FutureType
Word completion for medical reports

Master Thesis

Author
Lena Brandl

Graduation committee
Dr. Mariët Theune
Human Media Interaction group
Dr. Christin Seifert
Data Management and Biometrics group
Faculty of Electrical Engineering,
Mathematics and
Computer Science
University of Twente
Dr. Thomas Markus
Nedap Healthcare

March 30, 2020

UNIVERSITY
OF TWENTE.

nedap

Contents

Abstract

The current research contributes to the development of **FutureType**, a word completion tool for medical reports, from two perspectives. First, we evaluate the potential of a set of meta features about the patient, author and the report itself to improve FutureType's prediction capacity. Second, we conduct a large scale split test involving the collection of keystroke data from ten customer organizations of Nedap Healthcare, involving 7062 healthcare professionals and spanning 14 and a half weeks to investigate the transferability of instrinsic metrics of language model performance to evaluation with real end users. Our results pave the first steps towards a meta-enriched FutureType as we find distinctive power regarding vocabulary choices for three meta features: the healthcare sector a report originates from, the type of the report and the expertise of the author of the report. The results from the split test advocate a holistic approach to the evaluation of text prediction applications that takes into account both, the system's utility (i.e., the quality of its predictions) and its usability.

1 Introduction

For software companies such as **Nedap Healthcare**, one way of supporting healthcare professionals in their evermore demanding work due to an aging population and the lack of healthcare professionals on the job market is building their software for efficiency and ease of use. To this end, Nedap Healthcare puts effort and resources into developing **FutureType**, a word completion tool for medical record writing. In 2019, Hanekamp, Heesbeen, van der Helm and Valks [1] conducted research on the administrative pressure in long-term care, involving 7700 healthcare professionals in the Netherlands. They found that healthcare professionals spend on average 35% of their time on administrative tasks. This marks a sharp increase compared to 2016 and 2017, when professionals spent 25% of their time on medical documentation and compared to 2018, when administration claimed 31% of their time. Depending on the healthcare sector, professionals spend even more time on administration. In 2019, administrative tasks claimed 40% of working hours in mental health care. Study participants report that they invest most administration time in the electronic health record (EHR). In EHRs, care plans are written out, the intake of medication and the recovery process of patients are documented.

Unstructured text still accounts for the majority of medical documentation even though benefits of structured ontologies are well established [2]. Professionals prefer the ease of using unstructured, natural language and regard structured formats as not doing justice to the complexity of reality [2], [3]. In accordance with Sevenster, Ommering and Qian [4], we expect four main benefits of text prediction, and word completion in particular, in the healthcare domain:

1. Word completion saves keystrokes and thereby reduces the number of misspellings.
2. Word completion reassures the person who is typing that their mental model is aligned with the software, increasing their confidence and user experience.
3. Word completion can be used to explore the system's underlying vocabulary. As a result, it enables and encourages the user to write (medical) terms even if they cannot spell them.
4. Word completion encourages the use of standardized medical vocabulary which enhances the quality and reliability of documentation. High quality documentation also facilitates secondary use of medical data for research purposes and automation of workflows.

Due to the expected benefits of text prediction in healthcare, there has been an increasing interest in research about clinical text prediction in recent years (e.g., [5], [6], [3]). FutureType is not the first experimental text prediction application in a clinical setting. For example, Gong, Hua and Wang [3] developed an auxiliary text prediction interface for patient safety reporting. Patient safety incidents are nowadays often documented in a structured format, including supplementary narrative text fields for detailed information. However, due to work pressure and a lack of knowledge about standardized vocabulary, users often

leave the text fields empty or use inaccurate or incomplete terms and sentences to describe events. Using the text prediction system developed by Gong et al., users wrote more details in the narrative text fields and the quality of the narrative details increased.
In text prediction, suggestions are often generated by an underlying language model that is trained on documents that are similar to the text for which the text prediction model is used. Traditionally, language models predict the next word in a sentence based on a chosen number of preceding words and the letters of the word one is currently typing.
The clinical setting poses unique challenges for text prediction, including

- the usage of complex, medical terms that are often not found in regular language dictionaries,
- the efficient, note-like writing style including many numerical measurements that deviates from natural language grammar,
- the sensitivity of any real medical data that complicates finding suitable training data for text prediction based on machine learning approaches.

