifying relevant text instances (e.g. documents, tables etc.), known as information retrieval (IR) and (2) locating and extracting specific instances, known as information extraction (IE).[31] Hurst[32] and Milosevic *et al.*[30] propose breaking down table mining in a similar manner to include table detection (the equivalent of IR), classifying table regions (e.g., header cell, data cell), and named entity recognition of entities within table cells (the equivalent of IE) and finally, understanding the relation between cells. Kardas *et al.*[33] approach table mining in a similar way, initially carrying out table classification to identify relevant tables (IR) before segmenting table regions and recognising the entities in each cell (IE). A number of approaches can be applied to IR and IE including pattern-based and statistical approaches.

### 1.2.1 Pattern Based Approaches

Pattern-based approaches, locate specific occurrences in text, based on dictionaries and hand-crafted rules. In Biomedicine, pattern-based approaches have been used extensively to detect entities pertaining to Proteins, Genes and Drugs.[34,35] However, some of their main challenges include the diverse naming conventions, synonyms, and newly published terms. In addition, developing patterns is laborious and requires expert domain knowledge. In the pharmacology domain, early approaches were all pattern-based.[36,37] Specifically, in PK, rule-based approaches have been applied to extract numerical values from text associated with oral or systemic clearance, in relation to a single drug in human volunteers.[29] PK text is highly heterogenous, including extensive use of acronyms and abbreviations and highly domain-specific terminology. This represents a significant challenge for rule-based text mining approaches which struggle to deal effectively with the diversity of PK scenarios.

### 1.2.2 ML Approaches

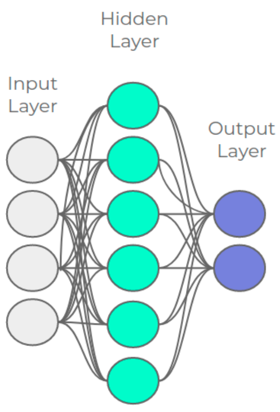
In contrast, ML approaches model complex rules and are particularly suited to deal with diversity. The goal of ML approaches is to understand the distribution of input text and develop predictive models for a specific tasks. Given a large number of examples (training data), ML algorithms build mathematical models to perform future predictions by inferring specific patterns from the data.[38] Textual patterns with high variability (e.g. multiple ways of expressing the same pk parameter) can be particularly efficiently modelled by ML. However, ML requires sufficient, high-quality training data. In the text-mining field, kernel-based Support Vector Machines methods were successfully applied to classify DDI mentions in a specially curated DDI corpus consisting on 1025 manually annotated documents from DrugBank and MedLine.[39,40] However, such a corpus is not available for PK parameters and covariates and this method relies on feature engineering.

Deep learning, however, allows high level representations of input data to be learnt automatically, bringing a breakthrough in NLP.[41] Deep Learning refers to ML based on artificial neural networks (ANNs), an architecture mimicking the human visual cortex, where processing data across multiple network layers extracts progressively higher level features. In supervised learning, labeled data is used to train the ANN to classify data correctly. In the forward pass, the input is fed through the sequence of layers and the output labels are measured against the true labels. In the backward pass, the weights and biases (or parameters) are adjusted to minimise error (between the model predictions and the true labels) through the network, known as back-propagation. Given the almost universal illustrated performance of deep learning, it has become a dominant method for biomedical NLP, generally outperforming traditional kernel-based and feature-based supervised methods, even with simple architectures.[37, 42, 43]

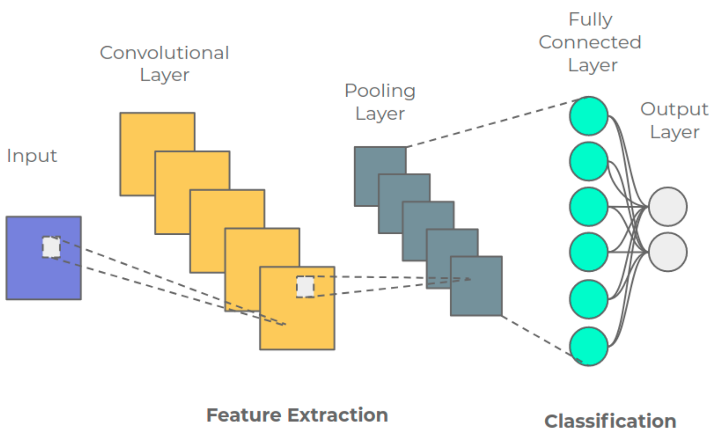
Multilayer Perceptrons (MLPs), Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), are three main types of ANN architectures, and have been explored widely across various NLP tasks. MLPs, are fully connected layers of neurons stacked into a feed-forward network. MLP architecture is composed of: (i) an input layer, consisting of a set of neurons representing the input features, (ii) an arbitrary number of hidden layers, which transforms values from the input layer with a weighted linear summation followed by a non-linear activation function (capable of approximating any continuous function) and (ii) an output layer which outputs the classification (Figure 3a). MLPs take a vector as input so cannot understand spatial relation of input features. CNNs, however, take a tensor as input and can therefore consider spatial relation of features. At the basic level CNNs are composed of: (i) a convolutional layer, which repeatedly applies a filter to the input, calculating the dot product to yield local feature maps (extracting relevant features), (ii) a pooling layer, obtaining translation invariant description and (iii) a fully connected layer for classification prediction (Figure 3b). As each filter is panned around the entire input tensor, CNNs are able to detect features in the input text, for example key terms or phrases, irrelevant of their position. However, MLPs and CNNs only consider the current input and so have no notion of order in a sequence. Lastly, RNNs can be thought of as a sequential architecture, where the current input is considered alongside what has been learned from the inputs received previously, making an RNN able to capture sequential information in input text (such as dependencies between words in a sentence). In Figure 3c, at time point 0 the output is y0, so when training the network at time point 1, y0, the output from the previous time-frame, is also considered in the current time-frame and backpropagation then updates the weights for both of these time-stamps. In a comparative study on textual data, it was found RNNs (specifically Long Short-Term Memory networks (LSTMs) and Gated recurrent units (GRUs)) prevail when there is a need to semantically understand the whole sequence, whereas CNNs have an advantage with position invariant features (essentially keyword recognition) or in cases of long sequences.[44] Adel and Schutze[45] and Wen *et al.*[46] support CNNs over RNNs (GRUs/LSTMs) for classification of long sentences, although results depend on model hyperparameters.[44] In this regard, CNNs are likely more suited to table classification, where sequences are long and where it is expected that semantic interpretation of the sequence is not as important as position invariant queues.

### 1.2.3 Table Mining Approaches

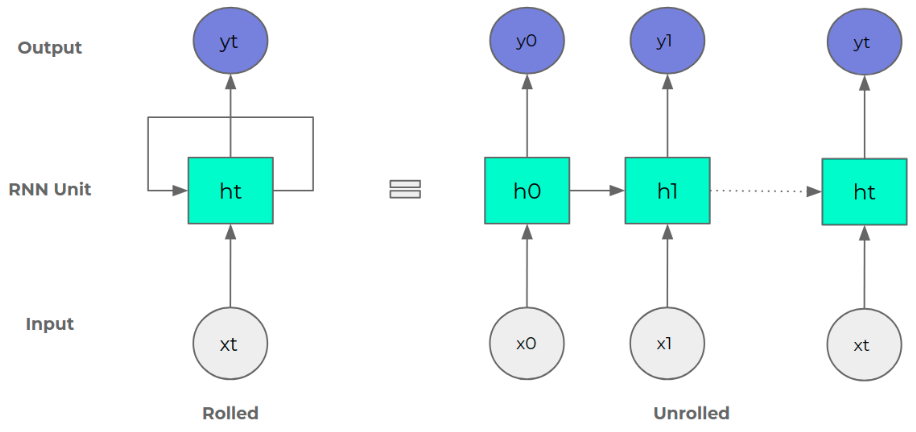
Table mining has received relatively little attention in the biomedical domain[30,47,48] and to my best knowledge there are no published work on this in the PK domain. However, in computer science fields, table mining is receiving increasing attention and significant progress has been made in detecting tables and recognising cell structure.[49–52] Computer vision based approaches boast the ability to detect tables within PDF documents where a wealth of scientific data is stored[50,52] and the ICDAR table mining competition focuses on PDF documents.[53] However, PubTabNet, an encoder-decoder neural network developed by Zhong *et al.*[51] converts images of tables into HTML code, making it possible for models built for HTML/XML to be applied to tables in PDF format. Scientific articles on PubMed Central use the PMC-XML format, meaning table mining can be framed as an NLP task. Milosevic *et al.*[30] conducted thorough work on table mining from PMC-XML data, using a combination of rule-based and ML approaches to analyse table cell contents and fill in an extraction template in a domain independent manner. However they note that for domains with complex table structure (such as PK), an ML pipeline trained on domain specific tables is needed. Kardas *et al.*[33] created an end-to-end systems



(a) MLP



(b) CNN



(c) RNN

**Figure 3: Diagrams of Basic (a) MLP, (b) CNN and (c) RNN Architectures. In (c) h refers to RNN units (not sequential units but the same unit at different time points), x refers to the inputs, y to the outputs and t,0,1 in all cases refer to the timepoints.**

to extract results from ML papers in LATEX text format, by first performing table type classification followed by table segmentation. Table classification was performed on captions alone using a pre-trained language model, ULMFit,[33] an RNN-like architecture, to best process sequential information in the caption. Hou *et al.*[54] frame ML results extraction from tables as an NLP problem, extracting the table into text and identifying and classifying Task-Dataset-Metric triples using a transformer model based on the pre-trained deep neural networks Bidirectional Encoder Representations from transformer (BERT), the current state-of-the-art in NLP.[55]

For drug discovery and development and clinical dosing, high quality, thorough PK data sets are required to improve parameter predictions and boost efficacy and safety of new drug candidates and reduce drug attrition. Automated IE holds potential to help PK researchers to filter a large amount of irrelevant literature and efficiently find results tables reporting useful prior *in vivo* PK parameters and covariate information for building predictive models.

# 2 Rationale

Bringing new drugs to market is both a costly and high risk process, with high failure rates at Phase I Clinical Trials, resulting in drug attrition.[1] Mathematical modelling of PKPD processes has become increasingly recognised for contributing to selection of the optimum drug candidates and dosing schedule across the clinical trials process and beyond. However, high quality, thorough ADME data sets are required to further improve PK parameter predictions and boost efficacy and safety of new drug candidates to reduce drug attrition.

To initialise PK models, prior mechanistic knowledge from similar compounds is typically used. However, despite the extensive research in this area, PK parameter predictions are limited by the lack of centralised data repositories.[14,15] Existing databases rarely contain rich *in vivo* PK parameters and sufficient covariate information for model building.[25] Instead, rich PK data is found in the literature in text format, often displayed most comprehensively in tables. Manually curating data from the literature and other sources is a laborious task for PK researchers, limiting the amount of data they can collect for predictive modelling.

Automatic information extraction (IE) methods have the capability to exploit the enormous amount of information in existing and new publications.[6,29] The highly heterogeneous nature of PK data makes ML methods the preferred approach to this problem. ML approaches have been widely used to classify scientific free-text from literature, and specifically with some success in the PK domain.[37,39,56] Despite advances in table mining methodology in textual data using NLP techniques,[33,54] to the best of my knowledge there are no published works on table mining for PK tables. Widely, table mining is split into two tasks (1) table recognition (classification as relevant) (2) recognising and extracting specific entities within the table into a structured format.[30,33] Automatic IE, using NLP and ML approaches, provides the tool-set to build an IE pipeline for tables in PK literature. This will make prior PK data, for PK parameter predictions, more accessible, benefiting both drug discovery and development and clinical dosing.

## 2.1 Aims and Objectives

To tackle the first step of table mining, this project aims to develop: (a) a labelled corpus of tables from PK papers and (b) a supervised ML pipeline to extract and classify tables from PK literature by the parameters and covariates they contain. Extraction is framed as an NLP task, accessing tables in textual format and comparing two neural network architectures for multi-label classification. Success of the pipeline is considered both quantitatively and qualitatively, as a combination of performance across classes, on individual classes and in comparison to the mean individual annotators performance (please see below for details). The pipeline aims to provide the following benefits to the PK community: (i) to help researchers to search more efficiently for PK parameters and corresponding covariate information for clinical PK modelling and drug discovery and (ii) to provide the initial step in a PK table data extraction pipeline which can be built upon to create an automatically updating centralised database, to cope with the ever-expanding literature.

To achieve (a) the specific objectives are as follows:

1. Extract Tables from PK Papers.
2. Design a Annotation Interface for PK tables and corresponding Annotation Guidelines.
3. Manage Expert Annotators to deliver upwards of 2500 labelled examples, in an iterative approach, resolving disagreements and updating guidelines accordingly.

To achieve (b) the specific objectives are as follows:

1. Investigate the best tokenization strategy and train a Tokenizer on table text.
2. Investigate the optimum preprocessing strategy for table text.
3. Investigate the optimum table representation strategy for table text.
4. Compare a simple Multilayer Perceptron (MLP) and a Convolutional Neural Network (CNN) on this task.

5. Investigate methods to deal with class imbalance in the data set.

6. Select a final best pipeline and tune model hyperparameters.

7. Assess final pipeline performance on unseen test data set. Success is a model that achieves high performance on classification of individual classes in terms of Precision, Recall and F1-score and across classes as measured with the summary performance metrics Micro-, Macro- and Weighted-F1 scores.

8. Carry out a Qualitative Assessment on a subset of test set tables (observing labels vs predictions) to ensure the model is behaving as expected on unseen data.

9. Calculate mean annotator F1-score across all annotators and compare to the final agreed true labels, and directly compare the performance of the model to that of a single pharmacologist. The classifier pipeline is considered effective if it achieves F1-score per class on unseen data which is in the same order of magnitude as the mean expert PK annotator.

# 3 Method

A supervised machine learning approach was developed to classify tables in Pharmacokinetic (PK) papers depending on the parameters and covariates they contained. To do so, annotated data was collected from PK experts (Corpus Development) and a classification pipeline was developed (Pipeline Development). Model code can be found in the Appendix (Github not shared to maintain anonymity).

## 3.1 Corpus Development

Tables were extracted from scientific papers and manually annotated for the PK parameters and covariates they contained in a multi-label approach.

### 3.1.1 Data Source and Parsing

Firstly, all papers were downloaded from the PubMed Open Access (PMOA) Database, from the official PubMed baseline and updatefiles FTP sites. This is a subset of Pubmed Central (PMC) articles, published under Creative Commons or similar licences, allowing reuse and redistribution of the full articles. There are no causes for concern from a privacy or ethical perspective with using data from this source.

Secondly, PK relevant papers (reporting novel PK parameters from *in vivo studies*) were selected from this corpus using the publicly available document classifier previously developed by Hernandez *et al.*[28] Tables were extracted from these papers in PMC-XML format by selecting the content of specific HTML table tags, pertaining to the structure of the tables (caption, rows etc.). The tables across PK documents were shuffled and a random selection of 3000 tables from the whole corpus was selected for annotation.

### 3.1.2 Size

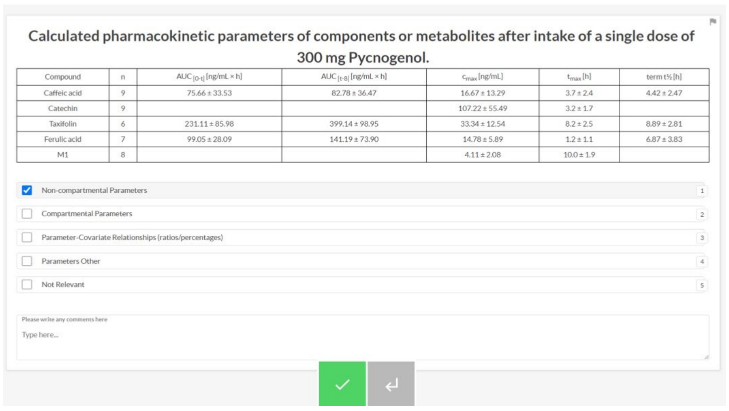
Final Training, Validation and Test sets of size 1498, 341 and 783 samples respectively were developed to train and evaluate performance with the different classification approaches. The size of these data sets was based on annotator availability, to ensure no less than three annotators per sample on the test and Validation sets and at least two per sample on the Training set.

### 3.1.3 Annotation

A custom labelling interface was developed in python using Prodigy Annotation Software (Figure 4) and set up to allow annotators to label via web-links. The labelling process was carried out by two Pharmacometricians from University College London, two Pharmacometricians from Mahidol Oxford Tropical Medicine Research Unit and one Clinical Pharmacokineticist from AstraZeneca. Labelling guidelines were provided to maximise consistency across annotators. Initially, at least two annotators labelled each table in the Training set, and at least three annotators labelled tables in the test and Validation sets. Review sessions with the full annotation team were held after each annotation iteration to resolve disagreements. Conflicting opinions or exceptions did occur and the labelling guidelines were iteratively updated to capture the resolutions to these to inform future labelling decisions to improve consistency. For each label, Cohen's Kappa Coefficient (K, Equation 1) was calculated, prior to resolution, to evaluate agreement between annotators. K compares the the agreement expected by chance ($p_e$) with the agreement observed between two annotators ($p_o$) on the true class (positive for a certain label).

$$K = \frac{(p_o - p_e)}{(1 - p_e)} \quad (1)$$

The labels cover PK parameter types from different PK analyses and different covariate types (Table 2). Two rounds of labelling is carried out for each table, firstly for PK parameters (Figure 4a) and then for covariates (Figure 4b) to allow annotators to focus on one aspect at a time to improve labelling speed and consistency. Labels are not mutually exclusive, meaning more than one label can be assigned to each table simultaneously, as tables can contain a variety of both covariate and parameter information.


Calculated pharmacokinetic parameters of components or metabolites after intake of a single dose of 300 mg Pycnogenol.

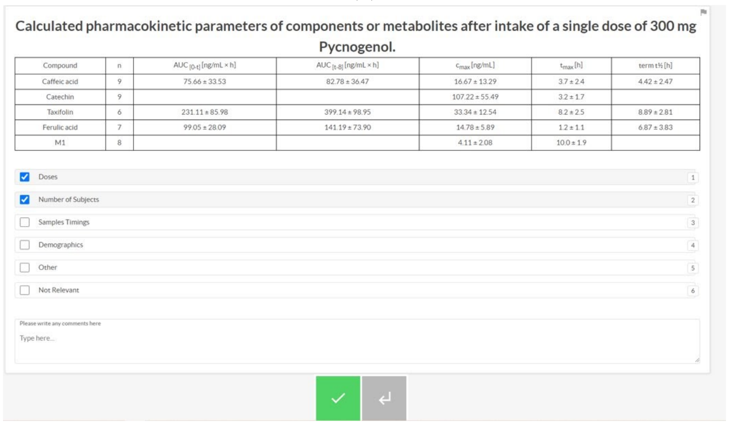
| Compound | n | AUC $_{[0\text{-}t]}$ [ng/mL × h] | AUC $_{[t\text{-}8]}$ [ng/mL × h] | $c_{max}$ [ng/mL] | $t_{max}$ [h] | term t½ [h] |
|---|---|---|---|---|---|---|
| Caffeic acid | 9 | 75.66 ± 33.53 | 82.78 ± 36.47 | 16.67 ± 13.29 | 3.7 ± 2.4 | 4.42 ± 2.47 |
| Catechin | 9 | | | 107.22 ± 55.49 | 3.2 ± 1.7 | |
| Taxifolin | 6 | 231.11 ± 85.98 | 399.14 ± 98.95 | 33.34 ± 12.54 | 8.2 ± 2.5 | 8.89 ± 2.81 |
| Ferulic acid | 7 | 99.05 ± 28.09 | 141.19 ± 73.90 | 14.78 ± 5.89 | 1.2 ± 1.1 | 6.87 ± 3.83 |
| M1 | 8 | | | 4.11 ± 2.08 | 10.0 ± 1.9 | |

- [✓] Non-compartmental Parameters 1
- [ ] Compartmental Parameters 2
- [ ] Parameter-Covariate Relationships (ratios/percentages) 3
- [ ] Parameters Other 4
- [ ] Not Relevant 5

Please write any comments here

Type here...

(a)


Calculated pharmacokinetic parameters of components or metabolites after intake of a single dose of 300 mg Pycnogenol.

| Compound | n | AUC $_{[0\text{-}t]}$ [ng/mL × h] | AUC $_{[t\text{-}8]}$ [ng/mL × h] | $c_{max}$ [ng/mL] | $t_{max}$ [h] | term t½ [h] |
|---|---|---|---|---|---|---|
| Caffeic acid | 9 | 75.66 ± 33.53 | 82.78 ± 36.47 | 16.67 ± 13.29 | 3.7 ± 2.4 | 4.42 ± 2.47 |
| Catechin | 9 | | | 107.22 ± 55.49 | 3.2 ± 1.7 | |
| Taxifolin | 6 | 231.11 ± 85.98 | 399.14 ± 98.95 | 33.34 ± 12.54 | 8.2 ± 2.5 | 8.89 ± 2.81 |
| Ferulic acid | 7 | 99.05 ± 28.09 | 141.19 ± 73.90 | 14.78 ± 5.89 | 1.2 ± 1.1 | 6.87 ± 3.83 |
| M1 | 8 | | | 4.11 ± 2.08 | 10.0 ± 1.9 | |

- [✓] Doses 1
- [✓] Number of Subjects 2
- [ ] Samples Timings 3
- [ ] Demographics 4
- [ ] Other 5
- [ ] Not Relevant 6

Please write any comments here

Type here...

(b)

Figure 4: Labelling Interfaces for Parameters (A) and Covariates (B), labels are not mutually exclusive.

**Table 2: Label Descriptions, Label is selected if Table or Caption meet the criteria of the given description**

| **Label** | **Description** |
|---|---|
| Non-Compartmental Parameters | Include PK parameters from non-compartmental analyses e.g. $AUC$, $Cmax$, $Tmax$, $Cmin$ |
| Compartmental Parameters | Include PK parameters from compartmental analyses e.g. Volume, Clearance, Micro and Macro rate constants |
| Parameter-Covariate Relationships | Include Ratios or percentage changes in parameters based on different covariate conditions |
| Parameters Other | Include PK parameters from PBPK analyses or Pharmacodynamic parameters e.g. $Emax$, $EC50$ |
| Doses | Include dose values and units |
| Number of Subjects | Include number of subjects in a study |
| Sample Timings | Include timing of samples taken after a dose was given |
| Demographics | Include demographic covariates e.g. species, age, sex, weight, height, creatinine clearance, liver function |
| Covariates Other | Include any other covariates not specified already e.g. administration routes, states (fed vs fasted), diets, genetic polymorphisms |

## 3.2 Pipeline Development

Different pre-processing steps, architectures and hyper-parameter combinations were compared to optimise PK table classification. A summary of the steps involved in the pipeline can be seen in Figure 5.