Recently, text prediction models have been extended with information beyond the immediate word context. Besides the immediate text context in which a next word suggestion is requested, we can think of other information that is useful for deciding which word is the best candidate for the prediction. For example, the text snippet below could be written in a patient report

> `As discussed with her gynecologist, Miss Doe stopped taking the pill the day before yesterday. Today, she complained about pain in the lower abdomen. I gave her mild painkillers for her < >`

As a human reader, when we predict the missing word in the sentence marked with < >, we take into account that here, `Doe` is a woman. The words marked in red also make our choice for `menstrual cramps` more likely than the `stomach flu`. In general, we would expect the words marked in red to rather occur in a report on a female patient than on a male patient. Extending text prediction models with information beyond the immediate word context exactly attempts to capture the additional value of knowing that `Miss Doe` is female for predicting the next word in the above example. We name models extended with such information `meta-enriched` language models. In the current research, we explore architecture extensions, with which we can infuse FutureType with meta information. In addition, we examine the added value of a set of candidate meta features for FutureType's prediction capacity. Our first research question (RQ1) is formulated as follows:

> **RQ1**: What is the potential of meta information about the patient, the author or the medical report itself to increase FutureType's prediction accuracy for word completions?

A second research interest is about how text prediction models are evaluated and how well common evaluation methods that do not involve user testing align with the demands of real word applications. We name evaluation methods that do not involve real users `intrinsic` evaluation. Methods that do involve real

users we call `extrinsic` metrics. User testing is time-consuming and expensive [7]. Therefore, text prediction models are often evaluated using intrinisc metrics that are common in natural language processing (NLP), such as `perplexity` and `mean reciprocal rank` for suggestion rankings [5]. Another popular form of intrinsic text prediction evaluation is the calculation of theoretical keystroke savings by simulating typing behaviour given an existing text corpus (e.g., [5], [8]). Saving keystrokes is the main objective of text prediction, but in reality, there are other important evaluation criteria for text prediction applications that are neglected by focussing on intrinsic measures. One example is the timing with which word suggestions are presented to a user that interacts with the text prediction system. Indeed, as Nielsen [9] stresses, a system's `usability` and `utility` are two sides of the same coin

> "It matters little that something is easy [to use] if it's not what you want. It's also no good if the system can hypothetically do what you want, but you can't make it happen because the user interface is too difficult."[1]

Our second research question (**RQ2**) is therefore formulated as follows:

> **RQ2**: How does performance as measured by intrinsic text prediction metrics translate to extrinsic measures of performance involving end users?

FutureType, a word completion tool for medical report writing, is under development within the Healthcare branch of the Dutch technology company **Nedap**. The text prediction feature is implemented in the **Ons** software package, an application cluster for healthcare professionals. The following section briefly gives an overview of Ons to facilitate understanding where FutureType is located in the software package and how users interact with the feature.

Nedap Ons

Nedap Ons is a software suite for healthcare professionals consisting of multiple applications, including an electronic health record (EHR)[2]. FutureType is currently implemented for three applications within Ons, the EHR (called Ons Dossier), Ons Agenda and Ons Groepszorg. Ons Agenda is primarily used by healthcare professionals that manage their own client appointments, such as physiotherapists and clinical psychologists, and focuses on making appointments with clients. Ons Groepszorg is a mobile app for registering attendances and absences in group care.
FutureType is implemented in the text fields of Ons applications. The feature provides word suggestions as the user types. FutureType provides one word suggestion at a time. Using the `Tab` key word suggestions can be accepted. Alternatively, as a temporary solution until FutureType is optimized for usage on mobile devices with no physical keyboard attached, word suggestions can also be accepted by clicking on the suggestion. A `thunder symbol` in the upper right corner of each text field can be clicked to toggle FutureType on and off

[1] https://www.nngroup.com/articles/usability-101-introduction-to-usability/, last accessed 2020-03-05

[2] https://nedap-healthcare.com/oplossingen/ons/suite/, last accessed 2020-03-05

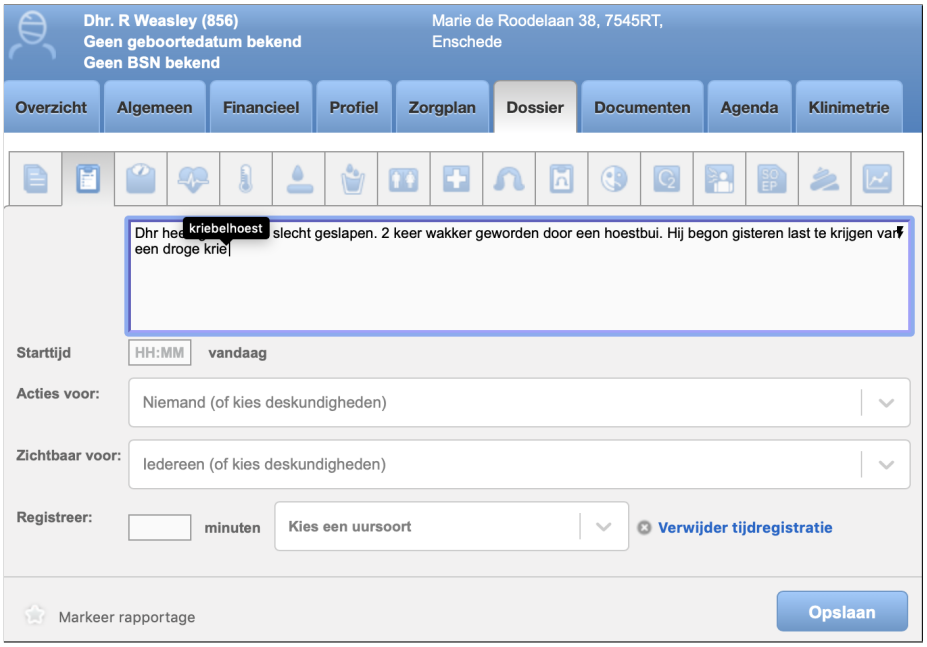


Figure 1: Screenshot of the FutureType word completion feature in Ons Dossier, the electronic health record of the Ons software suite. The word suggestion `kriebelhoest` (Engl. `dry cough`) is shown in a black speech bubble and can be accepted by pressing the `Tab` key. Alternatively, a suggestion can be accepted by clicking on it with the mouse cursor on laptops and desktops, and by hand on mobile devices. FutureType can be enabled/disabled for any text field by clicking the small black thunder symbol in the upper right corner of a text field. The screenshot was taken in Nedap's test environment and does not show any real client data.

for the field. Figure 1 shows an example text field that has been enhanced with the FutureType feature in Dossier.

1.1 Overview

This thesis report is subdivided into two main chapters. **Chapter 2: Towards a meta-enriched FutureType** represents the first pillar of contributions to FutureType's development and examines **RQ1**. We formally present the recurrent neural network (RNN) architecture of the current FutureType model. We then review approaches in scientific literature for adding meta features to recurrent neural language models, which serve as templates for how we can enrich FutureType with meta information in the future. We conclude the chapter with a thorough evaluation of a set of candidate meta features that we extracted from the Ons database, using two methods described in scientific literature: log-odds ratios [10] and the Jensen-Shannon divergence [11]. The meta feature evaluation tests the added value of the chosen set of meta features by investigating the extent to which they can be used to identify characteristic words in medical reports.