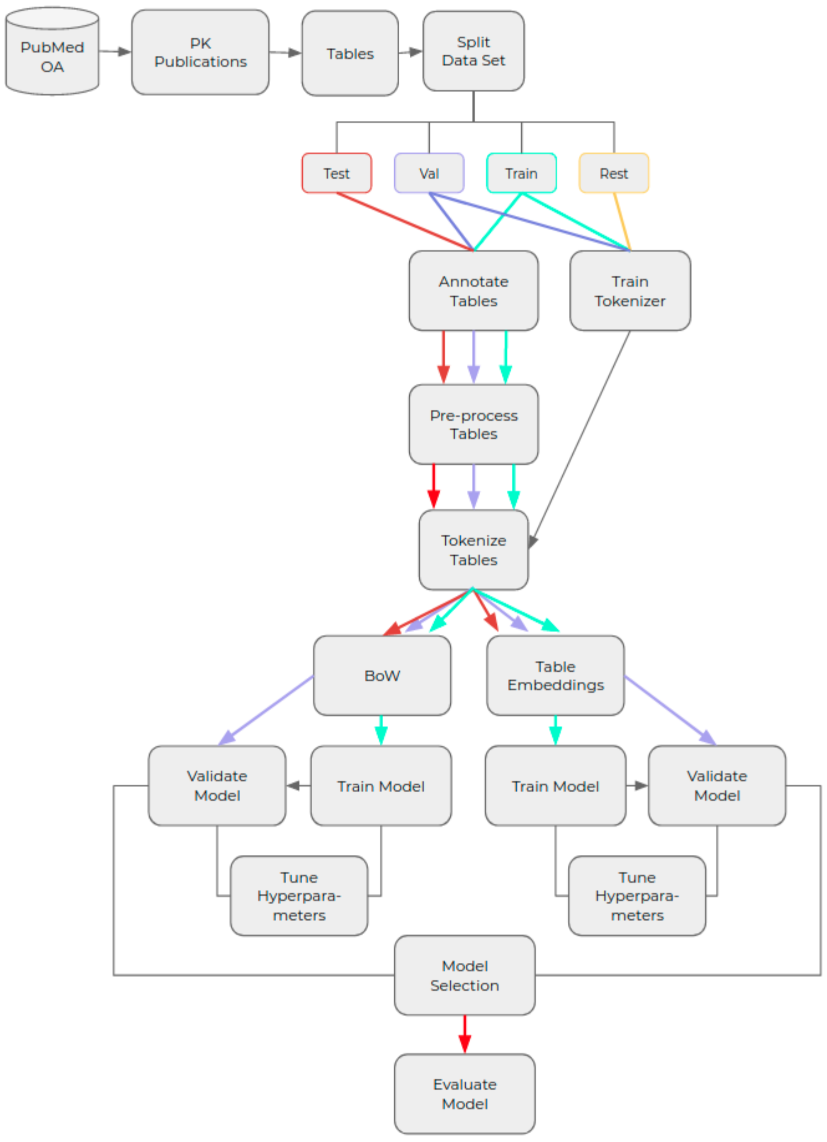


**Figure 5: Summary of the Machine Learning Pipeline for Pharmacokinetic Table Classification**

### 3.2.1 Preprocessing

Tables in PMC-XML format contain a number of HTML tags pertaining to table structure, for example marking headers, rows and cells of the table. As such table length can be very long (up to 7000 tokens). Passing too much information to a classifier can result in poor performance due to excessive noise in the data. However, relevant information can occur far down into the tables and HTML structural tags may also provide positional information which is important for table classification. To investigate this trade-off between including all relevant information and reducing noise, a number of preprocessing variations were applied to tables and classification performance was compared across these:

1. All HTML tokens were removed and only the caption, header row and first column were used.
2. All HTML tokens were removed and whole table text was used.
3. All HTML tokens were removed and special tokens were added to mark areas of caption, header row, first column and the rest of the body of the table.
4. HTML tokens pertaining to style were removed as these hold no relevance to table interpretation. HTML tokens representing table structure were kept in case these location queues provided important information for table interpretation.

### 3.2.2 Tokenization

Tokenization, refers to separating text into small meaningful units called tokens and is the primary step in modelling textual data. Due to the specialised vocabulary in the corpus, Tokenizers were trained from scratch. The Google SentencePiece sub-word Tokenizer was employed with the Hugging Face implementation. The core concept behind sub-word tokenization is that frequently occurring words should be in the vocabulary whereas rare words should be split into frequently occurring sub-words. This helps to avoid the exploding vocabulary problem of splitting words based on white space, whilst being able to learn meaningful context-independent representations, unlike in character level tokenization. To achieve these benefits, during training SentencePiece performs the following steps: (i) all characters including white spaces are encoded into unicode characters, therefore considering the text input as a raw input stream, (ii) Byte Pair Encoding (BPE) is employed to construct the best vocabulary for the training text. BPE involves initiating a token list with all unique characters from the training corpus and their frequency. Then using the most frequently occurring character pairs, characters are merged and these new merged characters are added to the list of tokens and their frequencies are subtracted from the character frequencies. This process occurs iteratively, further merging characters into sub-words and recalculating token frequencies until the maximum vocabulary size is met. Within the vocabulary each token now has a unique ID and during encoding with the trained Tokenizer, new input text is converted into a list of IDs corresponding to the sub-words found in the text.

SentencePiece Tokenizers were trained on (A) tables and captions from the whole of the PMOA data set minus those tables in the Test Set- 162,8358 tables (B) tables and captions from the PK subset of PMOA only minus those tables in the Test Set- 13,334 tables. The Tokenizer vocabulary size is a hyperparameter to tune. Reducing the size of the vocabulary means more words are represented as larger sub-word chunks, which can lead to better performance, by reducing the input size to the classifier. However, this needs to be balanced with making the vocabulary size so large that the Tokenizer is not generalisable to new inputs. Words that are not sufficiently represented within the vocabulary will result in information loss as will be marked as 'unknown'. As the corpus size is relatively small, a number of vocabulary sizes ranging from 3000-10,000 were compared. Unique HTML table tokens were obtained from the corpus and added to the Tokenizer as special vocabulary so that these are not split into sub-words during tokenization to allow tokenization of whole tables with minimal pre-processing.

### 3.2.3 Representation

As a baseline, tables were encoded as a Bag of Words (BoW). This is a sparse word representation which represents tables as fixed length vectors, with a length of the total vocabulary size from the Tokenizer used. If a token appeared in the input table, so its frequency is filled in in the corresponding place in the fixed length vector to produce a final multi-hot encoded vector (Figure 6a). Given an input table, the trained SentencePiece Tokenizer was applied to encode the table into a list of ids before converting this to a BoW vector.The BoW model treats all tokens as independent features and therefore does not consider context in the input text sequence.

Embeddings however, are a distributed representation for text which allow words with similar meanings to have a similar representation by mapping long, sparse vectors to short, dense, continuous vectors. Individual sub-words from tokenization are represented as dense fixed-size vectors which encode contextual and semantic information in a predefined vector space, which is lower in dimensionality than the total vocabulary size. These subword embeddings from the input table are grouped to produce an embedding matrix to represent the table with dimensions of embedding size and number of sub-words in the table (Figure 6b). An embedding layer is applied at the front end of the neural network classifiers to allow sub-word embedding so to be learnt jointly with the neural network in a supervised way, using back-propagation. In order to produce a fixed size embedding matrix to input into the classifier, padding and truncation are used. A maximum length for the input table is set and for any one table, if the number of tokens exceeds this length, so the table in truncated to the correct size. Otherwise, if the table is too short a special padding token is added to fill the input table up to the maximum length.

### 3.2.4 Class Imbalance

Arbitrary weighting of the loss function based on class frequency in the Training set does not necessarily represent class importance. Therefore, this is likely to create a situation where the classifier will over-fit to minority classes in cases of highly unbalanced data such as this. It is therefore preferable to increase the amount of data for training the model up front with approaches such as data augmentation. Data augmentation was carried out on the Training set to create more synthetic training samples. Data augmentation refers to increasing the diversity and size of a Training set by applying random but realistic transformations. In this case augmented samples were added back into the original training data set to increase the total training data set size. Two

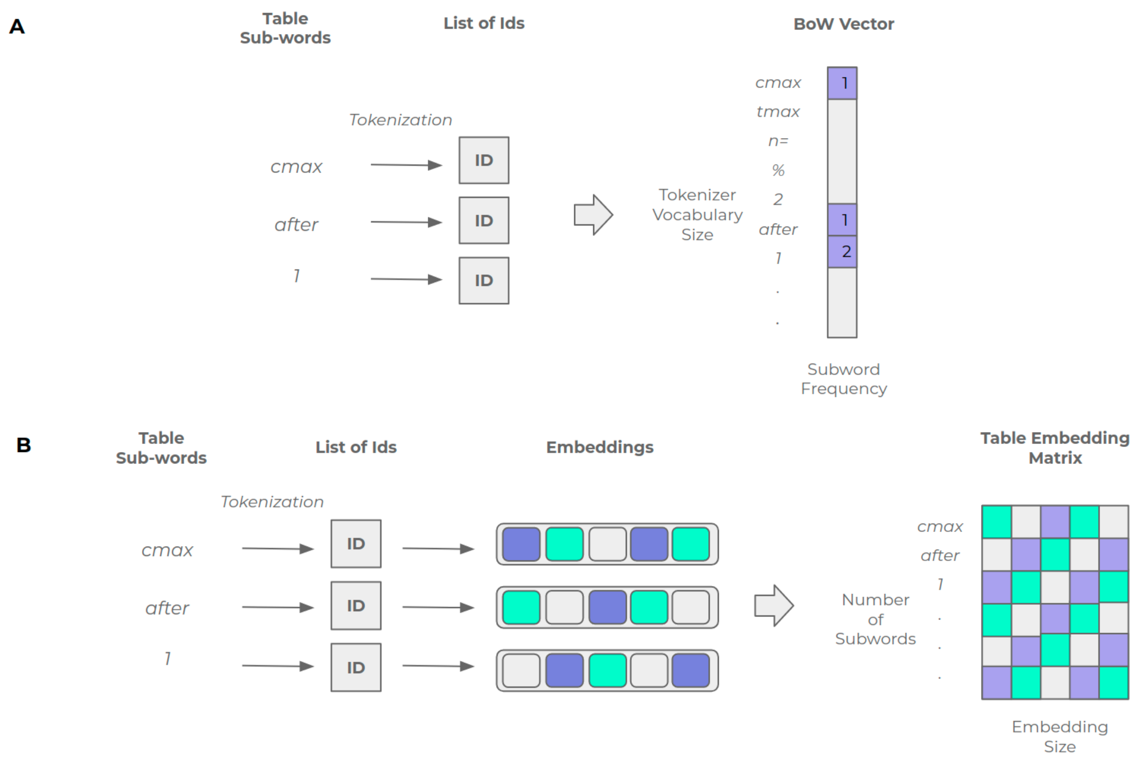


**Figure 6: Table Representation Techniques for Table Classification: (A) Table Bag of Words (B) Table Embedding Matrix**

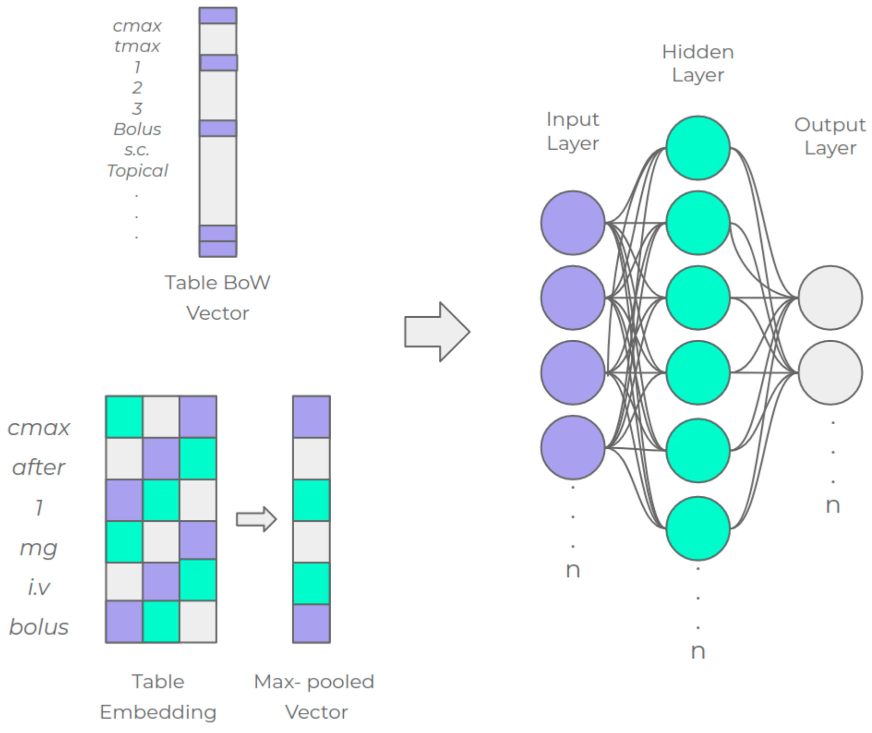
transformations were carried out and pipeline performance was compared with these in isolation or in combination. The first type of transformation was numerical and changed all float values in a table by 10% up or down and all integer values up or down by 1 with a 50:50 probability. The second type of transformation involved exchanging PK parameter and covariate terms and units with sensible synonyms. The replacement was designed so that tables would still be classified by humans with their original label set before augmentation. Under-sampling as close as possible to the minority class was also applied in combination with augmentation as compared to augmentation alone. Due to the multi-label nature of this data it was not possible to achieve complete class balance as labels do not always appear individually with enough frequency.

### 3.2.5 Table Classifiers

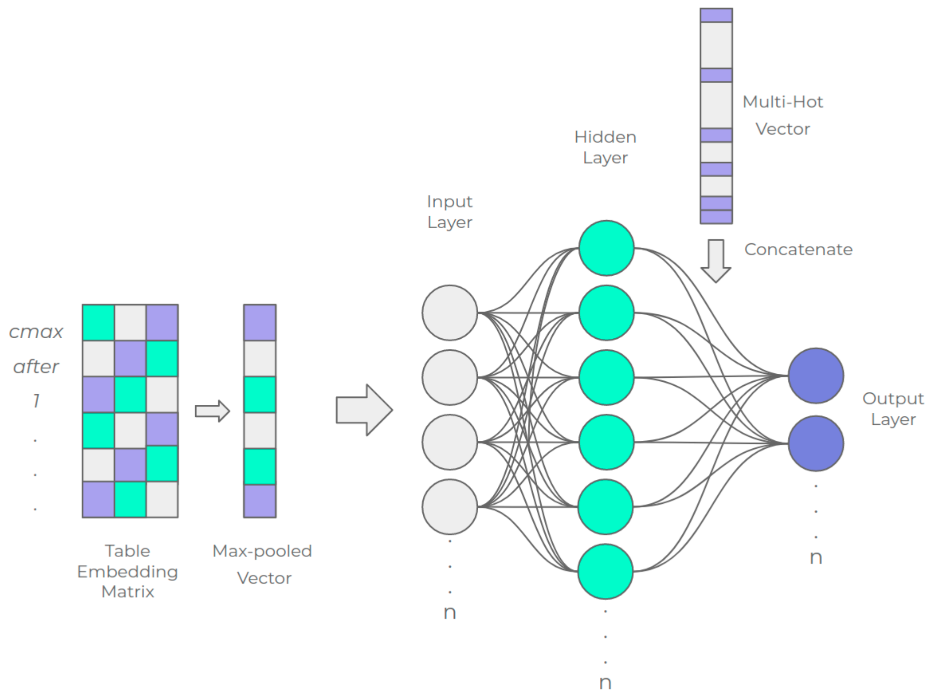
Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNN) were compared on this classification task. All classifiers were written in the python library Pytorch (code in Appendix). Initially, as a sanity check, classifiers were overfitted to a small batch of the training data to ensure that backpropagation was functioning correctly, weights were updating and the learning rate was in the correct order of magnitude. Models were trained until validation Weighted-F1 score levelled off (after which there are no significant performance increases to be gained) and later tuned as a hyperparameter.

MLPs refer to fully connected layers of neurons stacked into a feed forward network which are able to learn non-linear models. MLPs have been used widely with success for classification prediction problems and provide a good baseline against which to compare more complex neural network architectures. The first application of a MLP to this table classification task was comprised of an input layer which receives embeddings, which have been max-pooled to reduce their dimensionality, or BoW vectors, one hidden layer and an output layer with the same number of nodes as classes (Figure 7a). For each class, Binary Cross Entropy Loss is computed, thereby considering each class as a binary classification problem and summing these to calculate overall loss.[57,58] The second application was to adapt the classic MLP. A multi-hot encoded vector of the table input, with sub-word frequency counts in positions along a vector of length of the total vocabulary from the Tokenizer to represent the input table without pooling, was concatenated with the output of the hidden layer, before being passed to the output layer for classification (Figure 7b). This provides the benefit of reminding the network of any important features that may have been lost in the pooling operation.

CNNs are composed of a convolutional layer, applying a filter to the input and producing local feature maps, a pooling layer, obtaining translation invariant descriptions and a fully connected layer which outputs predicitons. CNNs have been used with success for text classification, generally outperforming traditional feature-based and kernel-based supervised methods, even with a simple architecture.[42,43] The first application of a CNN to the Table Classification task was the Multi-Channel N-GRAM CNN model, first proposed by Kim.[42] Figure 8a shows that the CNN model takes as input a table, represented by a sub-word embedding matrix. Following this, the embedding matrix is fed to the convolutional layer. Here, filters of different sizes extract features of different sizes, thus processing the document at different resolutions (different n-grams or groups



(a)



(b)

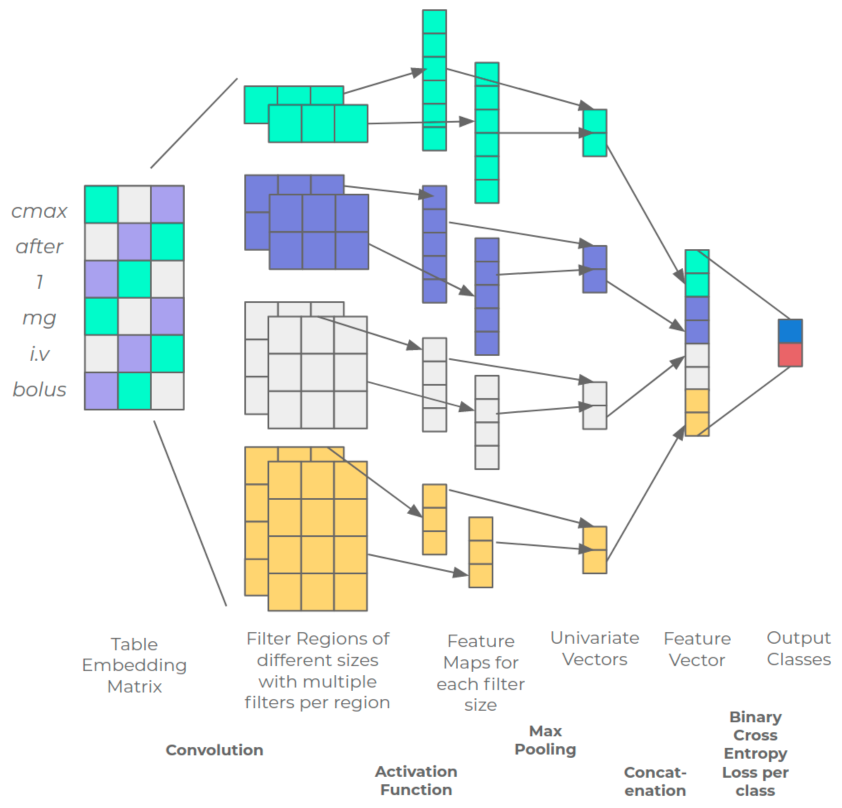
**Figure 7: MLP architectures for Table Classification: (A) Classic MLP (B) MLP with multihot vector concatenated with output of hidden layer**

of words) simultaneously. Thereafter, the pooling layer performs down-sampling on the feature maps. This is important in order to reduce the resolution and therefore the computational complexity and to pull out the most relevant and essential local features. Lastly, the feature vectors generated by pooling in the different filter channels are fed into a fully connected layer for classification. This layer has the same number of nodes as classes and for each class computes Binary Cross Entropy Loss, thereby considering each class as a binary classification problem and summing these to calculate overall loss.[57,58]The second application of a CNN to this task was to adapt the Multi-Channel N-GRAM CNN model[42] by concatenating a multi-hot encoded vector of the table input with the output feature vectors from the pooling layer before passing this to the fully connected layer for classification (Figure 8b). The purpose of this is to remind the network of any important features that may have been lost during the pooling stage.

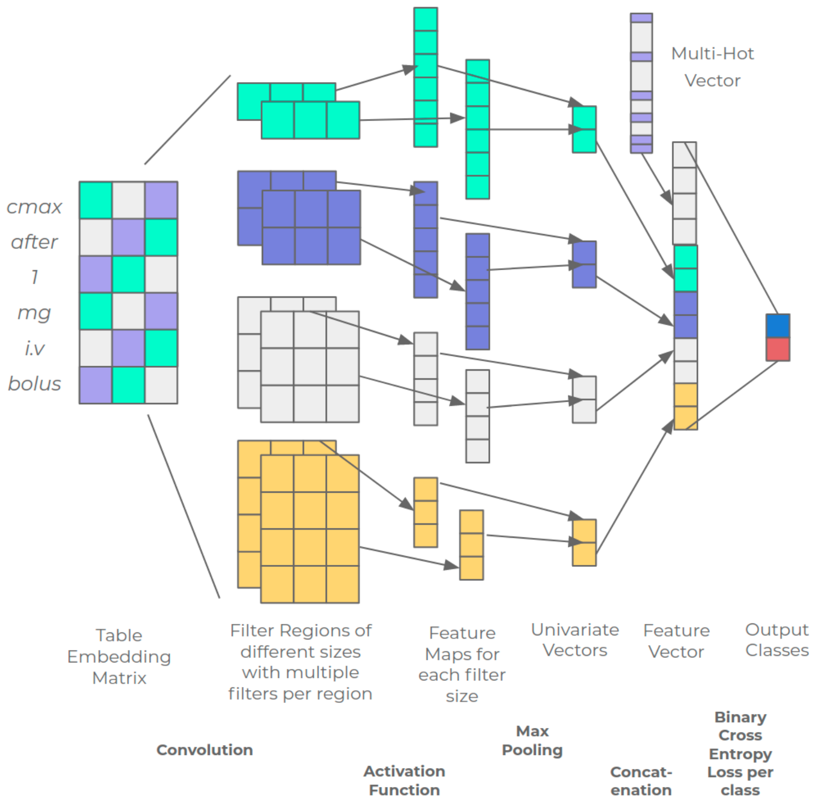
### 3.2.6 Evaluation

Pipeline development aimed to maximise classifier performance on all nine positive classes, by investigating different pipeline configurations and tuning classifier hyperparameters. Analyses were performed to assess the performance as a result of variations in table representations, input table fields, data augmentation and classifier architecture and pipeline hyperparameters. To compare pipeline configurations per class, Precision, Recall and F1-score (harmonic mean between Precision and Recall) were observed. Accuracy was not observed due to the highly class imbalanced nature of this data. In addition, Weighted, Micro and Macro F1-score across all positive classes were calculated (Table 3). Micro-F1 weights each prediction equally, Macro-F1 treats all classes equally and Weighted-F1 score weights each class F1 by the number of true instances. Consequently, Weighted-F1 gives a good indication of overall model performance for highly imbalanced multi-label tasks such as this, assigning less importance to classes with few true instances. However, balancing this with Macro-F1 score is important so as not to over-fit the model to majority classes.

The optimum table representations for classification were selected by performing analysis on: (1) Table Tokenization (2) Table Fields, and (3) Table Representations. Analysis



(a)



(b)

**Figure 8: CNN Architectures for Table Classification: (A) NGRAM CNN (B) NGRAM CNN with multihot vector concatenated with output feature vector**

**Table 3: Summary of Evaluation Metrics**

| Label | Description | Details |
|---|---|---|
| Precision | $\frac{TP}{TP+FP}$ | Ratio between the True Positives and all the Positives, calculated per class, considering occurrence and non-occurrence of the label as the minority and majority classes |
| Recall | $\frac{TP}{TP+FN}$ | Measure of the model correctly identifying True Positives, calculated per class, considering occurrence and non-occurrence of the label as the minority and majority classes |
| F1 | $2.\frac{Precision.Recall}{Precision+Recall}$ | Harmonic Mean of Precision and Recall, calculated per class, considering occurrence and non-occurrence of the label as the minority and majority classes |
| $F1_{Micro}$ | $2.\frac{Precision_{Micro}.Recall_{Micro}}{Precision_{Micro}+\ Recall_{Micro}}$ | Contribution of all classes is aggregated before metric calculations, therefore weighting all predictions equally |
| $F1_{Macro}$ | $\frac{F1_1+F1_2+...+F1_n}{n}$ | F1 is computed independently for each class and then averaged, therefore treating all classes equally |
| $F1_{Weighted}$ | $\frac{(F1_1.Support_1)+...+(F1_n.Support_n)}{\sum Support_{1-n}}$ | F1 is computed independently for each class and Weighted by support (the number of true instances for each label) and then averaged, useful in cases of class imbalance |

**Notes:** Per class/label: False Positives (FP) refers to the number of Tables where the model incorrectly predicts the positive class; and False Negatives (FN) refers to the number of Tables where the model incorrectly predicts the negative class; True Negatives (TN) refers to the number of Tables where the model correctly predicts the negative class; True Positives (TP) refers to the number of Tables where the model correctly predicts the positive class.

on (3) Data Augmentation (4) Classifier Architecture and (5) Final Pipeline Hyperparameters was also performed to assess the best performing architecture and pipeline configuration. As the data set is limited there is therefore associated variability on the final metrics, so bootstrapping was carried out to derive the distribution of F1-scores for each pipeline variation across analyses 2-4. Bootstrapping was carried out for 100 iterations, which was selected as the minimum value producing a Gaussian-like distribution of the output Weighted-F1 scores whilst considering computational constraints. At each iteration the training and Validation sets were combined and split randomly into temporary training (60%) and temporary test (40%) data sets with resampling with replacement to avoid evaluation bias. The classifier is fitted to the temporary Training set and the performance of the pipeline after each bootstrap iteration was recorded on the temporary Test set. The best performing hyperparameters were selected on the best performing pipeline from previous analyses. A 5-fold Cross Validation (CV) approach was employed on the whole Training set. 5-fold CV has the following steps: (1) data is shuffled and split randomly into 5 groups without replacement (2) one group is held out as the Test Set (3) the remaining groups are used to fit the model and it is evaluated on the held out group (4) this process is repeated until all 5 groups have been used as the Test Set (5) the mean of the performance scores calculated in the 5 iterations is taken as the final performance metric. K-fold CV approaches give a more accurate estimate of algorithm performance, like bootstrapping, by reducing effect of noise on performance estimates. However, unlike bootstrapping, CV samples without replacement with the primary purpose being to measure the generalisability of the model performance. Although grid-search is a useful tool for many ML applications, computational power grows exponentially with the number of parameters in neural networks. Consequently, validation curves (showing CV performance metrics over different hyperparameter values) were used to select the optimum hyperparameters. The hyperparameters tuned included:

1. Learning Rate: refers to the magnitude of change to the classifier weights in response to the error. A learning rate in the correct order or magnitude is essential to ensure gradient descent performs well.

2. Number of Epochs: each epoch refers to the number of times the whole Training

set is input into the classifier for training. An epoch is comprised of one or more batches. If the gap between the training and validation performance metrics is too great or too small, the current epoch is considered inappropriate as the model will be overfitting or underfitting respectively.

3. Batch (sometimes called mini-batch) Size: refers to the number of training samples used to estimate the error gradient at a time (so the number of training samples sent into a model at a time). Too large batches make it easy to fall into local optimums whereas too small batches are more 'noisy' and not representative enough.

4. Embedding Dimension: refers to the size of the dimension used to represent each token (the size of each sub-word vector). The size of these need to large enough to represent the context-dependent complexity of tokens in each table in the data set.

5. Dropout: refers to a regularization method whereby during training a certain percentage of layer outputs are randomly ignored, used to prevent a model over-fitting.

6. Number of Filters: This refers to the number of neurons in each layer of a CNN. CNNs apply a filter to an input to create a feature map and so there will be as many feature maps as filters. More filters allows CNNs to learn more complex abstractions from the data but also increase computational cost and model parameters.

7. Stride: This refers to the number of sub-words by which the filter shifts over the input table matrix (when the stride is one the filter moves one sub-word at a time). Increasing the stride allows information to be condensed but information is also lost.

Lastly, to obtain final performance metrics, the final classifier pipeline with optimised hyperparameters was utilized on the Test Set to give a measure of performance on unseen data. A qualitative assessment is also performed on 100 of the final outputted predictions compared to labels to observe examples where the model is going wrong and inform future improvements to the corpus and pipeline.

# 4 Results

## 4.1 Annotator Agreement and Label Statistics

For the Test Set, the initial pairwise Cohen's Kappa Scores (K) can be seen per label in Table 4. Disagreements were observed and in general these were due to: (i) positive instances for a label being missed by the annotator (ii) low frequency of categories in the data set such as 'Covariates Other' meaning annotators were less familiar with all scenarios and there was higher uncertainty around edge cases. Disagreements of type (i) were resolved through having multiple annotators per sample and review sessions. Errors of type (ii) were in part resolved by refinement of annotation guidelines and review sessions, however due to the small number of examples and broad acceptance criteria for some labels, some future cases still remained ambiguous. Pairwise K of over 0.75 was reached for all classes making up more than 10% of the data set, but broad classes which occurred infrequently (less than 10%, see Table 6) were more often missed when they did occur or were more often ambiguous due to the broad nature of the category (such as in the case of 'Covariates Other'), leading to low agreement scores. Agreements were reached on all disputes in post-labelling review sessions with the annotation team, in order to produce the final labelled data set. Following agreement of the final labels, F1-score was calculated on the labels from each annotator for each class compared to the final agreed labels, to allow direct comparison with model results. Mean F1-scores across annotators for each class mirrored the pattern of the pairwise K, with higher F1-scores on more prevalent classes, in particular 'Non-compartmental Parameters', 'Doses', 'Number of Subjects', 'Demographics' and 'Not Relevant' classes.

Tables were labelled by annotators in a multi-label fashion whereby a table can be assigned all labels that apply to it. Table 5 shows the maximum number of labels assigned to any one table was five and the majority of tables were assigned three or less labels or no labels (effectively being not relevant). There was no significant differences in multi-labelled instances between the data sets (p=0.05, Appendix Figure 16). On the final agreed labelled data sets, the class frequency distribution showed the greatest pro-

**Table 4: Annotator Agreement Metrics**

| Class Label | Cohen's Kappa Score | Mean F1-Score (Range) |
|---|---|---|
| Non-Compartmental Parameters | 0.86±0.06 | 0.92 (0.86-1) ±0.04 |
| Compartmental Parameters | 0.64±0.16 | 0.62 (0.49-1) ±0.20 |
| Parameter-Covariate Relationships | 0.23±0.18 | 0.46 (0.29-0.91) ±0.22 |
| Parameters Other | 0.49±0.13 | 0.68 (0.5-0.86) ±0.11 |
| Doses | 0.78±0.08 | 0.92 (0.85-0.96) ±0.03 |
| Number of Subjects | 0.83±0.01 | 0.91 (0.7-0.96) ±0.07 |
| Sample Timings | 0.51±0.13 | 0.65 (0.18-0.89) ±0.24 |
| Demographics | 0.56±0.19 | 0.77 (0.62-1) ±0.14 |
| Covariates Other | 0.22±0.08 | 0.47 (0.31-0.72) ±0.16 |
| Not Relevant | 0.75±0.04 | 0.73 (0.51-0.87) ±0.09 |
| **Mean** | 0.60±0.23 | 0.74±0.16 |

**Notes:** Cohen's Kappa Score, calculated pairwise between annotators and per label on the Test Set. Labels are grouped as parameters ('Non-compartmental parameters'- 'Parameters Other') and covariates ('Doses'-'Covariates Other'). Tables without labels are considered as 'Not Relevant' and agreement metrics were also calculated for this. F1-Scores, calculated for each annotator compared to the final agreed labels after review, mean and range provided, calculated per label on the Test Data set. All metrics reported with Standard Deviations and to 2 Decimal Places.

**Table 5: Frequency of Unlabelled, Labelled and Multi-labelled instances in Training, Validation and Test Sets**

| Number of Labels per Table | Frequency | | |
|---|---|---|---|
| | Training Set (%) | Validation Set (%) | Test Set (%) |
| 0 | 325 (22) | 61 (18) | 127 (16) |
| 1 | 394 (26) | 104 (31) | 259 (33) |
| 2 | 474 (32) | 94 (28) | 220 (28) |
| 3 | 215 (14) | 55 (16) | 118 (15) |
| 4 | 75 (5) | 24 (7) | 51 (7) |
| 5 | 15 (1) | 3 (1) | 8 (1) |
| **Total** | **1498** | **341** | **783** |

**Notes:** Please note tables without any labels are equivalent to being "not relevant". A Chi-squared test carried out on this as a two-way table (with 10 degrees of freedom and p=0.05) revealed no significant association between data set and number of labels per instances (See Appendix Figure 16).

**Table 6: Class Label Frequencies in Training, Validation and Test Data Sets**

| **Class Label** | **Class Label Frequency** | | |
|---|---|---|---|
| | Training Set (%) | Validation Set (%) | Test Set (%) |
| Non-Compartmental Parameters | 404 (15) | 91 (14) | 185 (14) |
| Compartmental Parameters | 60 (2) | 15 (2) | 30 (2) |
| Parameter-Covariate Relationships | 59 (2) | 15 (2) | 23 (2) |
| Parameters Other | 80 (3) | 21 (3) | 29 (2) |
| Doses | 481 (18) | 117 (19) | 242 (19) |
| Number of Subjects | 687(26) | 163 (26) | 322 (25) |
| Demographics | 389 (14) | 104 (17) | 228 (18) |
| Covariates Other | 93 (3) | 19 (3) | 49 (4) |
| Sample Timings | 109 (4) | 23 (3) | 65 (5) |
| Not Relevant | 325 (12) | 61 (10) | 127 (10) |
| **Total** | **2687** | **629** | **1300** |

**Notes:** Please Note Class Frequencies sum to greater than the number of samples in each data set due to the multi-label nature of this data. Labels are grouped as parameters ('Non-compartmental parameters'- 'Parameters Other') and covariates ('Doses'-'Covariates Other'). Tables without labels are considered as 'Not Relevant' and their frequency is also noted here. A Chi-squared test carried out on this as a two-way table (with 16 degrees of freedom and p=0.05) revealed no significant association between class frequencies and data set (See Appendix Figure 16).

portion of tables were labelled with 'Number of Subjects', closely followed by 'Doses', 'Non-compartmental Parameters' and 'Demographics' classes which all had a frequency of more than 300 in the Training set (Table 6). Additionally, the not relevant class (the equivalent of a table having no labels assigned) also had more than 300 examples in the Training set. On the other hand, 'Compartmental Parameters', 'Parameter-Covariate Relationships', 'Parameters Other', 'Sample Timings' and 'Covariates Other' had a low frequency with less than 100 examples for each class. There was no significant difference in class frequency distribution between the training, test and Validation sets (p=0.05, Appendix Figure 16).

## 4.2 Tokenization Strategies

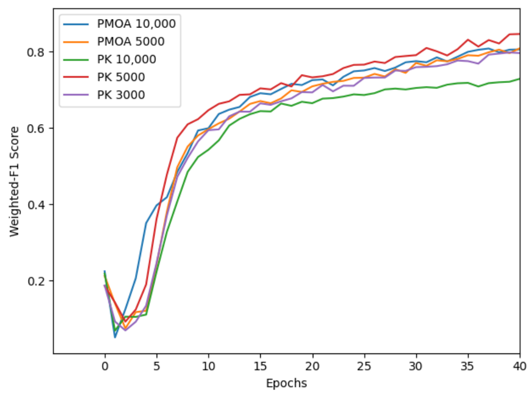
Tokenizers were trained on either all tables from PMOA or the PK subset only, both with Test set tables removed. Although output tokens were observed on a number of example tables it is difficult to get an idea of tokenization performance with qualitative observation alone. Consequently, to give an indication of which Tokenizer was optimal, each Tokenizer was tested as part of a machine learning pipeline using a basic MLP, with embeddings as table representations. Summary evaluation metrics were computed on the Validation Set for these different Tokenizers trained with different vocabulary sizes (Table 17). The Weighted-F1 scores and loss calculated on the Validation Set across training epochs is also shown in Figure 9d and 9c respectively. Loss decreases until around 15 epochs after which point it begins to rise with all Tokenizer variations. This coincides with Weighted-F1 score levelling off and only increasing very slightly beyond this point. It was observed that using a vocabulary size of 10,000 gives the lowest validation Weighted-F1 scores across training epochs and lowest final Weighted-F1 (PMOA= 0.63, PK =0.48) and Micro-F1 (PMOA= 0.63, PK =0.48) for both PMOA and PK data sets. Conversely however, for PK tables alone, the highest Macro-F1 score is achieved using 10,000 (0.64). Due to the smaller total vocabulary expected in PK tables alone, a vocabulary size of 3000 was also used to train the PK Tokenizer. The performance is lower across the board compared to 5000 vocabulary size. Evaluation Metrics calculated for each class are available in Table 17 in the Appendix. Classes with a lower support achieve a lower F1 score with all Tokenizer variations. "Covariates other" achieves a particularly low Precison, Recall and F1 score across Tokenizers, in some cases being zero (meaning there were no true positives). It is noteworthy that in the case of all classes with low support ("Compartmental Parameters", "Parameter-Covariate Relationships", "Parameters Other" and "Sample Timings"), Precision is high and Recall is low, leading to an overall lower F1-score. Overall, the Tokenizer trained on PK tables with 5000 vocabulary gives the highest Weighted-F1 (0.75) and Micro-F1 score (0.77) on the Validation Set and second highest Macro-F1 (0.57) and is therefore considered to be the best compromise in terms of being generalisable to new examples and coping with minority class tokens. Based on this, the PK-5000 Tokenizer is applied in all subsequent analyses.

## 4.3 Preprocessing Strategies

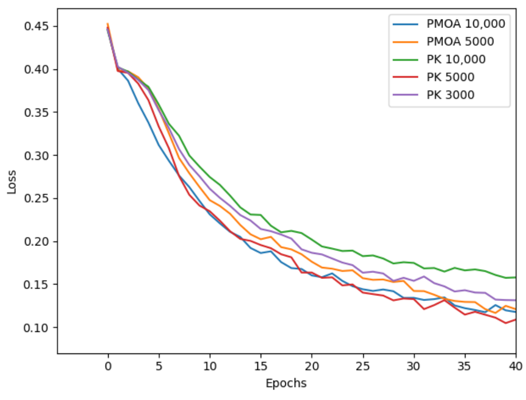
**Table 7: Summary Performance Metrics for Tokenizer Variations**

| Training Set | Vocab Size | Micro-F1 | Macro-F1 | Weighted-F1 |
|---|---|---|---|---|
| PMOA-tables | 10,000 | 0.68 | 0.39 | 0.63 |
| | 5,000 | 0.69 | 0.46 | 0.66 |
| PK-tables | 10,000 | 0.41 | **0.64** | 0.48 |
| | 5,000 | **0.77** | 0.57 | **0.75** |
| | 3,000 | 0.73 | 0.51 | 0.70 |

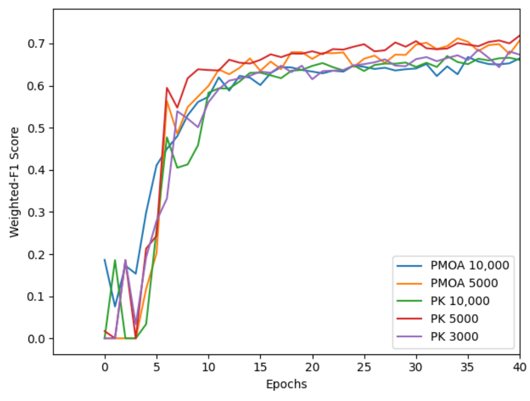
**Notes:** PMOA= Pubmed Open Access Papers. PK= Pharmacokinetic relevant subset of PubMed Open Access Papers. Highest results for each metric are highlighted in bold. Micro-, Macro- and Weighted-F1 Scores all taken over all positive classes (excluding those without labels/not relevant). All analyses performed using an MLP pipeline with embeddings as table representations.



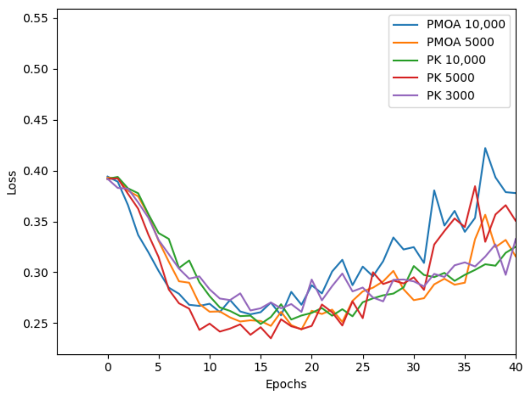
(a) Train Weighted F1



(b) Train Loss



(c) Validation Weighted F1

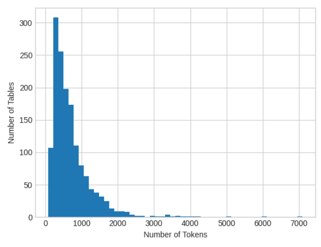


(d) Validation Loss

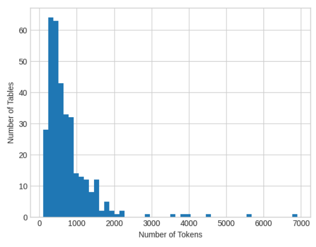
**Figure 9: Loss and Weighted F1-Score on the Training and Validation sets over Training Epochs for different Tokenizer Variants**

Distribution of the number of tokens per table, following tokenization with the optimal Tokenizer, can be observed in Figure 10. With minimal preprocessing, only removing style but keeping other HTML tags pertaining to table structure, maximum tokens on Training set tables are as high as 7000. However, the majority of tokens lie below 2000 tokens with the highest frequency being around 500 tokens (Figure 10a). This pattern is consistent on the validation(Figure 10b) and Test sets (Figure 10c). After removing all HTML tags, the overall token frequency decreases with the maximum number of tokens on the Training set being less than 3000, the majority of tables falling below 1000 tokens and the maximum frequency occurring at 200 tokens (Figure 10d). Again, the validation (Figure 10e) and Test set (Figure 10f) show a similar shift towards a lower maximum token frequency. With preprocessing to remove all HTML tags and only keep text from the caption, header row and first column of the table, the total tokens decrease further with the maximum tokens per table being less than 1000, the majority of tables having less than 300 tokens and the highest token frequency falling at around 100 tokens on the training, validation and Test sets (Figures 10g-10i).

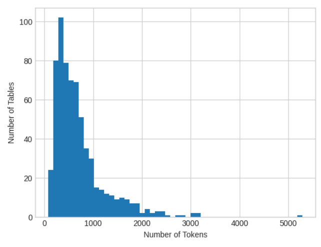
To find the optimal table preprocessing strategy, each strategy was tested as part of a machine learning pipeline using a basic MLP with the optimal PK-5000 Tokenizer and embeddings as table representations. The token distributions above (Figure 10) were used to select a sensible range of maximum lengths to use with each preprocessing strategy. Summary metrics calculated on the Validation Set following different table preprocessing strategies are displayed in Table 8. Additionally, Weighted-F1 scores and Loss calculated on the Validation Set across training epochs are displayed in Figure 11d and Figure 11c respectively. Across training epochs, validation F1 is consistently higher for removing HTML compared with other preprocessing methods after 5 epochs. Validation Loss decreases up to 15 epochs and then rises except for baseline preprocessing which rises after around 10 epochs, which coincides with levelling off of Weighted-F1 scores. It was observed that stripping away all HTML tags and leaving only the baseline (caption, header row and first column), dents performance considerably across all summary metrics (Figure 10). Using HTML tags and just removing style does enhance performance over the baseline, but has especially low Macro-F1 scores across all maximum token lengths (0.44-0.48). Removing HTML tags completely gives the best performance across different



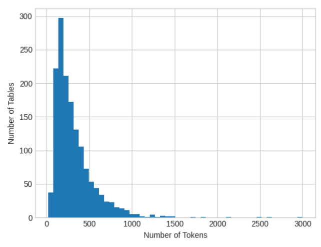
(a) Train- Style Removed



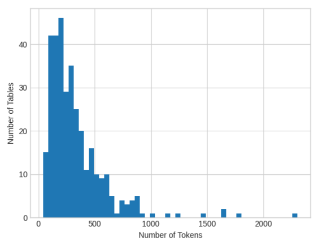
(b) Validation- Style Removed



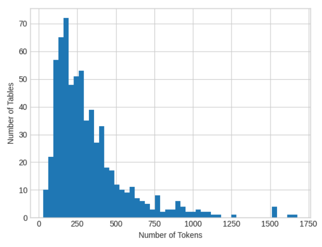
(c) Test- Style Removed



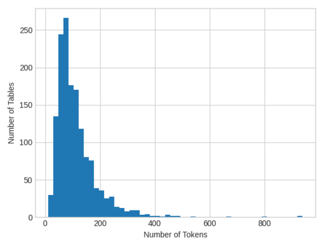
(d) Train- HTML Removed



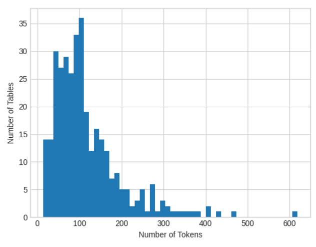
(e) Validation- HTML Removed



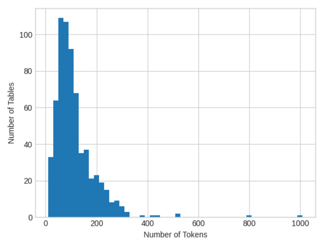
(f) Test- HTML Removed



(g) Train- Caption and Header row/col



(h) Validation- Caption and Header row/col



(i) Test- Caption and Header row/col

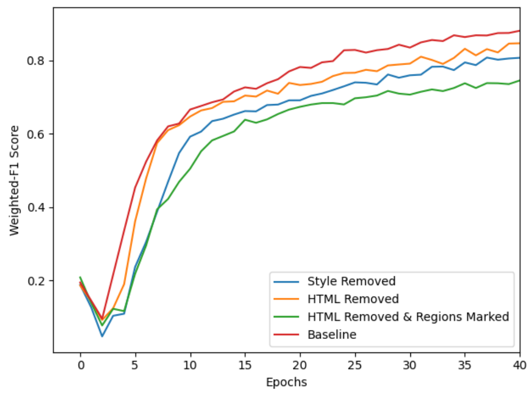
**Figure 10: Distribution of Table Token Lengths in the Training, Validation and Test Sets with different Preprocessing Strategies**

maximum token lengths. Maximum token length does not affect performance dramatically within each preprocessing strategy, however a maximum length of 500 rises above the rest when you remove HTML. There is a particularly marked increase in Macro-F1 score with this being 0.06 above the closest neighbour (HTML removed with 1000 maximum tokens) within the HTML removed category. However, the Macro-F1 is still the least high performing summary metric at 0.57 compared to Weighted- and Macro- F1 scores of 0.75 and 0.77 respectively. Taking the optimum model and marking the specific tables regions (caption, header row, first column and body) with special tokens does not offer any improvement and in fact lowers performance over removing HTML tags alone. The class breakdown of evaluation metrics are available in Table 18 in the Appendix. The same pattern is seen as in the Tokenizer results, where classes with low support have lower F1-scores and often have high Precision and low Recall. Overall, removing HTML tags is considered to be the optimum preprocessing method for table classification and 500 tokens is considered the optimum within this. Based on this, subsequent analyses will remove HTML table tags during preprocessing and use a maximum token length of 500.