Chapter 3: FutureType pilot presents the setup and execution of FutureType's first evaluation with real end users. Evaluating FutureType with its target group, healthcare professionals who write medical reports as part of their daily work activities, represents the second pillar of our contribution. We conduct a large scale A/B test over the course of 14 and a half weeks, among 10 customer organisations of Nedap Healthcare and collecting keystroke data from more than 7000 healthcare professionals. By employing eight FutureType models in the FutureType pilot that vary in their performance on an intrinsic evaluation metric, prediction accuracy, we investigate **RQ2**. In particular, we examine the impact of the internally measurable performance difference on how our users experience FutureType and how they perform when they use FutureType. In addition, the chapter reviews relevant literature on keystroke analysis, our chosen method of data collection for the pilot, and model ablation, which we employ to generate eight versions of FutureType that vary in prediction accuracy. Finally, we discuss our insights from the FutureType pilot and conclude with a set of recommendations for future user evaluations of FutureType.

2 Towards a meta-enriched FutureType

The current chapter examines the extent to which meta information external to the immediate text context can be used to improve FutureType's prediction capacity. First, in section 2.1, we review language model architectures that enable the inclusion of meta information in language models. In section 2.2, we present the current architecture and performance of FutureType, with no meta features implemented yet. As a first step towards enriching FutureType with meta information, we thoroughly evaluate a set of candidate meta features that we retrieved from the **Ons** database. These candidate meta features include

- the gender of the patient about whom the report is written,
- the gender of the employee who wrote the report,
- the healthcare sector from which the report originated,
- the expertise of the author of the report,
- from which care organisation the report originated,
- the type of the report,
- the age of the patient,
- the age of the employee who wrote the report.

Section 2.3 discusses our candidate meta features in more detail. Using log-odds ratios [10] and the Jensen-Shannon divergence (JSD) [11], we examine whether there are differences in word usage depending on our candidate meta features. If we can demonstrate differences in word choice depending on the chosen set of meta features, it is more likely that FutureType will profit from their inclusion in the model.

For the current chapter, it is important to distinguish between FutureType as a complete text prediction system and its individual components: the recurrent neural network (RNN) that yields next word predictions and the Python webservice and Javascript frontend that integrate the model into `Ons`. In this chapter, we refer to the RNN model whenever we use the name `FutureType`. The explanation of the other two components is beyond the scope of the current thesis, though some of the insights we collect may refer to improvements in either one of these components.
We close this chapter by describing the first steps taken towards a meta feature enriched FutureType model.

2.1 Related work: meta-enriched language models

Enriching recurrent neural network (RNN) language models with additional context information has its origins in research that aims to capture long-span dependencies in language models. Modelling long-span features in language models is closely tied to what has become to be known as the `vanishing gradient problem` [12], [13]. The vanishing gradient problem describes the phenomenon that the farther error signals are propagated back in time, the smaller back-propagated errors become until they are reduced to zero. This makes training long-span dependencies in traditional RNNs impossible. Since the discovery of vanishing gradients, several approaches have been proposed to tackle the problem. At the same time, this research contributes to enriching language models

with long-span information beyond a limited word context.

Notably, Mikolov and Zweig [13] refer to a number of successful methods tailored to N-gram language models, including latent semantic analysis (LSA) based approaches [14], [15]. In LSA, long-span history is represented as a vector in latent semantic space. The cosine similarity between a candidate next word and the modelled history can be interpolated with N-gram probabilities for more accurate predictions. While this works well for N-gram language models, it is less suitable for neural network architectures. Therefore, Mikolov and Zweig [13] contribute a RNN model architecture that takes an additional vector **f** as input that represents meta information beyond the immediate word context. **f** is directly connected to the RNN layer and the output layer of the model. Figure 2 shows a schematic overview of the context-enriched RNN model proposed by Mikolov and Zweig.