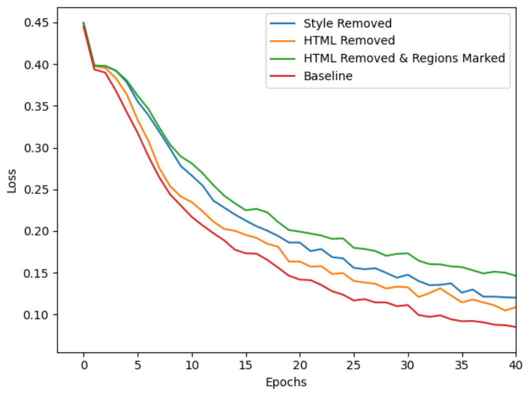
**Table 8: Summary Performance Metrics with Various Preprocessing Strategies**

| Regions Included | Max. Tokens | Micro-F1 | Macro-F1 | Weighted-F1 |
|---|---|---|---|---|
| Style Removed | 2000 | 0.74 | 0.44 | 0.69 |
| | 1000 | 0.72 | 0.44 | 0.68 |
| | 500 | 0.74 | 0.48 | 0.69 |
| HTML Removed | 1000 | 0.75 | 0.51 | 0.72 |
| | 500 | **0.77** | **0.57** | **0.75** |
| | 300 | 0.75 | 0.50 | 0.71 |
| | 200 | 0.75 | 0.49 | 0.71 |
| | 100 | 0.74 | 0.47 | 0.70 |
| HTML Removed &Regions Marked | 500 | 0.75 | 0.52 | 0.71 |
| Baseline* | 300 | 0.69 | 0.48 | 0.66 |
| | 100 | 0.69 | 0.47 | 0.69 |

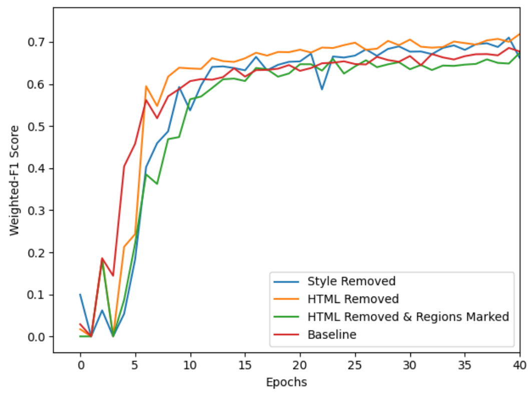
**Notes:** Baseline refers to only the Caption, Header Row and First column, with HTML tags removed. All analyses performed using PK-5000 Tokenizer and an MLP pipeline with embeddings as table representations. Highest results for each metric are highlighted in bold. Micro-, Macro- and Weighted-F1 Scores taken over all positive classes (excluding those without labels/not relevant)



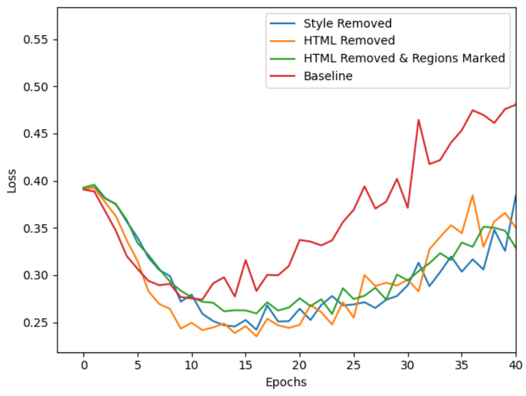
(a) Train Weighted-F1



(b) Train Loss



(c) Validation Weighted-F1



(d) Validation Loss

**Figure 11: Loss and Weighted F1-Score on Training and Validation Sets over Training Epochs for Different Preprocessing Strategies. Weighted-F1 Score calculated over all positive classes (excluding those without labels/not relevant)**

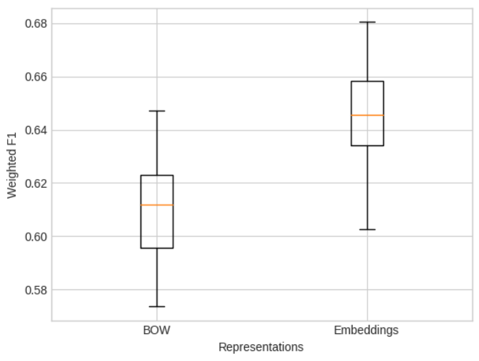
## 4.4 Table Representations

To test whether embeddings offer an advantage over the simpler BOW method for this classification task, representation methods were compared using an MLP Pipeline. Summary evaluation metrics calculated following 100 bootstrap iterations on the Training set are summarised in Table 9 and and Figure 12. In general, embeddings and BOW have overlapping confidence intervals across all summary metrics. However, embeddings have a higher median across the board compared to BOW and this is particularly notable for Micro- and Weighted-F1 scores. Figure 12 shows the distribution of Weighted-F1 scores is shifted higher for embeddings with a much greater median. For the Macro-F1 distribution, although the median is only a little higher, there is much less variance indicating more consistent performance on minority classes. F1 by class is available in Table 15 in the Appendix. Neither BOW nor embeddings perform well on minority classes but embeddings outperform BOW on majority classes and on the minority class 'Compartmental Parameters'. Given these results, embeddings were selected as the preferred method to represent tables and are used in all analyses going forward.

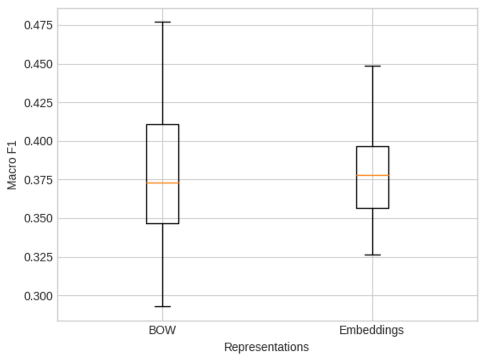
**Table 9: Table of Summary Performance Metrics after 100 Bootstrap Iterations, reported as Micro-F1 ($F1_{Micro}$), Macro-F1 ($F1_{Macro}$) and Weighted-F1 ($F1_w$) medians (with 95% CIs) and $F1_w$ interquartile variance (IQV)**

| Representation | $F1_{Micro}$ (95% CIs) | $F1_{Macro}$ (95% CIs) | $F1_w$ (95% CIs)) | $F1_w$ IQV |
|---|---|---|---|---|
| BOW | 66.0(64.0,67.8) | 37.3(31.6,45.3) | 61.2(57.7,64.3) | 6.6 |
| Table Embeddings | **70.0(67.4,72.8)** | **37.8(33.2,44.1)** | **65.4(62.6, 68.9)** | **6.3** |

**Notes:** All Analyses were performed with HTML tags removed during preprocessing, the PK-5000 Tokenizer and 500 maximum token length and using an MLP pipeline. Highest results for each metric are highlighted in bold. Micro-, Macro- and Weighted-F1 Scores all calculated over all positive classes (excluding those without labels/not relevant)



(a) Weighted F1



(b) Macro-F1

**Figure 12: Distribution of Weighted-F1 scores for different table representation strategies following 100 bootstrap iterations**

## 4.5 Model Architectures and Class Imbalance

To evaluate the best model architecture and optimum methods employed to deal with class imbalance, 100 bootstrapping iterations on the Training set were used to calculate evaluation metrics. Table 10 and Figure 13 show the results of this analysis. In addition, in the Appendix, a breakdown by class is available in Table 19 and histograms of the results from each 100 bootstrapping iterations in Figure 17. In general, the CNN architecture was superior to the MLP architecture in terms of Weighted-F1, which can be visualised in Figure 13a and CNNs also achieved a narrower Weighted-F1 IQV and Micro-F1 scores (Table 10). However, the pattern is not so clear for Macro-F1, where CNNs generally have lower variance but their medians are very similar to MLPs suggesting they do not offer much advantage on minority classes (Figure 13b). Addition of a Multihot vector to the CNN architecture did not improve performance but interestingly for the MLP it increased Macro-F1 score.

Multi-label under-sampling for the MLP decreased performance on majority classes (Weighted-F1) (Figure 13a) and overall performance (Micro-F1 score) and increased Weighted-F1 IQV (Table 10) but also increased performance on minority classes (Macro-F1-score) (Figure 13b). In the case of the CNN, multi-label undersampling also improved Macro-F1 score (Figure 13b) and slightly improved the median Weighted-F1 Score although confidence intervals overlap with the CNN alone and there is a greater Weighted-F1

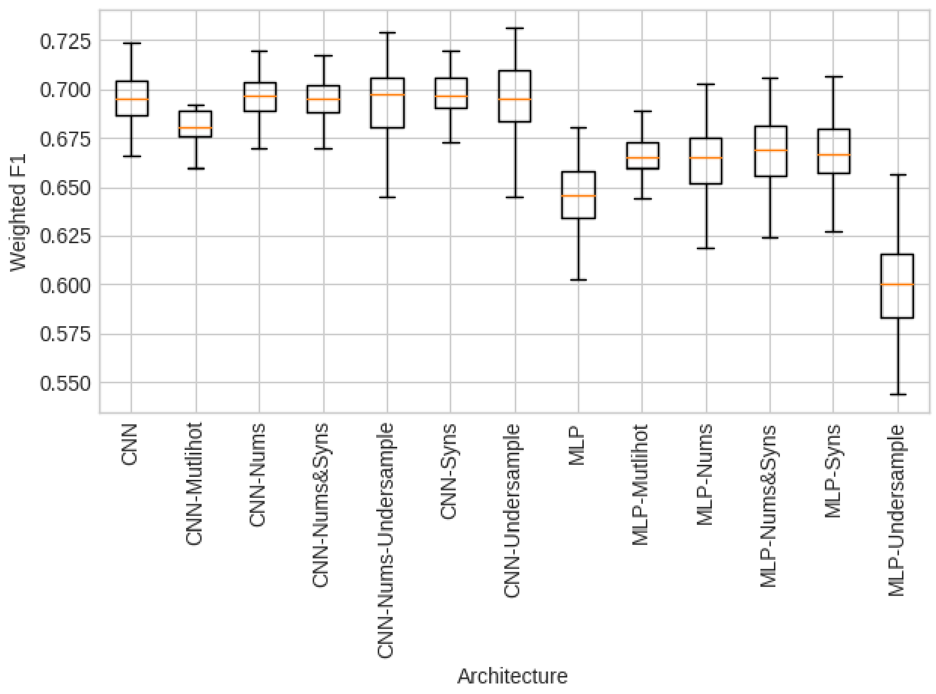
IQV. In the per class breakdown performance on minority classes is improved somewhat by umdersampling however classes such as “Covariates Other” and“Parameters Other” still achieve an F1-score of less than 25% and minority classes having a much greater Weighted-F1 IQV.

Augmenting synonyms does not improve summary metrics compared to the CNN alone (Table 10), and does not produce the boost in Macro-F1 seen with undersampling (Figure 13b), however it does reduce variance in both Weighted- and Macro-F1 scores (Figure 13). In addition, the per class breakdown reveals minority classes are impacted negatively by augmenting synonyms (Table 19). Augmenting numbers in the case of MLP increases the median Weighted-, Macro- and Micro F1 scores compared with the MLP alone and but Macro-F1 is not as high as in the case of undersampling (Figure 13b). However, in the case of the CNN, augmenting numbers also increases the median Micro- and Weighted- F1 scores, however Macro-F1 remains as low as with the CNN alone (Table 19). Looking at the class breakdown for the CNN, minority classes also have a lower F1 score than with resampling alone. Augmenting synonyms and numbers of the same table simultaneously gives only a modest increase in median F1-score in the case of the MLP. Augmenting both numbers and synonyms for the CNN does not do so well on Micro- or Weighted- F1 scores alone and has a very similar Macro-f1 score as augmenting numbers alone. In addition, augmenting both negatively impacts median F1-score on minority classes for the CNN (Table 19). As the CNN was the best architecture and augmenting numbers was the best augmentation method for the CNN and undersampling had the best affect on Macro-F1, combining these two techniques was also tested for the CNN. Augmenting numbers and undersampling gave the highest overall median Macro-f1 score (Figure 13b) of all the methods without compromising too much on overall Micro-F1 (Table 10). Weighted-F1 is lower than with augmenting numbers alone but slightly higher than with undersampling alone (Figure 13a). Looking at the per class F1 scores, similar to undersampling alone, F1 score is improved notably on minority classes and more so than with augmenting numbers alone (Table 19). In addition, F1 scores for majority classes remain high (Table 19). For the CNN, although results from undersampling alone are very similar across the board to that of augmenting numbers with undersampling, intuitively incorporating augmentation should produce a more robust model that is less liable to overfitting. Consequently, to achieve the best compromise across minority and majority classes and get the best overall performance whilst also generating a robust model, the CNN with augmentation of numbers and undersampling will be taken forward to the following analyses as the final best model.

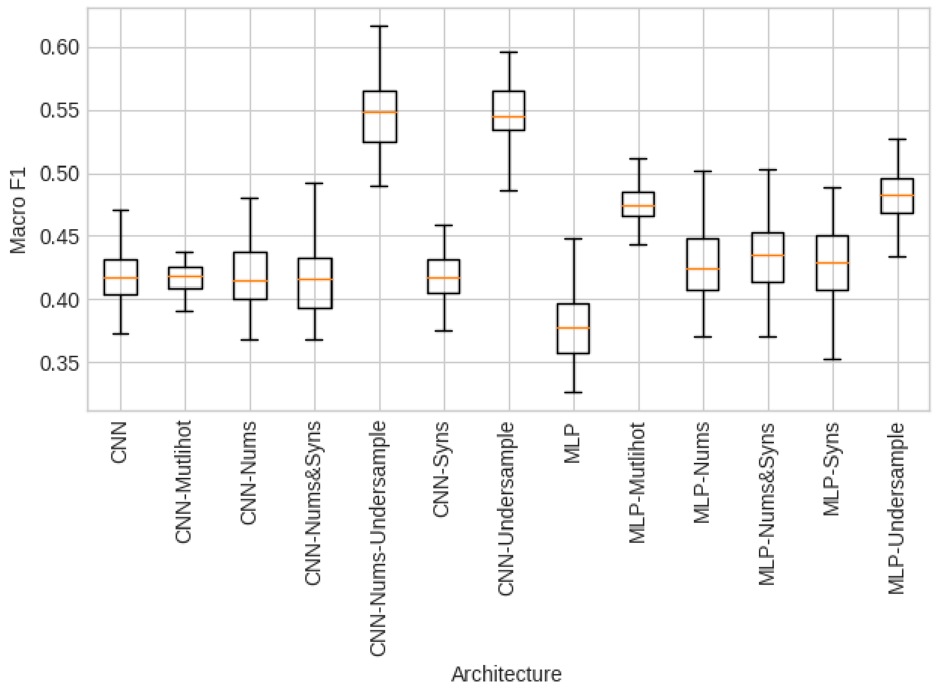
**Table 10: Table of Summary Performance Metrics after 100 Bootstrap Iterations, reported as Micro-F1 ($F1_{Micro}$), Macro-F1 ($F1_{Macro}$) and Weighted F1 ($F1_w$) medians (with 95% CIs) and $F1_w$ interquartile variance (IQV)**

| **Architecture** | **Augmentation** | **Undersampling** | $F1_{Micro}$ (95% CIs) | $F1_{Macro}$ (95% CIs) | $F1_w$ (95% CIs) | $F1_w$ IQV |
|---|---|---|---|---|---|---|
| MLP | None | No | 70.0(67.4,72.8) | 37.8(33.2,44.1) | 65.4 (62.6, 68.9) | 6.3 |
| | None | Yes | 58.5(52.6,62.5) | 48.2(43.0, 48.2) | 61.2(57.3, 65.4) | 8.1 |
| | Synonyms | No | 71.4(68.2,74.0) | 42.9(37.9,47.8) | 66.5 (63.0, 69.1) | 6.1 |
| | Numbers | No | 71.4(68.2,74.0) | 42.9(37.8,47.9) | 68.4 (65.0, 71.6) | 6.6 |
| | Synonyms & Numbers | No | 71.5(67.9,74.6) | 43.5(37.8,48.4) | 66.7(63.1, 69.5) | 6.4 |
| MLP+ Multihot Vector | None | No | 69.6(68.3,71.2) | 47.4(44.7,50.7) | 66.5(64.8,68.4) | 3.6 |
| CNN | None | No | 75.0(72.9,77.1) | 41.7(37.8,41.6) | 68.7 (66.3, 70.6) | 4.3 |
| | None | Yes | 72.0(67.3,74.6) | 54.4(49.0,58.4) | 69.4(65.4, 72.8) | 7.4 |
| | Synonyms | No | 75.1(73.0,76.8) | 41.6(38.1,45.6) | 69.8 (67.6, 71.8) | 4.2 |
| | Numbers | No | **75.1(73.2,76.9)** | 41.4(38.1,46.3) | **73.5 (70.8, 75.8)** | 5.0 |
| | Numbers | Yes | 71.9(68.2,74.3) | **54.8(49.9,59.7)** | 69.7(65.8,73.0) | 7.3 |
| | Synonyms & Numbers | No | 74.8(73.3,76.7) | 41.6(37.2,46.6) | 69.5 (67.5, 71.7) | 4.2 |
| CNN+ Multihot Vector | None | No | 73.0(71.2,74.0) | 41.8(39.2,43.6) | 68.1(66.2,69.2) | 3.0 |

**Notes:** For all analyses HTML tags were removed during preprocessing and the PK-5000 Tokenizer and embeddings were used in the Pipeline. Highest results for each metric are highlighted in bold. Micro-, Macro- and Weighted-F1 Scores all taken over all positive classes (excluding those without labels/not relevant).



(a) Weighted F1



(b) Macro-F1

**Figure 13: Distribution of Weighted-F1 scores for the different architectures with various class imbalance methods following 100 bootstrap iterations**

## 4.6 Hyperparameter Tuning

To select hyperparameters a five-fold cross validation (CV) approach was applied to the Training set using the final best model with the PK-5000 Tokenizer and embeddings. The hyperparameter values tested for each category can be observed in Table 11 and the optimum values chosen for these are also displayed. Optimal values were chosen from the range tested by those which gave the best Weighted-f1, Macro-f1 and Micro-f1 scores on 5-fold CV (Table 11). Validation curves displaying the training Weighted-F1 score and 5-fold CV Weighted-F1 score for each hyperparameter across the values tested is displayed in Figure 14. Validation curves displaying the training Macro-F1 score and CCV Macro-F1 score for each hyperparameter can be seen in Figure 15.

For learning rate values tested, learning rates of 0.01 and 0.005 gave the best performance across the board on summary metrics, but 0.01 gave a marginally better Macro-F1 score but this observation is not significant (standard deviations overlap) (Table 11). Learning rates tested above or below these values results in a significant decrease on all summary performance metrics (Figure 14a, 15a). Therefore, either 0.01 or 0.005 could be chosen as the learning rate based on these results but 0.01 is selected as the final value.

Batch size changes had a much smaller affect on Weighted (Figure 14b) and Macro-F1 scores (Figure 15b) and Micro-F1 scores than did learning rate changes. A batch size of 32 gave the best Macro-F1 score overall the standard deviations overlap with batch sizes of 16 and 64 and their Weighted and Micro-f1s are equal. Increasing the batch size to 128 produces a significant decline in Macro and Weighted-F1 scores (Table 11). Intuitively, 16, 32 or 64 could be chosen as the final batch size based on these results. 32 was chosen as the final value to optimise computationally whilst keeping batches as large as possible.

The number of epochs the model is trained for has a significant effect on Weighted (Figure 14c) and Macro F1-scores (Figure 15c). As training epochs increase so all summary metrics increase up until 15 epochs were they begin to level off and there is no significant difference seen between Macro-F1 scores of 15 and 20 epochs although Micro and Weighted-F1 scores still see a small significant increase (Table 11). To avoid over-fitting on the majority classes, 15 is selected as the final number of epochs for training as Macro-

F1 does not increase after this.

Increasing Embedding Size up until 50 causes a sharp increase in Weighted (Figure 14d) and Macro-F1 (Figure 15d) scores but beyond this point there are no significant changes in Micro-, Macro- or Weighted-F1 scores (Table 11). To reduce parameters in the model, 50 is selected as the minimum embedding size giving the best results.

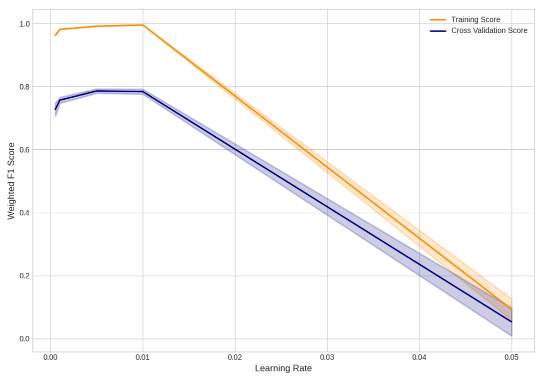
Adding dropout of 0.1 has a small significant increase across the board on summary metrics, however further increasing fro 0.2 to 0.5 slightly decreases performance on Micro- and Weighted-F1 and does not significantly increase Macro-F1 compared to 0.1 dropout (Figure 14e, 15e). High dropout at 0.8 for dropout significantly decreases performance on all summary metrics (Table 11). Consequently 0.1 is chosen as the optimum value.

Increasing the number of filters improves Weighted-F1 score (Figure 14f) and Macro-F1 (Figure 15f) scores up to 100, after which not significant increase in summary metrics is seen (Table 11). 100 is therefore selected as the optimised value for number of filters. Increasing stride value leads to a sharp decrease in performance on both Weighted- (Figure 14g) and Macro-F1 (Figure 15g) scores which is significant (Table 11). Therefore the minimum stride possible which is 1 is chosen as the final optimised value.