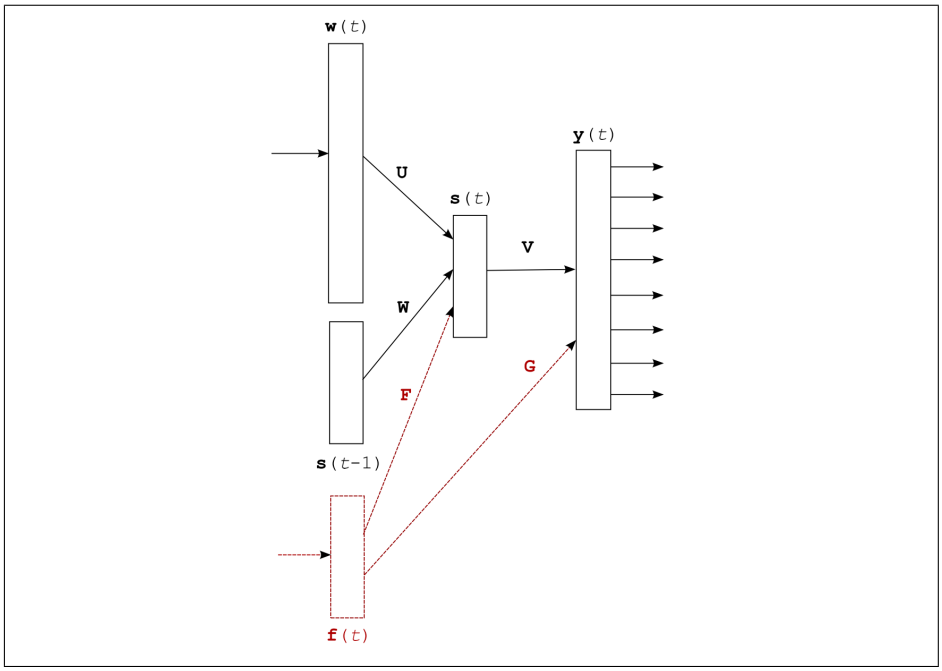


Figure 2: The feature-enriched recurrent neural network (RNN) model architecture proposed by Mikolov and Zweig [13]. In addition to the word context vector **w**(t) and the state at the previous timestep **s**(t-1), the feature vector **f** is connected to the current state layer **s**(t) and the output layer **y**(t), with own corresponding weight matrices **F** and **G**. Adapted from [13].

In initial experiments with this at the time novel RNN architecture, Mikolov and Zweig [13] used Latent Dirichlet Allocation (LDA) to extract topic information from the sentence history and provide the RNN model explicitly with this information using feature vector **f**. In addition, the authors build a custom modification for efficient integration and updating of LDA context vectors depending on the context window at the current time step. At the time of publication (2012), they report a new state-of-the-art perplexity on the Penn

Treebank (PTB) [16] portion of the Wall Street Journal corpus. By using their LDA representations as additional input to the model, data fragmentation can be avoided that is typically associated with the more traditional process of training multiple topic-specific language models.
Mikolov and Zweig further notice that their meta-enriched architecture can also be used to feed meta information that is external to the text to a RNN model. As an example, the authors hypothesize a feature vector **f** that represents habits of a user in voice search. In the case of FutureType, we indeed have a candidate set of meta features at our disposal from other sources than the immediate sentence history. This saves us the effort of extracting relevant meta information from the text itself, as Mikolov and Zweig did using LDA. Section 2.3 describes in more detail which meta features were extracted from the `Ons` database to enrich FutureType with additional context information.