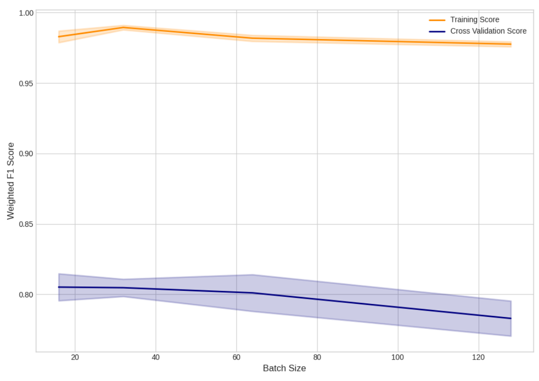
**Table 11: Summary Performance Metrics from 5-fold Cross-validation with different Hyperparameter values**

| **Hyperparameter** | **Value** | $F1_{Micro}$ | $F1_{Macro}$ | $F1_w$ |
|---|---|---|---|---|
| Learning rate | 0.05 | 0.12±0.16 | 0.03±0.05 | 0.06±0.10 |
| | **0.01** | 0.82±0.01 | 0.67±0.02 | 0.80±0.02 |
| | 0.005 | 0.82±0.01 | 0.66±0.05 | 0.80±0.01 |
| | 0.001 | 0.79±0.02 | 0.53±0.02 | 0.75±0.01 |
| | 0.0005 | 0.75±0.01 | 0.41±0.02 | 0.69±0.01 |
| Batch Size | 16 | 0.82±0.01 | 0.67±0.02 | 0.81±0.01 |
| | **32** | 0.82±0.01 | 0.68±0.01 | 0.81±0.01 |
| | 64 | 0.82±0.01 | 0.66±0.02 | 0.80±0.01 |
| | 128 | 0.81±0.01 | 0.62±0.02 | 0.78±0.01 |
| Epochs | 2 | 40.0±0.04 | 19.9±0.03 | 33.6±0.03 |
| | 5 | 66.3±0.03 | 51.9±0.04 | 63.7±0.04 |
| | 10 | 73.8±0.01 | 61.7(0.01) | 72.6±0.01 |
| | 15 | 75.4±0.02 | 64.5±0.03 | 74.4±0.02 |
| | **20** | 75.9±0.02 | 64.4±0.02 | 75.3±0.02 |
| Embed Size | 10 | 69.4±0.03 | 58.4±0.02 | 70.0±0.02 |
| | **50** | 76.9±0.03 | 68.1±0.03 | 77.4±0.02 |
| | 100 | 77.1±0.01 | 68.1±0.02 | 77.1±0.01 |
| | 200 | 77.1±0.02 | 68.3±0.02 | 77.9±0.01 |
| Dropout | None | 76.0±0.02 | 66.7±0.02 | 76.3±0.02 |
| | **0.1** | 77.1±0.02 | 67.3±0.02 | 77.4±0.01 |
| | 0.2 | 76.2±0.01 | 67.2±0.01 | 76.7±0.01 |
| | 0.3 | 76.8±0.01 | 66.9±0.01 | 77.1±0.01 |
| | 0.4 | 76.3±0.01 | 67.4±0.01 | 76.8±0.02 |
| | 0.5 | 76.3±0.00 | 66.6±0.01 | 76.5±0.01 |
| | 0.8 | 71.4±0.03 | 62.6±0.03 | 71.7±0.03 |
| N. Filters | 2 | 53.4±0.07 | 41.2±0.05 | 54.6±0.07 |
| | 5 | 62.0±0.03 | 51.4±0.03 | 62.3±0.03 |
| | 10 | 68.8±0.02 | 56.7±0.01 | 68.9±0.01 |
| | 50 | 75.0±0.01 | 64.3±0.01 | 74.9±0.01 |
| | **100** | 76.6±0.02 | 66.1±0.02 | 76.9±0.02 |
| | 200 | 75.7±0.01 | 66.2±0.01 | 76.9±0.01 |
| Stride | **1** | 77.1±0.00 | 66.7±0.01 | 77.1±0.01 |
| | 2 | 72.7±0.01 | 60.6±0.01 | 72.6±0.01 |
| | 3 | 70.2±0.01 | 56.5±0.02 | 69.8±0.02 |
| | 4 | 68.5±0.02 | 52.4±0.03 | 67.0±0.02 |
| | 5 | 65.9±0.02 | 49.2±0.01 | 64.9±0.02 |

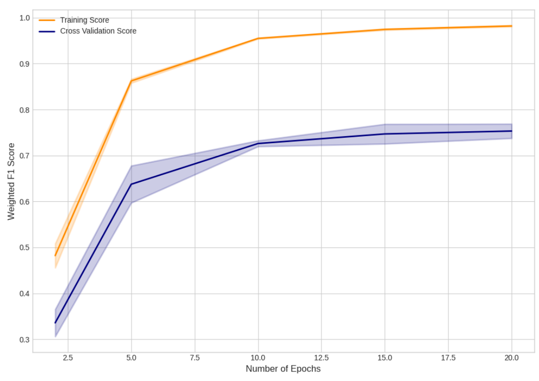
**Notes:** Hyperparameters were tuned using validation curves during model development. All analyses were performed using the final best pipeline. Micro-, Macro- and Weighted-F1 Scores all taken over all positive classes (excluding those without labels/not relevant). All values reported with standard deviations. The optimum values chosen for each hyperparameters are highlighted in bold.



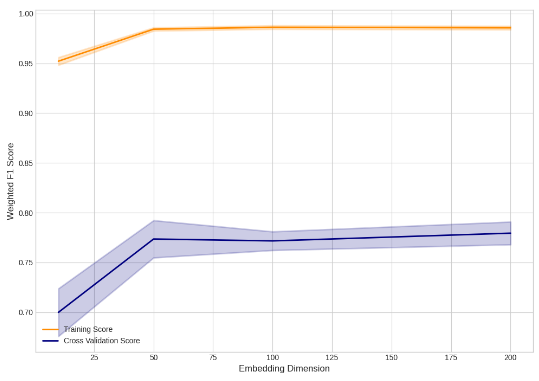
(a) Learning Rate



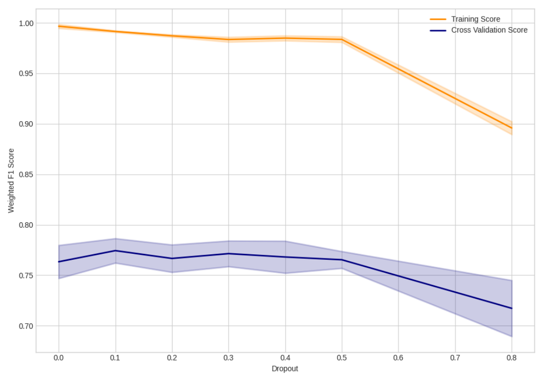
(b) Batch Size



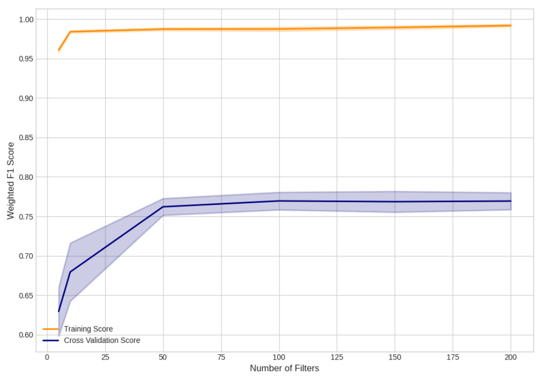
(c) Number of Epochs



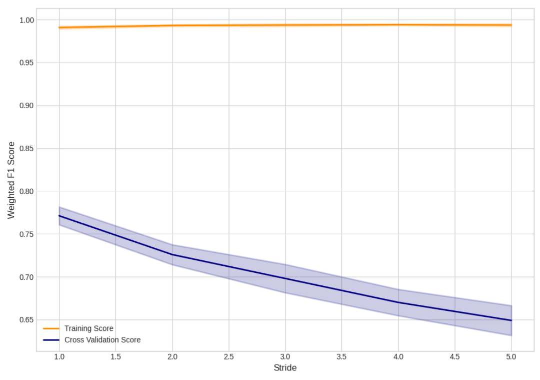
(d) Embedding Size



(e) Dropout

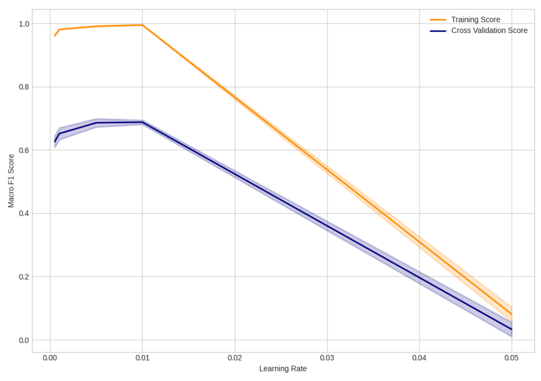


(f) Number of Filters

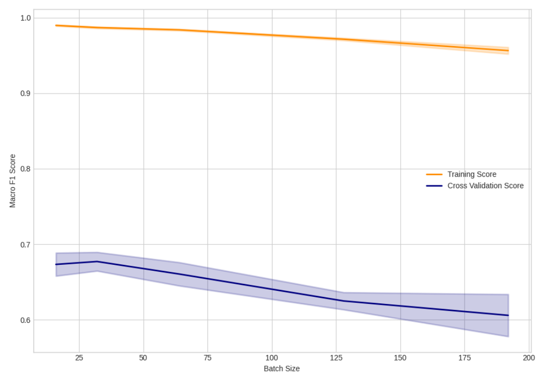


(g) Stride

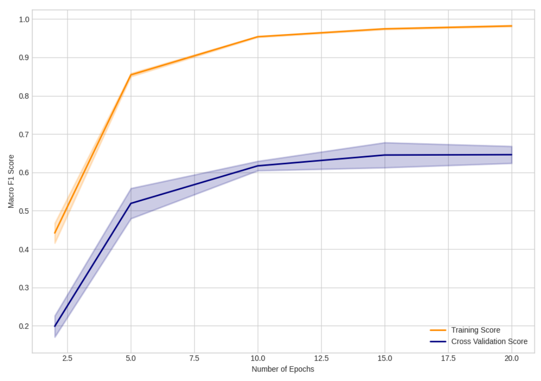
**Figure 14: Validation Curves of Weighted-F1 Score over different hyperparameters values from 5-fold Cross-validation using the final best pipeline**



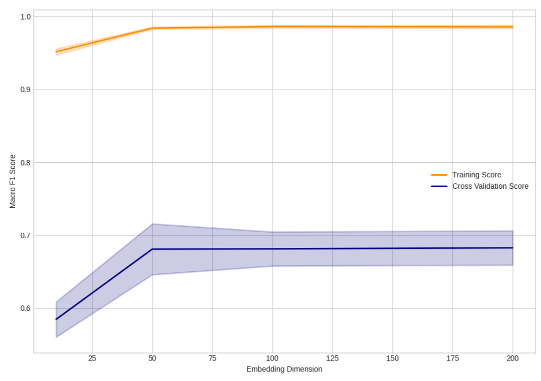
(a) Learning Rate



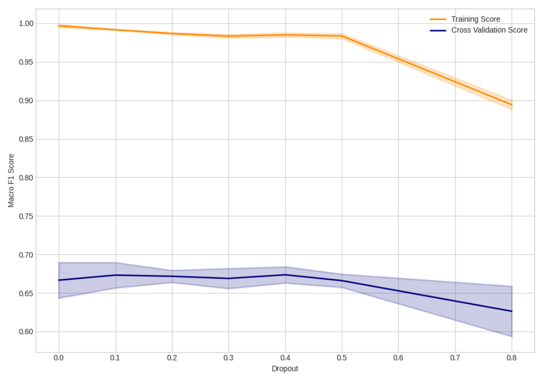
(b) Batch Size



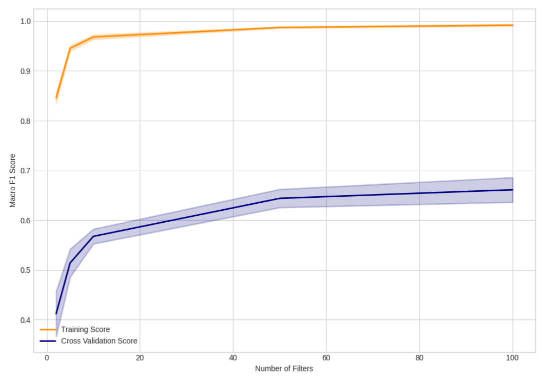
(c) Number of Epochs



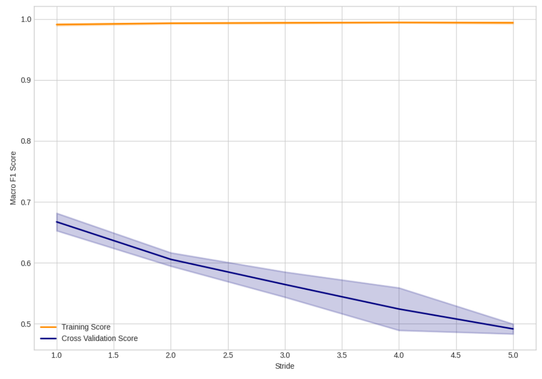
(d) Embedding Size



(e) Dropout



(f) Number of Filters



(g) Stride

**Figure 15: Validation Curves of Macro-F1 Score over different hyperparameters values from 5-fold Cross-validation using the final best pipeline**

## 4.7 Final Pipeline

Finally, the final best ML pipeline with optimised hyperparameters, summarised in Table 12, was fit to the entire Training set. The pipeline was then applied to the entire Test set and summary metrics were calculated and are displayed in Table 13. Overall, the best performance estimates of the final pipeline on unseen data were a Micro-F1 of 0.76, a Macro-F1 was 0.66 and a Weighted-F1 of 0.76. In addition, per class metrics were calculated and can be seen in Table 14. Performance is particularly high for 'Non-compartmental Parameters' and 'Number of Subjects' (above 0.8) and reasonably high for 'Doses' and 'Demographics' (above 0.7) all of which have high support in the data set. These metrics reveal even with class imbalance methods employed and optimised hyper-parameters, performance on minority classes where support is low is still limited. Particularly for 'Parameters Other', 'Sample Timings' and 'Covariates Other'. 'Compartmental Parameters' and 'Parameter-Covariate Relationships' achieve reasonably high F1-scores given their low support. Notably, the original imbalance seen between Precision and Recall for minority classes in earlier analyses has been improved by under-sampling and augmentation, with Precision and Recall being much more balanced across all classes.

Comparing these results directly to the average annotator F1-scores, reveals that this model is doing equally well or better than the average individual annotators on many classes ('Non-Compartmental Parameters', 'Compartmental Parameters', 'Parameter-Covariate Relationships', 'Parameters Other'). The model F1-scores are lower than the average individual annotators but in the same order of magnitude for the remaining classes ('Doses', 'Number of Subjects', 'Sample Timings', 'Demographics') except for 'Covariates Other' on which the model performs especially poorly but it is notable that the Kappa Agreement Score between annotators is severely limited for this class and annotator F1-score is less than 0.5.

Lastly, a qualitative assessment was performed to manually check the predictions versus the labels on 100 examples from the Test Set. In general, true positives on the analysis were confirmed to be error free in terms of the data labelling. Figure 16 shows examples from the qualitative analysis, showing some true positive cases for a selection of labels and corresponding false positive and false negative cases. Interestingly, a false positive occurs for ‘Number of Subjects’ when there is presence of ‘n=8’ but this is out of context (does not occur to subjects in this case) (Figure 16b). The classifier also falsely predicts ‘Non-compartmental Parameters’ when the true label is ‘Compartmental Parameters’ (Figure 16f). Notably, Doses and Number of subjects are predicted incorrectly when tokens corresponding to units, which can also be used for doses, and ‘Total’, which can often be used in conjunction with ‘Subjects’ or ‘N’, are present (Figure 16i).

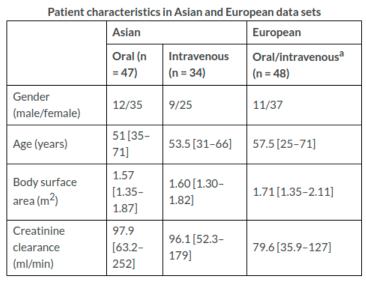
**Table 12: Optimised Architecture and Hyperparameters for the Final Model**

| | |
|---|---|
| Base Architecture | N-GRAM CNN |
| Multihot-vector | No |
| Preprocessing | HTML Table Tags Removed |
| Tokenization | SentecePiece BPE Model trained on PK tables, vocab size=5000 |
| Augmentation | Numbers |
| Undersampling | Yes |
| Maximum Token Length | 500 |
| Learning rate | 0.01 |
| Number of Training Epochs | 15 |
| Batch size | 32 |
| Drop-out | 0.1 |
| Embedding size | 50 |
| Number of Filters | 100 |
| Stride | 1 |

**Table 13: Summary Performance Metrics of the Final Pipelines on the Test Set**

| **Micro-F1** | **Macro-F1** | **Weighted-F1** |
|---|---|---|
| 0.78 | 0.66 | 0.76 |

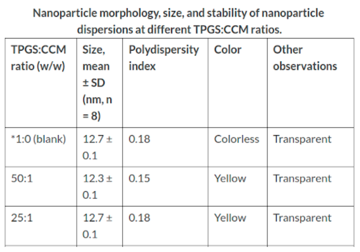
**Notes:** Micro-, Macro- and Weighted-F1 Scores all taken over all positive classes (excluding those without labels/not relevant).



Patient characteristics in Asian and European data sets

| | Asian | | European |
|---|---|---|---|
| | Oral (n = 47) | Intravenous (n = 34) | Oral/intravenous[a] (n = 48) |
| Gender (male/female) | 12/35 | 9/25 | 11/37 |
| Age (years) | 51 [35–71] | 53.5 [31–66] | 57.5 [25–71] |
| Body surface area ($m^2$) | 1.57 [1.35–1.87] | 1.60 [1.30–1.82] | 1.71 [1.35–2.11] |
| Creatinine clearance (ml/min) | 97.9 [63.2–252] | 96.1 [52.3–179] | 79.6 [35.9–127] |

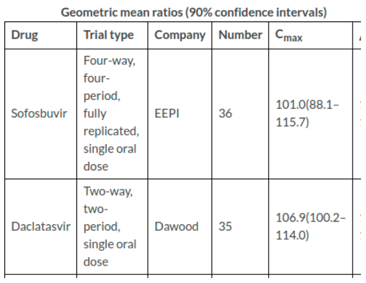
(a) TP Number of Subjects and Demographics/FN Covariates Other



Nanoparticle morphology, size, and stability of nanoparticle dispersions at different TPGS:CCM ratios.

| TPGS:CCM ratio (w/w) | Size, mean ± SD (nm, n = 8) | Polydispersity index | Color | Other observations |
|---|---|---|---|---|
| *1:0 (blank) | 12.7 ± 0.1 | 0.18 | Colorless | Transparent |
| 50:1 | 12.3 ± 0.1 | 0.15 | Yellow | Transparent |
| 25:1 | 12.7 ± 0.1 | 0.18 | Yellow | Transparent |

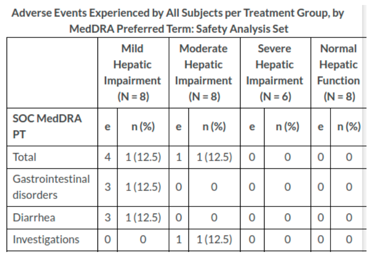
(b) FP Number of Subjects



Geometric mean ratios (90% confidence intervals)

| Drug | Trial type | Company | Number | $C_{max}$ |
|---|---|---|---|---|
| Sofosbuvir | Four-way, four-period, fully replicated, single oral dose | EEPI | 36 | 101.0(88.1–115.7) |
| Daclatasvir | Two-way, two-period, single oral dose | Dawood | 35 | 106.9(100.2–114.0) |

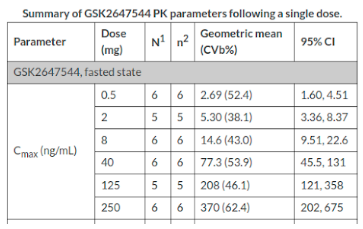
(c) FP Demographics



Adverse Events Experienced by All Subjects per Treatment Group, by MedDRA Preferred Term: Safety Analysis Set

| | Mild Hepatic Impairment (N = 8) | | Moderate Hepatic Impairment (N = 8) | | Severe Hepatic Impairment (N = 6) | | Normal Hepatic Function (N = 8) | |
|---|---|---|---|---|---|---|---|---|
| **SOC MedDRA PT** | e | n (%) | e | n (%) | e | n (%) | e | n (%) |
| Total | 4 | 1 (12.5) | 1 | 1 (12.5) | 0 | 0 | 0 | 0 |
| Gastrointestinal disorders | 3 | 1 (12.5) | 0 | 0 | 0 | 0 | 0 | 0 |
| Diarrhea | 3 | 1 (12.5) | 0 | 0 | 0 | 0 | 0 | 0 |
| Investigations | 0 | 0 | 1 | 1 (12.5) | 0 | 0 | 0 | 0 |

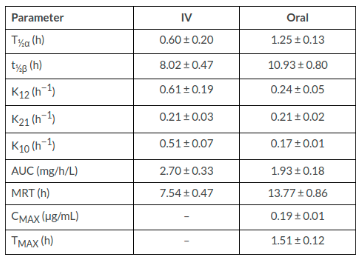
(d) TP Covariates Other



Summary of GSK2647544 PK parameters following a single dose.

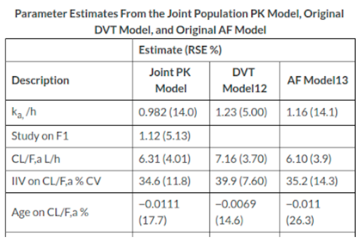
| Parameter | Dose (mg) | N[1] | n[2] | Geometric mean (CVb%) | 95% CI |
|---|---|---|---|---|---|
| GSK2647544, fasted state | | | | | |
| $C_{max}$ (ng/mL) | 0.5 | 6 | 6 | 2.69 (52.4) | 1.60, 4.51 |
| | 2 | 5 | 5 | 5.30 (38.1) | 3.36, 8.37 |
| | 8 | 6 | 6 | 14.6 (43.0) | 9.51, 22.6 |
| | 40 | 6 | 6 | 77.3 (53.9) | 45.5, 131 |
| | 125 | 5 | 5 | 208 (46.1) | 121, 358 |
| | 250 | 6 | 6 | 370 (62.4) | 202, 675 |

(e) TP Non-Compartmental Parameters



| Parameter | IV | Oral |
|---|---|---|
| $T_{½α}$ (h) | 0.60 ± 0.20 | 1.25 ± 0.13 |
| $t_{½β}$ (h) | 8.02 ± 0.47 | 10.93 ± 0.80 |
| $K_{12}$ ($h^{-1}$) | 0.61 ± 0.19 | 0.24 ± 0.05 |
| $K_{21}$ ($h^{-1}$) | 0.21 ± 0.03 | 0.21 ± 0.02 |
| $K_{10}$ ($h^{-1}$) | 0.51 ± 0.07 | 0.17 ± 0.01 |
| AUC (mg/h/L) | 2.70 ± 0.33 | 1.93 ± 0.18 |
| MRT (h) | 7.54 ± 0.47 | 13.77 ± 0.86 |
| $C_{MAX}$ (μg/mL) | – | 0.19 ± 0.01 |
| $T_{MAX}$ (h) | – | 1.51 ± 0.12 |

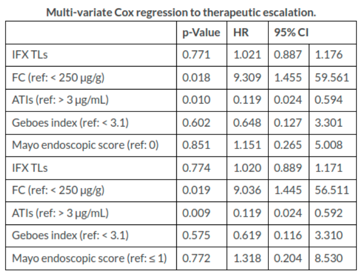
(f) FP Non-Compartmental Parameters/FN Compartmental Parameters



Parameter Estimates From the Joint Population PK Model, Original DVT Model, and Original AF Model

| | Estimate (RSE %) | | |
|---|---|---|---|
| Description | Joint PK Model | DVT Model12 | AF Model13 |
| $k_a$, /h | 0.982 (14.0) | 1.23 (5.00) | 1.16 (14.1) |
| Study on F1 | 1.12 (5.13) | | |
| CL/F,a L/h | 6.31 (4.01) | 7.16 (3.70) | 6.10 (3.9) |
| IIV on CL/F,a % CV | 34.6 (11.8) | 39.9 (7.60) | 35.2 (14.3) |
| Age on CL/F,a % | −0.0111 (17.7) | −0.0069 (14.6) | −0.011 (26.3) |

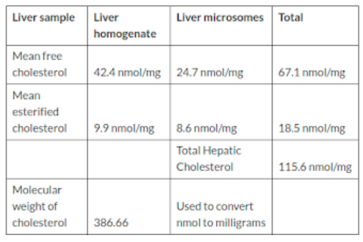
(g) TP Compartmental Parameters



Multi-variate Cox regression to therapeutic escalation.

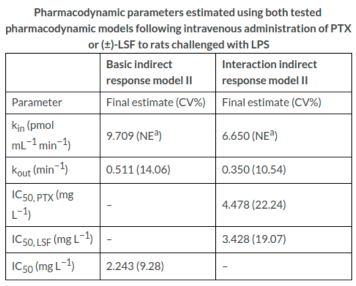
| | p-Value | HR | 95% CI | |
|---|---|---|---|---|
| IFX TLs | 0.771 | 1.021 | 0.887 | 1.176 |
| FC (ref: < 250 μg/g) | 0.018 | 9.309 | 1.455 | 59.561 |
| ATIs (ref: > 3 μg/mL) | 0.010 | 0.119 | 0.024 | 0.594 |
| Geboes index (ref: < 3.1) | 0.602 | 0.648 | 0.127 | 3.301 |
| Mayo endoscopic score (ref: 0) | 0.851 | 1.151 | 0.265 | 5.008 |
| IFX TLs | 0.774 | 1.020 | 0.889 | 1.171 |
| FC (ref: < 250 μg/g) | 0.019 | 9.036 | 1.445 | 56.511 |
| ATIs (ref: > 3 μg/mL) | 0.009 | 0.119 | 0.024 | 0.592 |
| Geboes index (ref: < 3.1) | 0.575 | 0.619 | 0.116 | 3.310 |
| Mayo endoscopic score (ref: ≤ 1) | 0.772 | 1.318 | 0.204 | 8.530 |

(h) TP Not Relevant



| Liver sample | Liver homogenate | Liver microsomes | Total |
|---|---|---|---|
| Mean free cholesterol | 42.4 nmol/mg | 24.7 nmol/mg | 67.1 nmol/mg |
| Mean esterified cholesterol | 9.9 nmol/mg | 8.6 nmol/mg | 18.5 nmol/mg |
| | | Total Hepatic Cholesterol | 115.6 nmol/mg |
| Molecular weight of cholesterol | 386.66 | Used to convert nmol to milligrams | |

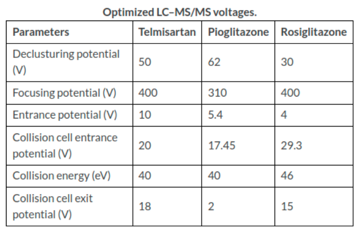
(i) FN Not Relevant/FP Doses Number of Subjects



Pharmacodynamic parameters estimated using both tested pharmacodynamic models following intravenous administration of PTX or (±)-LSF to rats challenged with LPS

| | Basic indirect response model II | Interaction indirect response model II |
|---|---|---|
| Parameter | Final estimate (CV%) | Final estimate (CV%) |
| $k_{in}$ (pmol $mL^{-1}$ $min^{-1}$) | 9.709 (NE[a]) | 6.650 (NE[a]) |
| $k_{out}$ ($min^{-1}$) | 0.511 (14.06) | 0.350 (10.54) |
| $IC_{50, PTX}$ (mg $L^{-1}$) | – | 4.478 (22.24) |
| $IC_{50, LSF}$ (mg $L^{-1}$) | – | 3.428 (19.07) |
| $IC_{50}$ (mg $L^{-1}$) | 2.243 (9.28) | – |

(j) TP Parameters Other



Optimized LC–MS/MS voltages.

| Parameters | Telmisartan | Pioglitazone | Rosiglitazone |
|---|---|---|---|
| Declusturing potential (V) | 50 | 62 | 30 |
| Focusing potential (V) | 400 | 310 | 400 |
| Entrance potential (V) | 10 | 5.4 | 4 |
| Collision cell entrance potential (V) | 20 | 17.45 | 29.3 |
| Collision energy (eV) | 40 | 40 | 46 |
| Collision cell exit potential (V) | 18 | 2 | 15 |

(k) FN Parameters Other

**Figure 16: Examples from the qualitative analysis of tables from the Test Set and their predicted versus true labels**

**Table 14: Performance Metrics Calculated per Class for the Final Pipelines on the Test Set**

| **Class Label** | **Support** | Pr | Re | F1 |
|---|---|---|---|---|
| Non-Compartmental Parameters | 185 | 0.94 | 0.93 | 0.93 |
| Compartmental Parameters | 30 | 0.77 | 0.57 | 0.65 |
| Parameter-Covariate Relationships | 23 | 0.88 | 0.65 | 0.75 |
| Parameters Other | 29 | 0.54 | 0.45 | 0.49 |
| Doses | 242 | 0.89 | 0.71 | 0.79 |
| Number of Subjects | 322 | 0.88 | 0.80 | 0.84 |
| Sample Timings | 65 | 0.56 | 0.54 | 0.55 |
| Demographics | 228 | 0.68 | 0.76 | 0.72 |
| Covariates Other | 49 | 0.21 | 0.24 | 0.23 |

**Notes:** Pr is Precision, Re is Recall and F1 is F1-score. Calculated on the Test Set on model output predictions compared to true labels for each table.

# 5 Discussion

Overall, this project developed a PK table classification pipeline from scratch. To the best of my knowledge, no such pipeline has been attempted before in the PK domain. The Classifier Pipeline takes in tables in PMC-XML format from PK papers and outputs a set of labels for each table in terms on the parameters and covariates contained therein. The final pipeline achieves a Micro-F1 score of 0.78, a Macro-F1 score on 0.66 and a Weighted-F1 score of 0.76 and the Classifier Pipeline even outperforms a single annotator in terms of F1-score on four out of nine classes. The overall pipeline performance is limited by severe class imbalance in the data set, due to low performance on minority classes. This pipeline will benefit clinical pharmacologists helping them to identify tables from the PK literature which contain parameters and covariates of interest for clinical PK modelling and drug development. Furthermore, this pipeline provides the initial step towards an automatic data extraction from tables in the PK literature, into a centralised database to provide the big data needed for ML methods to be employed more readily in PK modelling.

## 5.1 Annotator Agreement

The K between annotators revealed a lower level of agreement (under 0.7) on less prevalent, broader classes such as 'Covariates Other' (Figure 4). This is likely because annotators saw these classes less often and thus using our iterative approach to updating labelling guidelines, fewer examples of less prevalent classes in each labelling round meant less edge cases were revealed and documented, resulting in greater uncertainty surrounding these infrequent classes. In addition, less frequently occurring classes were more easily missed as they were not expected to occur. Mean annotator F1-scores, showed the same pattern with higher F1-scores on more frequently occurring classes. Particularly low mean F1-scores were observed for 'Parameter-Covariate Relationships' and 'Covariates Other' (Figure 4). For the former this is a fairly straightforward category but was likely frequently missed during labelling as it tended to occur in the 'body' of the table. However, for the latter this was a broad category and therefore often more widely interpreted leading to lower K and mean F1-scores compared to the final agreed labels. It it important to note that a low K makes resolution more difficult in review sessions and likely therefore leads to more variance within those classes. This is a limitation in that greater within class variance on infrequent classes makes it more difficult for the Classifier to learn those classes reliably, hindering performance.

## 5.2 Tokenization

Tokenization with a vocabulary size of 10,000 showed limited performance overall and on majority classes for both PMOA and PK data sets (Figure 7) suggesting 10,000 vocabulary is too large for these tables. This allows too many whole words to become tokens and consequently overfitting to the specific words in the training set[59] and therefore the Tokenizer is not generalisable enough to new examples in unseen data. This problem is particularly acute for majority classes which are more frequent in the data set, leading to poorer performance on these. However interestingly for PK tables alone, which by definition have a smaller vocabulary compared to all PMOA tables, the highest Macro-F1 score (which considers all classes equally) was achieved using 10,000. In this way, a much greater vocabulary may be benefiting the minority classes, allowing less frequently occurring words to be tokenized as larger tokens, for which more meaningful context-independent representations can be more readily learnt, and in turn provide more rich information to the Classifier. It follows that performance with a vocabulary size of 3000 was also lower across the board of summary performance metrics suggesting 3000 is too small a vocabulary. Not enough larger tokens are able to be created and smaller sub-word tokens are more prevalent, which are more difficult for the Classifier to learn a meaningful context-independent representation for.[59] A vocabulary size of 5000 offered the best compromise between these two sizes allowing more larger sub-words with meaningful context-independent representations but still allowing transferability to new unseen text. It is difficult to strike a balance due to the severe class imbalance on this data, as lower Macro-F1 score compared to the other metrics still suggests that the Tokenizer is more optimised for majority classes and so is not a perfect solution. Ideally, a tokenizer would be trained on balanced data set, however in practise this is simply not practical as a large amount data is required to train a tokenizer and so unlabelled data was used which did not allow for class balancing methods to be readily employed.

Classes with low support achieved a lower F1 score with all Tokenizer variations (Table 17) and it was noted Precision is high and Recall is low. Precision is a measure of the relevancy of a result whereas Recall is a measure of how many truly relevant results are returned. High Precision and low Recall on a class can be interpreted as the Classifier returning very few results for that class but the majority of its predicted labels being correct, when compared to the true labels.[60] Essentially, in most cases where they do occur, minority classes are not being predicted but when they are predicted they are mostly correct. This is likely because the Classifier has seen a relatively small quantity of this class in training, so it has learnt a stringent representation of it.[61,62] Conversely, on majority classes there is both high Precision and Recall meaning many labels are returned and most are correct. "Covariates other" achieved particularly low Precison, Recall and F1 scores for all Tokenizers. This is likely due to the broad nature of this class, making it more difficult for the network to learn this class particularly with the low frequency.

For both tokenizer and preprocessing variations, validation loss increased after around 15 epochs, as validation Weighted-F1 began to level off although this was still increasing very slightly for all Tokenizer variants (Figure 9c, 9d). Beyond 15 epochs the network is likely over-fitting to majority classes (especially as Weighted-F1 score favours majority classes). In this way stopping training at this point or shortly after could be beneficial to majority class performance on unseen data. However, the slight increase in F1-score over time may indicate that the network may be still learning minority classes after this point. This is a challenge with multi-label classification as the network is likely learning spurious patterns and useful ones simultaneously[62] and so cutting off the learning process early could cap model performance on minority classes.

## 5.3 Preprocessing

Preprocessing to leave only the baseline without HTML, reduced performance considerably across all summary metrics (Figure 8) unlike for Kardas *et al.*[33] and Milosevic *et al.*[30] where the caption alone achieves good classification performance for ML tables and adverse events/demographics tables respectively. This is likely because some essential information indicating PK class labels can be found in the 'body' of the table. Leaving in the HTML tags, pertaining to table structure, enhanced performance on majority classes (Micro and Weighted-F1) to some extent over the baseline, but had especially low Macro-F1 scores across all maximum token lengths (Figure 8). The most likely explanation is that table tags add more noise to the data, masking signal from tokens associated with minority classes. Removing HTML tags completely, and including the 'body' of the table gave the best results across the board when using a maximum length of 500 tokens. Macro-F1, is still relatively weak Figure 8), again due to the severe class imbalance. It is unlikely preprocessing variations can help to deal with class imbalance in any significant way as tokens pertaining to all classes in general can be present in different table regions or the caption. However, including HTML tags or trying to reduce the length of input to the model has a negative effect on minority classes. It appears that HTML tags or marking table regions do not provide important structural information, suggesting token location within the table is not essential information for table classification.

Similar to the Tokenizer results, minority classes had high Precision and low Recall except for 'Covariates Other' which was low across the board (Table 18). This can be attributed to the severe class imbalance leading the Classifier to predict minority classes very rarely although precisely as it has learnt stringent representations for these.[61, 62]

Neural networks are affected by input length in terms of performance as longer sequences by definition contain more 'noise' but shorter sequences may cut out essential information for classification.[44] This accounts for why within removing HTML, 1000 tokens is too many and less than 500 tokens is too few, with 500 tokens being the optimum input length for the Classifier (Figure 8).

## 5.4 Table Representations

Bootstrap results showed embeddings to be the optimal method for representing tables in terms of having a higher Micro- and Weighted-F1 and a less variable Macro-F1 (Table 9). Previous biomedical text classification studies agreed with this outcome when there is sufficient labelled training data.[63, 64] This is as expected, as embeddings are contextual representations for each token, which can be updated by back-propagation during training, making it possible to learn a detailed context-dependent representation for each token. However, BOW are fixed spare vectors, which have a lot of empty space without valuable information likely contributing to increased 'noise' and therefore variance for the Macro-F1s.

## 5.5 Model Architecture

The CNN architecture achieved higher Micro- and Macro- performance compared to the MLP on all variants (Figure 13, 13b). In order to feed the embedding matrix into the MLP, the matrix was manually max-pooled to squeeze salient information into one dimension. This max-pooling results in information loss before passing the table into the model likely explaining lower MLP performance. However, the CNN instead accepts the embedding matrix as input in its original form and carries out convolution on this to pull out salient information into many feature maps meaning less information is lost and more complex features can be extracted.[42]

Concatenating a multi-hot vector to the input of the final Classifier fully-connected layer, improved Macro-F1 score for the MLP (Figure 13b). This implies the signal pertaining to the minority classes is more likely to be lost during max-pooling as the information is less salient than other information in the input (especially as this is multi-label data so it is likely a minority label often co-occurs with a majority label). Therefore, reminding the MLP of this information is helpful to minority class classification but detrimental to majority class classification as this dilutes the features from these that have been learnt by the network, hence the lower Micro- and Weighted-F1 scores. This effect is not seen in the CNN (Figure 13) where data is not max-pooled at the beginning therefore maintaining information on minority classes and so the multi-hot vectors likely just dilute the features learnt by the CNN.

## 5.6 Class Imbalance

With class label frequencies ranging from as low as 59 to as high as 687 on the training set, this poses a key challenge to ML, as model designs are generally built around the assumption of of equal sized classes.[61] Under-sampling produced a marked increase in Macro-F1 for both the MLP and CNN (Figure 13b), signifying an improvement on minority classes. This is likely as the model sees proportionally more of the minority classes therefore does not over-fit so much to majority classes. Although, due to the multi-label nature of this data, this is limited as under-sampling cannot produce an exact class balance. The drop in Weighted-F1 seen with under-sampling for the MLP (Figure 13) is not surprising as under-sampling effectively penalises majority classes, removing samples from the data set and therefore reducing performance on these classes. From this perspective, a limitation of under-sampling is that it does not make use of all the labelled data available and thus reduces performance on classes which have sufficient data to achieve high performance.[65] Interestingly, this effect is not so marked on the CNN, perhaps as avoiding max-pooling at the beginning allows more robust patterns to be learnt on less data. However, Weighted-F1 IQV is increased for the CNN, suggesting a reduction in total data set size does lead to more variable results on majority classes.

Data augmentation of synonyms, increasing the total data set size, did not improve summary metrics for the CNN or boost Macro-F1 as with under-sampling (Figure 13, 13b). This is likely attributable to classes still being heavily unbalanced, as augmentation simply increases the data set size without changing the distribution of classes. Minority classes were disproportionately affected by augmentation of synonyms (Figure 19). This could be the result of the hand-crafted synonyms dictionary, which in reality may augment some classes more than others which may result in the model effectively seeing some samples twice as little has been changed about them. Conversely, due to the small representation of some classes within the training set, augmenting their synonyms may change them so much as to make them more difficult to learn as there is not a large enough distribution of them.

Augmenting numbers improved summary metrics across the board for the MLP (Figure 13, 13b) suggesting it improves score on both minority and majority classes. This suggests the MLP may be learning spurious connections between numerical values and certain classes and training the model on synthetic examples with augmented numbers helps to stop the model learning these spurious associations. However, Macro-F1 score was not increased to the extent as with under-sampling, again likely as the class distribution is unchanged in this larger augmented data set. The CNN showed a similar pattern with Micro- and Weighted-F1 scores increasing with augmentation of numbers but Macro-F1 remaining low (Figure 13, 13b), suggesting the CNN is also learning some spurious relationships between numbers and majority classes. However, for minority classes, the main limiting factor is still the data distribution and so augmentation cannot provide real gains without also correcting the severe class imbalance. Due to the multi-label nature of this data it is also difficult to augment only minority classes as there are few example with only one label and most of these are for majority classes.

Augmentation of both synonyms and numbers was also tried in combination, in case for example with augmenting only numbers alone, the model would over-fit to the non-numerical tokens from the input as it will see these more than once and vice versa with augmenting synonyms. Augmenting both did not give the same improvement as seen with augmentation of numbers alone on either the MLP and the CNN (Figure 13, 13b) suggesting that the model is not over-fitting to synonyms when numbers alone are changed. It is expected that augmenting both would give better results but perhaps this is changing the table too much away from its original form to be recognisable to its class, although it is still recognisable to the human observer. This could be the fault of the hand-crafted synonym dictionary which may not cover all the nuances in the data. More sophisticated augmentation techniques such as learning to augment data with a neural network,[66] generating synthetic examples using generative adversarial networks (GANs) or simply cutting out areas of each table[67] could benefit performance.

Combining augmentation of numbers with under-sampling was expected to give the best results by increasing the size and variance in the data set before under-sampling so there would be more samples in each class. The median Weighted-F1 and Macro-F1 were slightly higher that under-sampling alone for the CNN but likely there was not a big increase as the class distribution is still the same as with under-sampling alone (as there is the same proportions of examples with the same multi-label combinations) (Figure 13, 13b).

## 5.7 Hyperparameters

Learning rate on the Final Best Model was optimum from 0.01 to 0.005 with learning rates above or below this giving a significant decrease in all summary performance metrics(Figure 11). This is because, too small a learning rate can makes the convergence rate slow (and would therefore need more training epochs) and has more potential to get stuck in local minima, whereas too large a learning rate means parameters will oscillate back on forth on both sides of the optimal solution.[68] Changing batch sizes had less of a strong effect on the summary performance metrics with 16, 32 and 64 all giving the best summary performance metrics and only a larger batch size of 128 causing a significant decline in summary performance metrics (Figure 11). This can be attributed to smaller batch sizes being more 'noisy' and therefore offering a regularising effect, but too smaller batch sizes are not representative and the network cannot converge. Larger batch sizes however make the gradient direction nearly stable and at risk of falling into a local optimum.[68]

Summary performance metrics increased with the number of epochs until 15 epochs after which only a small significant increase in Micro- and Weighted-F1 scores and no change in Macro-F1 was seen up to 20 epochs (Figure 11). This is because more training epochs allows the model to learn more real connections but also more spurious connections, widening the gap between training and validation performance metrics, and consequently overfitting to the majority classes.[68] Due to the imbalanced nature of this data it is difficult to choose the optimum time to stop training and likely the majority classes will be learnt by the model much faster during training as these are more frequent in the training set. At the stage where the model starts to over-fit to the training set for these classes, it is likely still learning on minority classes.

Increasing embedding size, increased all summary metrics up until a dimension of 50, after which there is no significant increase (Figure 11) as too smaller embedding dimension means the context of each token in the input table cannot be represented meaningfully. However, there reaches a critical size above which tokens increasing the embedding size only serves to increase computational cost.[69] A dropout of 0.1 was the best, giving a small significant increase across the board on summary metrics, however dropout up to 0.5 did not improve things over no dropout and too greater dropout (0.8) lead to a drop in all summary performance metrics (Figure 11). Dropout forces the neural network to learn more robust features and too great a dropout inhibits the model from learning, whereas a small amount of dropout should help to avoid overfitting and force the whole network to learn. Notably data augmentation also has a regularizing effect and may well outperform other regularization methods,[66] perhaps accounting for why dropout does not benefit the model performance much in this case.

Increasing the number of filters (number of neurons in CNN) improved summary performance metrics up to a threshold of 100 after which there was no significant performance gains (Figure 11). The higher the number of filters, the higher the number of abstractions (feature maps) the the network is able to make from the input embedding matrix, extracting more complex features. However, above a certain threshold increasing the number of filters only increases parameters in the model and computational cost without performance gains. Increasing the stride (tokens the convolutional filter slides over at a time) value decreased performance (Figure 11). This is expected as if a pattern is visible more strongly on a position that is skipped, vital information is lost, so a larger stride is equal to retaining less information. So in this case a stride of one is necessary to retain all the information and as any token can hold key information.

## 5.8 Final Model

The final pipeline achieves a Micro-F1 score of 0.78, a Macro-F1 score on 0.66 and a Weighted-F1 score of 0.76. Macro-F1 score was limited due to severe class imbalance and the per class results show overall Precision, Recall and F1-score are higher on majority classes (Figure 13). ‘Parameter Covariate Relationships’ and ‘Compartmental Parameters’ still perform reasonably well despite their low frequency (Figure 14). Undersampling is likely responsible for this allowing a more even distribution of these in the training data set, but at the expense of reducing majority class samples in the training set which has likely resulted in the lower final Micro- and Weighted-F1 scores than would have been achieved in a pipeline without under-sampling.

Per class F1 score was particularly high for ‘Non-compartmental Parameters’, ‘Number of Subjects’ and ‘Doses’ (all over 0.75) (Figure 14). This is a good result for our future application of the model our group are particularly interested in finding non-compartmental analyses results along with the corresponding doses and number of subjects. However, other PK groups may wish to extract different combinations and so the final pipeline achieved the best compromise across classes of all the pipelines tested. Also, per class Precision and Recall are reasonably balanced for all classes unlike in earlier analyses meaning the pipeline returns results and gets correct results equally well. This is particularly an improvement for minority classes where previously this was markedly skewed towards Precision. Performance on some minority classes was still severely limited even with under-sampling and augmentation, particularly for ‘Parameters Other’ and ‘Covariates Other’. Interestingly, these classes were broader in nature in terms of their definition and they also both achieved a low annotator K and mean annotator F1-score, whereas all the classes on which performance is high from the Classifier Pipeline, K and mean annotator F1 are high (Figure 4). From this perspective, class definitions and consistency of data labelling may be limiting model performance on ‘Parameters Other’ and ‘Covariates Other’ which cannot be fixed through re-sampling and data augmentation techniques.