From ConcatCell to FactorCell

The context-enriched RNN architecture Mikolov and Zweig [13] originally proposed in 2012 has inspired recurrent neural network adaptations such as language model personalization [17] and taking genres into account in a multi-genre broadcast speech transcription task [18]. The architecture has recently been named **ConcatCell** by Jaech and Ostendorf [19], [20]. The authors [19] show mathematically that adding a context embedding to the recurrent layer via concatenation boils down to using a context-adjusted bias at the recurrent layer, like so

$$\begin{aligned} h_t &= \sigma(\hat{W}[w_t, h_{t-1}, c] + b) \\ &= \sigma(W[w_t, h_{t-1}] + Vc + b) \\ &= \sigma(W[w_t, h_{t-1}] + b^{'}) \end{aligned} \tag{1}$$

Where h_t is the current hidden state and w_t a word embedding. $\hat{W} = [W\,V]$ is the weight matrix that transforms the concatenation of the hidden state from the previous time step h_{t-1}, w_t and the context representation c to produce h_t. Here, Vc is equivalent to Mikolov's and Zweig's feature vector **f** and its corresponding weight matrix **F** that connects the vector to the recurrent layer. Note that the above formula only holds for context embeddings that are constant for all time steps of an input sequence.

Instead of using context as an additional input, Jaech and Ostendorf [19], [20] propose an architecture where context is used to adapt the recurrent layer weight matrix. The idea is that increasing the direct influence of context information on the model parameters produces models that are more responsive and adapted to context. Mathematically, the authors extend the `ConcatCell` architecture by introducing a context-dependent weight matrix $W^{'} = W + A$. `ConcatCell` uses a single weight matrix W that is shared across all context settings. A is an adaptation matrix that is generated by taking the product of the context embedding vector c and a set of left and right basis tensors that together produce a rank matrix **r**. Given that the context representation has a dimensionality of k, the word embedding w of e and the recurrent hidden states of d, we can describe the dimensions of the left and right base tensors Z_L and Z_R, like so

$$
\begin{aligned}
Z_L &\in \mathbb{R}^{k \times (e+d) \times r} \\
Z_R &\in \mathbb{R}^{r \times d \times k}
\end{aligned}
\tag{2}
$$

Together, the two base tensors hold k different rank r matrices, each of the size of W. A is generated like this

$$
A = (c \times_1 Z_L)(Z_R \times_3 c) \tag{3}
$$

where $\times_i$ denotes with which dimension of the tensor the product is taken. For both base tensors, this is the dimension that matches with k, the dimensionality of the context embedding.
Taken together, left and right base tensors can be used as a factor to transform the context embedding c to adapt the recurrent weight matrix, like so

$$
\begin{aligned}
h_t &= \sigma(W^{'}[w_t, h_{t-1}] + b^{'}) \\
W^{'} &= W + (c \times_1 Z_L)(Z_R \times_3 c^T) \\
b^{'} &= Vc + b
\end{aligned}
\tag{4}
$$

During model training, rank **r** is treated as an additional hyperparameter and controls the extent to which the generic weight matrix W is adapted with context information. Jaech and Ostendorf call this the **FactorCell** model because they adapt the recurrent weight matrix with a factored component. The authors note that if the context is known in advance, $W^{'}$ can be pre-computed which means that despite having many more parameters than the simpler `ConcatCell` model, computational cost at runtime is comparable. They also note that the `ConcatCell` model is a special case of the `FactorCell`, namely when Z_L and Z_R are both set to zero.

Jaech and Ostendorf [19] thoroughly evaluate their `FactorCell` architecture in direct comparison to Mikolov's and Zweig's `ConcatCell` model and a third, popular context-based adaptation to the softmax output, called the `SoftmaxBias` (e.g, [21], [22]). The authors note that the latter is a simplification of the `ConcatCell` model which in turn is a simplification of the `FactorCell` model. The three methods are compared on four publicly available word-level and two character-level data sets. The `FactorCell` model is on par or outperforms alternative methods in both, perplexity and text classification accuracy, for all six tasks. As long as contexts are known in advance, the benefits of the `FactorCell` model come with no additional computational costs at test time, since its transformations can be pre-computed.