The final Classifier Pipeline achieves F1-scores in the order of magnitude of an individual PK expert annotator. Particularly notable, the pipeline outperforms the mean PK-annotator on F1-score on 4/9 classes (‘Non-Compartmental Parameters’, ‘Compartmental Parameters’, ‘Parameter-Covariate Relationships’, ‘Parameters Other’) (Figure 14, 4). This link between annotator agreement metrics and final model F1-score is an interesting one demonstrating that model performance is closely linked to data quality and higher uncertainty between annotators likely leads to greater variance in final labels and results in more complexity for the model to learn.

A qualitative assessment on 100 examples from the Test Set revealed some interesting examples of where the model is predicting incorrectly, from which insight can be gained on improvements that could be made. The false positive for 'Number of Subjects' (Figure 16b) is likely the results of the presence 'n=8' in a different context. Additionally, 'Doses' and 'Number of subjects' are predicted falsely (Figure 16i) likely due to dose units and the use of 'Total'. This could suggest the Classifier would benefit from more sequential understanding to better represent tokens in different contexts and so an LSTM (type of RNN) architecture may be beneficial in these cases. However, compensatory methods to deal with long inputs would need to be applied as 500 tokens is needed for optimum performance and LSTMs performance is limited for long sequences. The false positive 'Non-compartmental Parameters' (Figure 16f) when the true label is 'Compartmental Parameters' likely occurred as these two classes do have overlapping token distributions and due to the low frequency of the 'Compartmental Parameters' class suggesting more training samples are needed around the decision boundary of these two classes.

## 5.9 Limitations

Final performance, particularly on minority classes, is a limitation of this table classification pipeline. Severe class imbalance in our labelled data (which was randomly sampled from the true distribution) is a serious limitation to the development of a high performing, robust model. Undersampling, one method used to tackle class imbalance, is limited as it reduces the total number of samples the model is being trained on and changes the class distribution away from the true distribution in the literature. This serves to both penalise the majority classes lowering performance on unseen data for those classes and risks that minority classes will be over-predicted when this model is applied more widely.[65] In addition, even with under-sampling, the Tokenizer has been trained on unbalanced data from the unlabelled PK tables data set. This is a limitation as even with under-sampling of the labelled data, minority classes are not represented so well by tokenization.

As discussed, inter-annotator agreement being low on certain classes, particularly those with low frequency, is a limitation. High uncertainly surrounding a class will likely mean even final agreed labels are inconsistent and there is more within class variability which will make the class harder to learn.[70] Ideally, a high K should be achieved on all classes before training a model, but due to severe class imbalance, the broad nature of some of the minority categories and time constraints of the project, this was not achievable.

Notably, any correlation between classes is not taken into account using this model architecture as each class is viewed as a binary classification problem. Therefore, any innate correlations which may help the model to predict the correct label set for an instance are not able to contribute. Interestingly, Facebook get round this by predicting a probability distribution of label sets and using a softmax activation to minimise cross-entropy between the predicted distribution and target distribution for each data sample.[71]

Finally, although the project is available on github, this work is of limited benefit to the PK community until it is translated into an online tool, on which tables with certain parameters and covariates can be found using a search-engine-like interface.

## 5.10 Future Perspectives

Following on from the above, a key future goal is to translate this classifer pipeline into an online-search-engine for tables in PMC papers and integrate it with the document Classifier already developed by Hernandez *et al.*[28] This is essential to make this pipeline widely accessible and usable by those with and without ML expertise in the PK community. Additionally, tables and captions could benefit from sequential understanding.[44] Additionally, Kardas et al.[33] successfully employ an ULMFit Model, which has an LSTM architecture, to classify ML results tables based on their captions. A future model architecture could therefore either use an LSTM, chunking the input and combining results at the end or use a hybrid approach with an LSTM captions with a CNN for the rest of the table and combine their outputs.

Looking forward, for this tool to be used widely performance on minority classes needs further improvement. As observed, techniques to balance and synthetically increase minority classes are limited so more data needs to be collected for minority classes, but ideally in an automated fashion. Firstly, active learning could be employed to make better use of PK experts by having them label data points which are estimated to be most valuable to the model, so selecting tables that lie close to the decision boundary for minority classes.[72,73] These examples could then be formally labelled by PK experts and added to the labelled data set. These approaches can be complimented with weak supervision, a commonly used method to get lower quality labels more efficiently at a higher abstraction level by using pattern matching and user-defined heuristics. Weak supervision could also be used independently from active learning to generate lower quality labels for pretraining the Classifier before training it with high quality supervised labels.[74,75] Furthermore, Zero-shot or Few-shot learning is a field of ML concerned with predicting topics on which the model has not been trained, or has only seen a few examples of. This could be of real benefit for this task, allowing classes to be defined as specific token lists and encoding these in parallel to the tables. Comparing similarity of a table to the class encoding before backpropagating, allows human knowledge to be transferred to the model meaning fewer examples are needed per class. This method has been demonstrated successfully with GPT-3 and BERT language models for classes with few or no samples.[76,77]

## 5.11 Conclusion

This project developed the first labelled data set of tables from PK literature alongside a PK table classification pipeline for tables from PK papers in terms on the parameters and covariates they contain. The final best pipeline achieved good performance on all majority classes but further qualitative analysis of results suggests a recurrent architecture such as an LSTM could further boost performance. Performance on some broader minority classes was limited due to severe class imbalance, despite use of undersampling and data augmentation techniques, so collecting more labelled data for these classes could bring real improvements. Overall, the pipeline achieves as good an F1 as the mean PK annotator across majority classes and in some cases exceeds PK annotator performance. Once translated into a usable tool, the Classifier Pipeline will be an asset to clinical pharmacologists and drug developers, helping them to filter tables from the PK literature containing parameters and covariates of interest for their PK modelling task. In a wider context, this pipeline provides the initial step towards a table data extraction pipeline to extract key PK prior data from tables in the literature into a centralised PK database to benefit drug development and clinical dosing.

## 6 Acknowledgements

I would like to thank my supervisors for their support and guidance throughout this project, allowing me freedom to direct the project to explore new areas and improve my skills and knowledge in ML, NLP and Pharmacometrics. I would also like to thank the annotation team for their invaluable contribution to data labelling, without which this project would not have been possible.

# 7 Appendix

**Table 15: Performance Metrics calculated per Class for Table Representation Variations**

| **Class Label** | **Embeddings** | **BOW** |
|---|---|---|
| Non-Compartmental Parameters | 90.2(80.4, 92.9) | 78.7(74.3,82.3) |
| Compartmental Parameters | 16.0(0.0,53.5) | 0.0(0.0,64.9) |
| Parameter-Covariate Relationships | 0.0(0.0,30.5) | 0.0(0.0,40.8) |
| Parameters Other | 0.0(0.0,14.6) | 0.0(0.0,21.2) |
| Doses | 64.6(51.5,74.1) | 68.8(65.5,72.4) |
| Number of Subjects | 80.5(72.2,83.5) | 71.5(69.1,74.0) |
| Sample Timings | 0.0(0.0,9.0) | 24.2(0.0,45.6) |
| Demographics | 69.6(63.6,75.2) | 64.8(60.7,70.2) |
| Covariates Other | 0.0(0.0,0.0) | 0.0(0.0,20.8) |

**Notes:** Weighted-F1-scores from 100 bootstrap iterations on the entire training set. All analyses performed on tables with HTML tags removed, using the PK-5000 tokenizer and an MLP pipeline.

**Table 16: Chi-Squared Results on two-way tables for Label Frequency Distributions and Multi-label occurrences, p=0.05**

| | Critical Value | Training Set | Validation Set | Test Set |
|---|---|---|---|---|
| Label Frequencies | 26.296 | 11.04750301 | 4.798322067 | 12.89395383 |
| Multi-label Occurrences | 18.307 | 9.696084302 | 2.990512204 | 11.18226427 |
| Significance | - | Not Significant | Not Significant | Not Significant |

**Table 17: Performance Metrics calculated Per Class for Different Tokenizer Variations**

| | | **All-tables** | | | | | | **PK-tables** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **10,000** | | | **5,000** | | | **10,000** | | | **5,000** | | | **3,000** | | |
| **Class Label** | **Support** | Pr | Re | F1 | Pr | Re | F1 | Pr | Re | F1 | Pr | Re | F1 | Pr | Re | F1 |
| Non-Compartmental Parameters | 91 | 0.88 | 0.93 | 0.91 | 0.92 | 0.89 | 0.90 | 0.94 | 0.83 | 0.88 | 0.95 | 0.79 | 0.86 | 0.91 | 0.89 | 0.90 |
| Compartmental Parameters | 15 | 0.82 | 0.30 | 0.44 | 1.00 | 0.30 | 0.46 | 0.60 | 0.10 | 0.17 | 0.94 | 0.50 | 0.65 | 0.88 | 0.23 | 0.37 |
| Parameter-Covariate Relationships | 15 | 0.78 | 0.30 | 0.44 | 0.70 | 0.30 | 0.42 | 0.60 | 0.26 | 0.36 | 1.00 | 0.26 | 0.41 | 0.71 | 0.22 | 0.33 |
| Parameters Other | 21 | 0.35 | 0.21 | 0.26 | 0.45 | 0.17 | 0.25 | 0.00 | 0.00 | 0.00 | 0.37 | 0.38 | 0.37 | 0.40 | 0.28 | 0.38 |
| Doses | 117 | 0.63 | 0.57 | 0.60 | 0.84 | 0.74 | 0.79 | 0.78 | 0.69 | 0.73 | 0.89 | 0.79 | 0.84 | 0.83 | 0.72 | 0.77 |
| Number of Subjects | 163 | 0.83 | 0.82 | 0.82 | 0.79 | 0.81 | 0.80 | 0.79 | 0.77 | 0.86 | 0.87 | 0.84 | F1 | 0.80 | 0.81 | 0.81 |
| Sample Timings | 23 | 0.29 | 0.08 | 0.12 | 0.73 | 0.12 | 0.21 | 0.50 | 0.02 | 0.03 | 0.54 | 0.31 | 0.39 | 0.60 | 0.28 | 0.38 |
| Demographics | 104 | 0.84 | 0.62 | 0.71 | 0.86 | 0.65 | 0.74 | 0.97 | 0.48 | 0.65 | 0.96 | 0.63 | 0.76 | 0.78 | 0.60 | 0.68 |
| Covariates Other | 19 | 0.25 | 0.02 | 0.04 | 0.00 | 0.00 | 0.00 | 0.30 | 0.06 | 0.10 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

**Notes:** Pr is Precision, Re is Recall and F1 is F1-score. Calculated on the Test Set on model outputted predictions compared to true labels for each table. All analyses performed with an MLP pipeline using embeddings.

**Table 18: Performance Metrics Calculated per Class for different Table Pre-processing Variations**

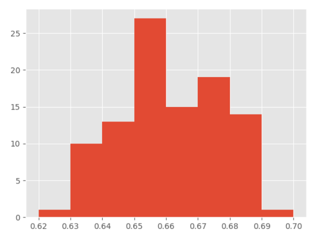
| | | Style Removed | | | | | | | | | Regions Included HTML Removed | | | | | | | | | | | | | | | Regions Marked | | | Baseline* | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 2000 | | | 1000 | | | 500 | | | 1000 | | | 500 | | | 300 | | | 200 | | | 100 | | | 500 | | | 300 | | | 100 | | |
| **Class Label** | **Support** | Pr | Re | F1 | Pr | Re | F1 | Pr | Re | F1 | Pr | Re | F1 | Pr | Re | F1 | Pr | Re | F1 | Pr | Re | F1 | Pr | Re | F1 | Pr | Re | F1 | Pr | Re | F1 | Pr | Re | F1 |
| Non-Compartmental Parameters | 91 | 0.92 | 0.90 | 0.91 | 0.92 | 0.89 | 0.90 | 0.92 | 0.90 | 0.91 | 0.93 | 0.84 | 0.88 | 0.95 | 0.79 | 0.86 | 0.93 | 0.91 | 0.92 | 0.94 | 0.89 | 0.91 | 0.88 | 0.88 | 0.88 | 0.95 | 0.89 | 0.92 | 0.87 | 0.90 | 0.88 | 0.90 | 0.89 | 0.89 |
| Compartmental Parameters | 15 | 0.81 | 0.43 | 0.57 | 0.71 | 0.40 | 0.51 | 0.74 | 0.47 | 0.57 | 0.83 | 0.33 | 0.48 | 0.94 | 0.50 | 0.65 | 0.88 | 0.50 | 0.64 | 0.79 | 0.50 | 0.61 | 0.90 | 0.30 | 0.45 | 0.95 | 0.67 | 0.78 | 0.82 | 0.30 | 0.44 | 0.71 | 0.40 | 0.51 |
| Parameter-Covariate Relationships | 15 | 1.00 | 0.04 | 0.08 | 0.50 | 0.04 | 0.08 | 0.80 | 0.17 | 0.29 | 0.60 | 0.13 | 0.21 | 1.00 | 0.26 | 0.41 | 0.42 | 0.22 | 0.29 | 0.44 | 0.30 | 0.36 | 0.33 | 0.04 | 0.08 | 1.00 | 0.22 | 0.36 | 0.67 | 0.26 | 0.38 | 0.40 | 0.26 | 0.32 |
| Parameters Other | 21 | 0.50 | 0.03 | 0.06 | 0.19 | 0.10 | 0.13 | 0.80 | 0.17 | 0.29 | 0.35 | 0.21 | 0.26 | 0.37 | 0.38 | 0.37 | 0.29 | 0.07 | 0.11 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.03 | 0.07 | 0.40 | 0.14 | 0.21 | 0.67 | 0.07 | 0.12 |
| Doses | 117 | 0.76 | 0.73 | 0.74 | 0.81 | 0.73 | 0.77 | 0.81 | 0.73 | 0.77 | 0.79 | 0.82 | 0.81 | 0.89 | 0.79 | 0.84 | 0.81 | 0.76 | 0.79 | 0.87 | 0.73 | 0.79 | 0.80 | 0.79 | 0.79 | 0.76 | 0.76 | 0.76 | 0.75 | 0.55 | 0.64 | 0.82 | 0.60 | 0.70 |
| Number of Subjects | 163 | 0.80 | 0.87 | 0.83 | 0.87 | 0.76 | 0.81 | 0.80 | 0.85 | 0.83 | 0.86 | 0.82 | 0.84 | 0.87 | 0.84 | 0.86 | 0.85 | 0.82 | 0.84 | 0.87 | 0.79 | 0.83 | 0.85 | 0.78 | 0.81 | 0.78 | 0.86 | 0.82 | 0.76 | 0.75 | 0.76 | 0.81 | 0.70 | 0.75 |
| Sample Timings | 23 | 0.00 | 0.00 | 0.00 | 0.33 | 0.03 | 0.06 | 0.00 | 0.00 | 0.00 | 0.54 | 0.23 | 0.32 | 0.54 | 0.31 | 0.39 | 0.36 | 0.08 | 0.13 | 0.62 | 0.12 | 0.21 | 0.67 | 0.22 | 0.33 | 0.45 | 0.20 | 0.28 | 0.88 | 0.11 | 0.19 | 0.57 | 0.12 | 0.20 |
| Demographics | 104 | 0.89 | 0.68 | 0.77 | 0.92 | 0.57 | 0.70 | 0.97 | 0.56 | 0.71 | 0.91 | 0.64 | 0.75 | 0.96 | 0.63 | 0.76 | 0.91 | 0.61 | 0.73 | 0.94 | 0.58 | 0.72 | 0.87 | 0.61 | 0.71 | 0.97 | 0.59 | 0.74 | 0.94 | 0.59 | 0.72 | 0.95 | 0.58 | 0.72 |
| Covariates Other | 19 | 0.00 | 0.00 | 0.00 | P0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.02 | 0.04 | 0.00 | 0.00 | 0.00 | 0.33 | 0.06 | 0.10 | 0.00 | 0.00 | 0.00 | 0.44 | 0.08 | 0.14 | 0.00 | 0.00 | 0.00 | 0.57 | 0.08 | 0.14 | 0.20 | 0.02 | 0.04 |

**Notes:** Pr is Precision, Re is Recall and F1 is F1-score. Calculated on the Test Set on model outputted predictions compared to true labels for each table. All analyses performed with the PK-5000 tokenizer and an MLP pipeline with embeddings.

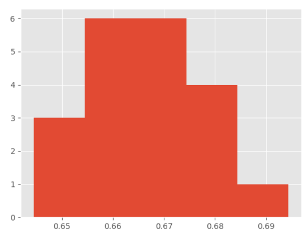
**Table 19: Bootstrap Results per Class for Architecture and Class Imbalance Strategy Variations**

| | $F1_w$ **Median (95% CIs)** | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Architecture** | **MLP** | | | | | **MLP+ Mutlihot** | **CNN** | | | | | | **CNN + Mutlihot** |
| Augmentation | None | None | Syn | Num | Syn& Num | None | None | None | Syn | Num | Num | Syn & Num | None |
| Resampling | No | Yes | No | No | No | No | No | Yes | No | No | Yes | No | No |
| **Class Label** | | | | | | | | | | | | | |
| Non-Compartmental Parameters | 90.2(80.4, 92.9) | 77.0(63.4,86.3) | 90.1(81.2,93.0) | 90.2(83.0,93.1) | 90.1(81.1, 93.2) | 85.2(82.6,87.4) | 93.3(89.9,94.9) | 91.3(85.0, 94.6) | 93.0(89.6,94.9) | 93.5(90.8, 95.0) | 91.6(85.5,94.4) | 93.4(89.9,93.4) | 90.7(88.7,93.2) |
| Compartmental Parameters | 16.0(0.0,53.5) | 50.3(32.0,63.6) | 34.5(0.0,56.1) | 34.9(1.8,56.7) | 32.0(0.0,56.9) | 45.0(37.2,57.0) | 14.1(0.0,51.0) | 56.5(26.7,74.4) | 19.8(0.0,46.7) | 45.5(16.8, 67.3) | 55.2(31.5,75.4) | 10.3(0.0,46.3) | 25.5(11.9,39.9) |
| Parameter-Covariate Relationships | 0.0(0.0,30.5) | 42.9(28.6,56.9) | 15.9(0.0,41.8) | 12.6(0.0,45.4) | 12.5(0.0,43.5) | 27.9(16.3,43.8) | 14.0(0.0,42.7) | 51.8(33.3,66.7) | 15.2(0.0,48.1) | 36.2(8.3,57.6) | 50.0(23.9,69.0) | 12.9(0.0,41.9) | 13.1(4.5,27.4) |
| Parameters Other | 0.0(0.0,14.6) | 22.3(8.7, 33.2) | 4.6(0.0,19.5) | 3.8(0.0,22.6) | 5.8(0.0, 22.7) | 11.1(2.9,21.4) | 76.2(67.2,81.0) | 18.0(0.0,38.6) | 0.0(0.0,9.1) | 3.1(0.0, 12.1) | 18.6(0.0,37.1) | 0.0(0.0,5.6) | 0.0(0.0,6.6) |
| Doses | 64.6(51.5,74.1) | 53.3(37.6,67.2) | 69.4(58.0,76.5) | 69.3(54.5,76.8) | 69.3(55.1,76.3) | 68.4(63.8,71.0) | 76.2(67.2,81.0) | 62.5(33.9,70.8) | 76.3(68.3,81.5) | 80.5(77.8, 84.3) | 62.8(37.1, 71.7) | 76.4(67.3,81.7) | 71.1(65.9,75.3) |
| Number of Subjects | 80.5(72.2,83.5) | 72.5(60.2,78.0) | 80.1(73.9,83.2) | 80.6(71.6,83.4) | 80.1(72.9,83.4) | 78.0(76.1,80.5) | 85.5(82.9,88.1) | 83.1(79.1,85.5) | 85.5(83.3,87.9) | 83.5(88.4,86.2) | 82.8(79.1,85.9) | 85.5(82.6,87.5) | 84.6(81.6,86.3) |
| Sample Timings | 0.0(0.0,9.0) | 30.4(19.0, 44.1) | 0.0(0.0,21.8) | 2.7(0.0,27.3) | 2.7(0.0, 30.3) | 22.0(12.9,30.2) | 5.2(0.0,31.1) | 41.6(9.7,52.8) | 2.8(0.0,30.1) | 20.7(2.4,49.5) | 40.5(8.6,58.4) | 5.0(0.0,30.8) | 9.1(2.5,19.4) |
| Demographics | 69.6(63.6,75.2) | 57.0(49.1,64.1) | 69.9(65.4,74.3) | 69.5(63.7,74.0) | 70.0(64.2,74.5) | 66.4(65.2,71.3) | 72.3(68.8, 75.9) | 66.3(59.9,72.1) | 72.8(68.8,76.3) | 74.7(70.7,79.7) | 66.4(59.8,71.8) | 72.6(67.6,76.4) | 71.3(66.2,74.6) |
| Covariates Other | 0.0(0.0,0.0) | 22.0(15.4,32.2) | 0.0(0.0,15.2) | 0.0(0.0,13.3) | 0.0(0.0, 15.1) | 14.7(5.5,22.2) | 0.0(0.0,9.4) | 20.3(5.5,33.5) | 0.0(0.0,9,5) | 6.7(0.0,20.7) | 19.6(5.5,34.5) | 0.0(0.0,8.7) | 3.3(0.0,10.7) |

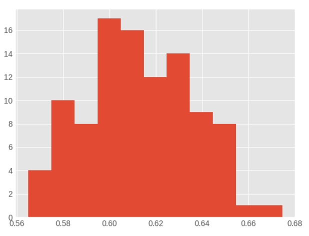
**Notes:** Weighted-F1-scores from 100 bootstrap iterations on the entire training set. All analyses performed on tables with HTML tags removed and using a pipeline with the PK-5000 tokenizer and embeddings.



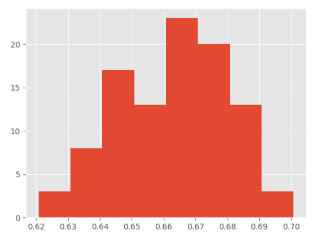
(a) MLP



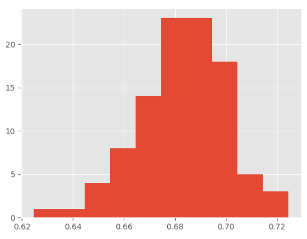
(b) MLP + Multihot Vector



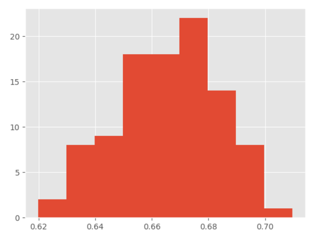
(c) MLP + Resampling



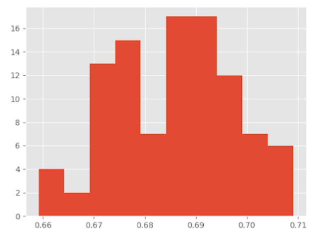
(d) MLP + Augment Synonyms



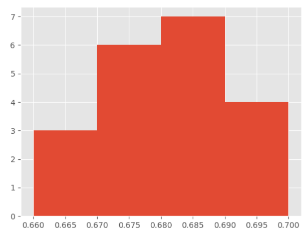
(e) MLP + Augment Numbers



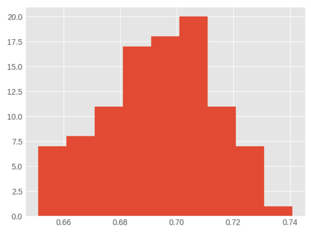
(f) MLP+ Augment Both



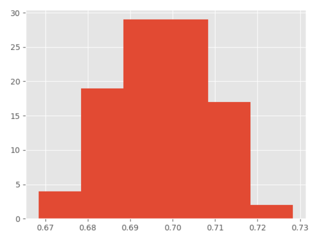
(g) CNN



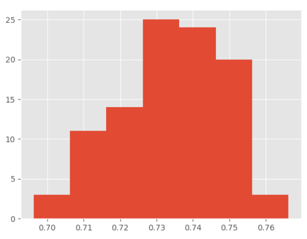
(h) CNN + Multihot Vector



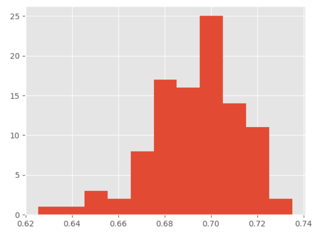
(i) CNN+ Resampling



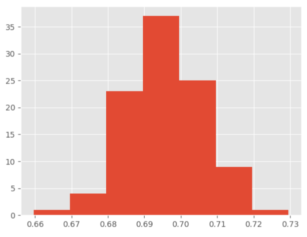
(j) CNN+ Augment Synonyms



(k) CNN+ Augment Numbers



(l) CNN+ Augment Numbers + Resample



(m) CNN+ Augment Both

**Figure 17: Distribution of Weighted-F1 scores from 100 Bootstrap Iterations with various architecture and class imbalance method variations. All analyses performed on tables with HTML tags removed and using a pipeline with the PK-5000 tokenizer and embeddings.**

```python
import torch
from torch import nn
from torch.nn import functional as F, Sequential
# ========== MLP with 2 Hidden Layers and Learnable Embedding Matrix ============#
class MLP(nn.Module):
    def __init__(self, numb_classes, h_size, embeds_size, vocab_size, padding_idx, drop_out):
        """
        numb_classes : 9  (classes)
        h_size: number of neurons in the hidden layer
        vocab_size: total size of tokenizer vocabulary + padding tokens )
        padding_idx: token index of padding token
        drop_out: percentage of neurons to drop the outputs of for regularisation
        """
        super(MLP, self).__init__()
        self.embedding = torch.nn.Embedding(vocab_size, embeds_size, padding_idx)
        self.fc1 = nn.Linear(embeds_size, h_size)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(drop_out)
        self.fc2 = nn.Linear(h_size, numb_classes)

    def forward(self, x):
        embeddings = self.embedding(x)
        max_pooled = torch.max(embeddings, dim=1, keepdim=False)[0]
        out_fc1 = self.fc1(max_pooled)
        out_relu = self.relu(out_fc1)
        out_dropout = self.dropout(out_relu)
        out_fc2 = self.fc2(out_dropout)
        return out_fc2
```

```python
# ========== NGRAM CNN with Learnable Embedding Matrix ============#
class CNN(nn.Module):
    def __init__(self, numb_classes, inp_channels, numb_filters, filter_sizes, embeds_size, vocab_size, padding_idx, stride, drop_out
):
        super(CNN, self).__init__()
        """
        numb_classes : 9  (classes)
        inp_channels : Number of input channels. 1 as the input data has dimension = (batch_size * input_length * embeds_size)
        filter_sizes : List of different kernel_heights. Convolution is performed 4 times and results from each kernel_height are concatenated.
        numb_filters : number of neurons in each layer
        stride: number of tokens the filter shifts over the input matrix by.
        embeds_size: dimension by which to represent each token (embedding matrix will be max_length by embeds_size)
        vocab_size: total size of tokenizer vocabulary + padding tokens)
        padding_idx: token index of padding token
        drop_out: percentage of neurons to drop the outputs of for regularisation
        """
        self.embedding = torch.nn.Embedding(vocab_size, embeds_size, padding_idx=padding_idx)
        self.output_size = numb_classes
        self.inp_channels = inp_channels
        self.numb_filters = numb_filters
        self.filter_sizes = filter_sizes
        self.stride = stride

        # convolutional layers definition
        self.c1 = nn.ModuleList(
            [nn.Conv2d(self.inp_channels, self.numb_filters, (k_height, embeds_size), self.stride) for k_height in self.filter_sizes])
        self.drop_out = nn.Dropout(drop_out)
        self.fc1 = nn.Linear((len(filter_sizes) * numb_filters), numb_classes)

    def forward(self, x):
```

```python
        embeds = self.embedding(x)
        input = embeds.unsqueeze(1)
        convs = [F.relu(c(input)).squeeze(3) for c in self.c1]
        pooled = [F.max_pool1d(i, i.size(2)).squeeze(2) for i in c]
        concat = torch.cat(pooled, 1)
        concat = self.dropout(concat)
        logits = self.fc1(concat)
        return logits
```

```python

# ========== MLP with MutliHot Encoded Vector and Learnable Embedding Matrix ============#
class MLP_HOT(nn.Module):
    def __init__(self, numb_classes, h_size, embeds_size, vocab_size, padding_idx, drop_out):
        """
        numb_classes : 9  (classes)
        hidden_size: number of neurons in the hidden layer
        vocab_size: total size of tokenizer vocabulary + padding tokens)
        padding_idx: token index of padding token
        drop_out: percentage of neurons to drop the outputs of for regularisation
        """
        super(MLP_HOT, self).__init__()
        self.embedding = torch.nn.Embedding(vocab_size, embeds_size, padding_idx=padding_idx)
        self.fc1 = nn.Linear(embeds_size, h_size)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(drop_out)
        self.fc2 = nn.Linear((h_size + vocab_size), numb_classes)

    def forward(self, x, multi):
        embeddings = self.embedding(x)
        max_pooled = torch.max(embeddings, dim=1, keepdim=False)[0]
        out_fc1 = self.fc1(max_pooled)
        out_relu = self.relu(out_fc1)
        out_dropout = self.dropout(out_relu)
        concat_multi = torch.cat((multi, out_dropout), dim=1)
        out_fc2 = self.fc2(concat_multi)
        return out_fc2
```

```python
# ========== CNN MutliHot Encoded Vector and Learnable Embedding Matrix ============#
class CNN_HOT(nn.Module):
    def __init__(self, numb_classes, inp_channels, numb_filters, filter_sizes, embeds_size, vocab_size, padding_idx, stride, drop_out):
        super(CNN_HOT, self).__init__()
        """
        numb_classes : 9  (classes)
        inp_channels : Number of input channels. 1 as the input data has dimension = (batch_size * input_length * embeds_size)
        filter_sizes : List of different kernel_heights. Convolution is performed 4 times and results from each kernel_height are concatenated.
        numb_filters : number of neurons in each layer
        stride: number of tokens the filter shifts over the input matrix by.
        embeds_size: dimension by which to represent each token (embedding matrix will be max_length by embeds_size)
        vocab_size: total size of tokenizer vocabulary + padding tokens)
        padding_idx: token index of padding token
        drop_out: percentage of neurons to drop the outputs of for regularisation
        """
        self.embedding = torch.nn.Embedding(vocab_size, embeds_size, padding_idx=padding_idx)
        self.output_size = numb_classes
        self.inp_channels = inp_channels
        self.numb_filters = numb_filters
        self.filter_sizes = filter_sizes
        self.stride = stride

        # conv layers definition
        self.c1 = nn.ModuleList(
            [nn.Conv2d(self.inp_channels, self.numb_filters, (k_height,
    embeds_size), self.stride) for k_height in self.filter_sizes])
        self.dropout = nn.Dropout(drop_out)
        self.fc1 = nn.Linear(((len(filter_sizes) * numb_filters) +
   vocab_size), numb_classes)

    def forward(self, x, multi):
        embeds = self.embedding(x)
        input = embeds.unsqueeze(1)
        convs = [F.relu(conv(input)).squeeze(3) for conv in self.c1]
        pooled = [F.max_pool1d(i, i.size(2)).squeeze(2) for i in convs]
        concat = torch.cat(pooled, 1)
        concat = self.dropout(concat)
        concat = torch.cat((concat, multi), dim=1)
        logits = self.fc1(concat)
        return logits
```

## References

[1] Chi Heem Wong, Kien Wei Siah, and Andrew W Lo. Estimation of clinical trial success rates and related parameters. *Biostatistics*, 20(2):273–286, 2019.

[2] Paul Morgan, Piet H Van Der Graaf, John Arrowsmith, Doug E Feltner, Kira S Drummond, Craig D Wegner, and Steve DA Street. Can the flow of medicines be improved? fundamental pharmacokinetic and pharmacological principles toward improving phase ii survival. *Drug discovery today*, 17(9-10):419–424, 2012.

[3] Joseph F Standing. Understanding and applying pharmacometric modelling and simulation in clinical practice and research. *British journal of clinical pharmacology*, 83(2):247–254, 2017.

[4] Charlotte IS Barker, Eva Germovsek, Rollo L Hoare, Jodi M Lestner, Joanna Lewis, and Joseph F Standing. Pharmacokinetic/pharmacodynamic modelling approaches in paediatric infectious diseases and immunology. *Advanced drug delivery reviews*, 73:127–139, 2014.

[5] Ene I Ette and Paul J Williams. Population pharmacokinetics i: background, concepts, and models. *Annals of Pharmacotherapy*, 38(10):1702–1706, 2004.

[6] Heng-Yi Wu, Shreyas Karnik, Abhinita Subhadarshini, Zhiping Wang, Santosh Philips, Xu Han, Chienwei Chiang, Lei Liu, Malaz Boustani, Luis M Rocha, et al. An integrated pharmacokinetics ontology and corpus for text mining. *BMC bioinformatics*, 14(1):1–15, 2013.

[7] Johan Gabrielsson and Daniel Weiner. Non-compartmental analysis. In *Computational toxicology*, pages 377–389. Springer, 2012.

[8] Diane R Mould and RN Upton. Basic concepts in population modeling, simulation, and model-based drug development. *CPT: pharmacometrics & systems pharmacology*, 1(9):1–14, 2012.

[9] Huixi Zou, Parikshit Banerjee, Sharon Shui Yee Leung, and Xiaoyu Yan. Application of pharmacokinetic-pharmacodynamic modeling in drug delivery: Development and challenges. *Frontiers in Pharmacology*, 11:997, 2020.

[10] Hannah M Jones, Iain B Gardner, and Kenny J Watson. Modelling and pbpk simulation in drug discovery. *The AAPS journal*, 11(1):155–166, 2009.

[11] Mohamad Shebley, Punam Sandhu, Arian Emami Riedmaier, Masoud Jamei, Rangaraj Narayanan, Aarti Patel, Sheila Annie Peters, Venkatesh Pilla Reddy, Ming Zheng, Loeckie de Zwart, et al. Physiologically based pharmacokinetic model qualification and reporting procedures for regulatory submissions: a consortium perspective. *Clinical Pharmacology & Therapeutics*, 104(1):88–110, 2018.

[12] Jenny Y Chien, Stuart Friedrich, Michael A Heathman, Dinesh P de Alwis, and Vikram Sinha. Pharmacokinetics/pharmacodynamics and the stages of drug development: role of modeling and simulation. *The AAPS journal*, 7(3):E544–E559, 2005.

[13] Daniela Schuster, Christian Laggner, and Thierry Langer. Why drugs fail-a study on side effects in new chemical entities. *Current pharmaceutical design*, 11(27):3545–3559, 2005.

[14] Zhuyifan Ye, Yilong Yang, Xiaoshan Li, Dongsheng Cao, and Defang Ouyang. An integrated transfer learning and multitask learning approach for pharmacokinetic parameter prediction. *Molecular pharmaceutics*, 16(2):533–541, 2018.

[15] Mason McComb, Robert Bies, and Murali Ramanathan. Machine learning in pharmacometrics: Opportunities and challenges. *British Journal of Clinical Pharmacology*, 2021.

[16] D. McGinnity. Evolution of drug metabolism and pharmacokinetics in drug discovery and development. *European Pharmaceutical Review*, 17, 01 2012.

[17] Rodolfo S Simões, Vinicius G Maltarollo, Patricia R Oliveira, and Kathia M Honorio. Transfer and multi-task learning in qsar modeling: advances and challenges. *Frontiers in pharmacology*, 9:74, 2018.

[18] M Paul Gleeson, Nigel J Waters, Stuart W Paine, and Andrew M Davis. In silico human and rat v ss quantitative structure- activity relationship models. *Journal of medicinal chemistry*, 49(6):1953–1963, 2006.

[19] Yuchen Wang, Haichun Liu, Yuanrong Fan, Xingye Chen, Yan Yang, Lu Zhu, Junnan Zhao, Yadong Chen, and Yanmin Zhang. In silico prediction of human intravenous pharmacokinetic parameters with improved accuracy. *Journal of chemical information and modeling*, 59(9):3968–3980, 2019.

[20] OMx Personal Health Analytics Inc. Drugbank, 2006.

[21] National Institutes of Health. Pubchem, 2004.

[22] MultiMedia LLC. Pharmgkb, 2000.

[23] EMBL-EBI. Chembl, 2011.

[24] George Papadatos, Anna Gaulton, Anne Hersey, and John P Overington. Activity, assay and target data curation and quality in the chembl database. *Journal of computer-aided molecular design*, 29(9):885–896, 2015.

[25] KR Przybylak, JC Madden, E Covey-Crump, L Gibson, C Barber, M Patel, and MTD Cronin. Characterisation of data resources for in silico modelling: benchmark datasets for adme properties. *Expert opinion on drug metabolism & toxicology*, 14(2):169–181, 2018.

[26] Jan Grzegorzewski, Janosch Brandhorst, Kathleen Green, Dimitra Eleftheriadou, Yannick Duport, Florian Barthorscht, Adrian Köller, Danny Yu Jia Ke, Sara De Angelis, and Matthias König. Pk-db: pharmacokinetics database for individualized and stratified computational modeling. *Nucleic acids research*, 49(D1):D1358–D1364, 2021.

[27] Franco Lombardo, Giuliano Berellini, and R Scott Obach. Trend analysis of a database of intravenous pharmacokinetic parameters in humans for 1352 drug compounds. *Drug Metabolism and Disposition*, 46(11):1466–1477, 2018.

[28] Ferran Gonzalez Hernandez, Simon J Carter, Juha Iso-Sipilä, Paul Goldsmith, Ahmed A Almousa, Silke Gastine, Watjana Lilaonitkul, Frank Kloprogge, and Joseph F Standing. An automated approach to identify scientific publications reporting pharmacokinetic parameters. *Wellcome Open Research*, 6(88):88, 2021.

[29] Zhiping Wang, Seongho Kim, Sara K Quinney, Yingying Guo, Stephen D Hall, Luis M Rocha, and Lang Li. Literature mining on pharmacokinetics numerical data: a feasibility study. *Journal of biomedical informatics*, 42(4):726–735, 2009.

[30] Nikola Milosevic, Cassie Gregson, Robert Hernandez, and Goran Nenadic. A framework for information extraction from tables in biomedical literature. *International Journal on Document Analysis and Recognition (IJDAR)*, 22(1):55–78, 2019.

[31] Sophia Ananiadou and John McNaught. *Text mining for biology and biomedicine.* Citeseer, 2006.

[32] M Hurst. The interpretation of tables in texts phd thesis. *University of Edinburgh, School of Cognitive Science, Informatics*, 2000.

[33] Marcin Kardas, Piotr Czapla, Pontus Stenetorp, Sebastian Ruder, Sebastian Riedel, Ross Taylor, and Robert Stojnic. Axcell: Automatic extraction of results from machine learning papers. *arXiv preprint arXiv:2004.14356*, 2020.

[34] Daniel Hanisch, Katrin Fundel, Heinz-Theodor Mevissen, Ralf Zimmer, and Juliane Fluck. Prominer: rule-based protein and gene entity recognition. *BMC bioinformatics*, 6(1):1–9, 2005.

[35] Nathan Harmston, Wendy Filsell, and Michael PH Stumpf. What the papers say: Text mining for genomics and systems biology. *Human genomics*, 5(1):1–13, 2010.

[36] Isabel Segura Bedmar. Application of information extraction techniques to pharmacological domain: extracting drug-drug interactions. *Universidad of Carlos III de Madrid, Computer Science Department*, 2010.

[37] Tianlin Zhang, Jiaxu Leng, and Ying Liu. Deep learning for drug–drug interaction extraction from the literature: a review. *Briefings in bioinformatics*, 21(5):1609–1627, 2020.

[38] Christopher M Biship. Pattern recognition and machine learning (information science and statistics), 2007.

[39] Isabel Segura-Bedmar, Paloma Martínez, and María Herrero-Zazo. Lessons learnt from the ddiextraction-2013 shared task. *Journal of biomedical informatics*, 51:152–164, 2014.

[40] María Herrero-Zazo, Isabel Segura-Bedmar, Paloma Martínez, and Thierry Declerck. The ddi corpus: An annotated corpus with pharmacological substances and drug–drug interactions. *Journal of biomedical informatics*, 46(5):914–920, 2013.

[41] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[42] Yoon Kim. Convolutional neural networks for sentence classification. corr abs/1408.5882 (2014). *arXiv preprint arXiv:1408.5882*, 2014.

[43] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

[44] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*, 2017.

[45] Heike Adel and Hinrich Schütze. Exploring different dimensions of attention for uncertainty detection. *arXiv preprint arXiv:1612.06549*, 2016.

[46] Ying Wen, Weinan Zhang, Rui Luo, and Jun Wang. Learning text representation using recurrent convolutional neural network with highway layers. *arXiv preprint arXiv:1606.06905*, 2016.

[47] Jing Peng, Xinyi Shi, Yiming Sun, Dongye Li, Baohui Liu, Fanjiang Kong, and Xiaohui Yuan. Qtlminer: Qtl database curation by mining tables in literature. *Bioinformatics*, 31(10):1689–1691, 2015.

[48] Antonio Jimeno Yepes and Karin Verspoor. Towards automatic large-scale curation of genomic variation: improving coverage based on supplementary material. *BioLINK SIG*, 2013:39–43, 2013.

[49] Nataliya Le Vine, Claus Horn, Matthew Zeigenfuse, and Mark Rowan. Identifying table structure in documents using conditional generative adversarial networks. *arXiv preprint arXiv:2001.05853*, 2020.

[50] Jiayi Yuan, Hongye Li, Meng Wang, Ruyang Liu, Chuanyou Li, and Beilun Wang. An opencv-based framework for table information extraction. In *2020 IEEE International Conference on Knowledge Graph (ICKG)*, pages 621–628. IEEE, 2020.

[51] Xu Zhong, Elaheh ShafieiBavani, and Antonio Jimeno Yepes. Image-based table recognition: data, model, and evaluation. *arXiv preprint arXiv:1911.10683*, 2019.

[52] Xinyi Zheng, Douglas Burdick, Lucian Popa, Xu Zhong, and Nancy Xin Ru Wang. Global table extractor (gte): A framework for joint table identification and cell structure recognition using visual context. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 697–706, 2021.

[53] Liangcai Gao, Yilun Huang, Hervé Déjean, Jean-Luc Meunier, Qinqin Yan, Yu Fang, Florian Kleber, and Eva Lang. Icdar 2019 competition on table detection and recognition (ctdar). In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1510–1515. IEEE, 2019.

[54] Yufang Hou, Charles Jochim, Martin Gleize, Francesca Bonin, and Debasis Ganguly. Identification of tasks, datasets, evaluation metrics, and numeric scores for scientific leaderboards construction. *arXiv preprint arXiv:1906.09317*, 2019.

[55] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[56] Isabel Segura-Bedmar, Paloma Martínez, and César de Pablo-Sánchez. A linguistic rule-based approach to extract drug-drug interactions from pharmacological documents. In *BMC bioinformatics*, volume 12, pages 1–11. BioMed Central, 2011.

[57] Aditya K Menon, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. Multilabel reductions: what is my loss optimising? In *Advances in Neural Information Processing Systems 32*. NeurIPS, 2019.

[58] Andrew Maxwell, Runzhi Li, Bei Yang, Heng Weng, Aihua Ou, Huixiao Hong, Zhaoxian Zhou, Ping Gong, and Chaoyang Zhang. Deep learning architectures for multi-label classification of intelligent health risk prediction. *BMC bioinformatics*, 18(14):121–131, 2017.

[59] Thamme Gowda and Jonathan May. Finding the optimal vocabulary size for neural machine translation. *arXiv preprint arXiv:2004.02334*, 2020.

[60] Sebastian Raschka. An overview of general performance metrics of binary classifier systems. *arXiv preprint arXiv:1410.5330*, 2014.

[61] Mateusz Buda, Atsuto Maki, and Maciej A Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249–259, 2018.

[62] Bartosz Krawczyk. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4):221–232, 2016.

[63] Yijun Shao, Stephanie Taylor, Nell Marshall, Craig Morioka, and Qing Zeng-Treitler. Clinical text classification with word embedding features vs. bag-of-words features. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2874–2878. IEEE, 2018.

[64] Vithya Yogarajan, Henry Gouk, Tony C Smith, Michael Mayo, and Bernhard Pfahringer. Comparing high dimensional word embeddings trained on medical text to bag-of-words for predicting medical codes. In *ACIIDS 2020*, pages 97–108. Springer, 2020.

[65] Shaza M Abd Elrahman and Ajith Abraham. A review of class imbalance problem. *Journal of Network and Innovative Computing*, 1(2013):332–340, 2013.

[66] Joseph Lemley and Peter Corcoran. Deep learning for consumer devices and services 4—a review of learnable data augmentation strategies for improved training of deep neural networks. *IEEE Consumer Electronics Magazine*, 9(3):55–63, 2020.

[67] Dinghan Shen, Mingzhi Zheng, Yelong Shen, Yanru Qu, and Weizhu Chen. A simple but tough-to-beat data augmentation approach for natural language understanding and generation. *arXiv preprint arXiv:2009.13818*, 2020.

[68] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[69] Li Deng and Yang Liu. *Deep learning in natural language processing.* Springer, 2018.

[70] Ron Artstein. Inter-annotator agreement. In *Handbook of linguistic annotation*, pages 297–313. Springer, 2017.

[71] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens Van Der Maaten. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European conference on computer vision (ECCV)*, pages 181–196, 2018.

[72] Jingbo Zhu, Huizhen Wang, Benjamin K Tsou, and Matthew Ma. Active learning with sampling by uncertainty and density for data annotations. *IEEE Transactions on audio, speech, and language processing*, 18(6):1323–1331, 2009.

[73] Ping Gu, Zhao Ling, Si Yu Shao, and Meng Zhou. Active sample selection through sparse neighborhood for imbalanced datasets. In *2019 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE, 2019.

[74] Mostafa Dehghani, Aliaksei Severyn, Sascha Rothe, and Jaap Kamps. Learning to learn from weak supervision by full supervision. *arXiv preprint arXiv:1711.11383*, 2017.

[75] Yanshan Wang, Sunghwan Sohn, Sijia Liu, Feichen Shen, Liwei Wang, Elizabeth J Atkinson, Shreyasee Amin, and Hongfang Liu. A clinical text classification paradigm using weak supervision and deep representation. *BMC medical informatics and decision making*, 19(1):1–13, 2019.

[76] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[77] Ledell Wu, Fabio Petroni, Martin Josifoski, Sebastian Riedel, and Luke Zettlemoyer. Scalable zero-shot entity linking with dense entity retrieval. *arXiv preprint arXiv:1911.03814*, 2019.