Recent advances in context-enriched language models

Besides Jeach's and Ostendorf's `FactorCell` extension to Mikolov's and Zweig's influential `ConcatCell` architecture, researchers have explored a variety of methods for generating a contextual representation that can be fed into neural language models (e.g., [23], [24]). One notable recent contribution by Zheng, Chen,

Huang, Liu, and Zhu employs a `trait fusion module` to embed persona representations in a personalized dialogue response task. The authors combine explicitly represented "personality traits", namely speaker `age`, `gender` and `location`, using one of three methods: `traits attention`, `traits average`, and `traits concatenation`. `Traits attention` merges all traits into a context vector v_p using an attention mechanism that is based on the previous hidden state and an attention weight $a^{'}$ that is computed during model training for each trait. v_p is then obtained as a weighted sum of the individual trait representations, like so

$$v_p = \sum_{i=1}^{N} a_i^{'} v_{t_i} \tag{5}$$

where v_{t_i} denotes the trait embedding representation of trait t_i. Since the traits considered in the study by Zheng et al. [23] are all single-valued, the authors used simple look-up tables for trait encoding. The second trait fusion method, `traits average` is a special case of `traits attention` where all trait representation are weighted equally. The final trait fusion method, `traits concatenation` simply concatenates the single-valued trait representations into one context vector, with no additional attention mechanism.

Zheng et al. call the personality context vector v_p "persona representation". The authors implement and evaluate two methods for incorporating v_p into a sequence to sequence model: `Persona-aware attention` (PAA) and `persona-aware bias` (PAB). Ultimately, the two persona decoding methods implemented by Zheng et al. [23] mirror two earlier discussed popular context-based adaptation methods. PAB boils down to a `SoftmaxBias` approach, while PAA is similar to the `FactorCell` model.

What started as an approach to tackling the `vanishing gradient problem` while preventing data fragmentation by training multiple smaller language models has developed into a research discipline of its own: meta-enriched RNN language models. FutureType is a RNN language model that is likely to profit from meta information because of its specific medical vocabulary.
The reviewed literature on enriching RNN language models with meta information reports performance improvements above models that lack additional information beyond the immediate text context. We reviewed three popular approaches to incorporating meta information into RNN language models: the `ConcatCell` model, the `FactorCell model` and the popular `SoftmaxBias` approach. However, none of the reviewed research explains their motivation for including the specific meta information they selected. Before deciding on a model architecture to implement, we need to make a well-informed selection of candidate meta features. It is unclear how researchers in the reviewed literature distinguished between candidate meta features that are likely to benefit model performance from those that will only clutter the parameter space.
Therefore, as an initial step towards enriching FutureType with meta information, the remainder of this chapter explores the potential of the meta features available in Ons for improving FutureType's text prediction capacity. To this end, we first introduce the current model architecture and performance of Fu-

tureType, with no meta features implemented. Then, we conduct an elaborate meta feature evaluation using log-odds ratios and Jensen-Shannon divergence (JSD) scores to explore differences in the vocabulary of medical documents, depending on variables such as the type of the report or the gender of the patient.

2.2 Baseline FutureType

Architecture

In a nutshell, the text prediction model named **FutureType** takes a sequence of eight words as input and predicts a single word as output. In the process, words are represented as embedding vectors of size 300. As Yin and Shen [25] note, 300 is the most commonly used ad-hoc dimensionality for embeddings. In his article on empirical observations on word embeddings, Arora [26] also discusses:

> "A striking finding in empirical work on word embeddings is that there is a sweet spot for the dimensionality of word vectors: neither too small, nor too large."[3]

To reveal why the rule of thumb dimensionality of 300 works well for many settings, Yin and Shen [25] developed a method based on mathematical theory as well as empirical results to determine the optimal dimensionality for word embeddings, depending on the corpus they are trained on. Their method aims at finding the optimal dimensionality that minimizes the `Pairwise Inner Product (PIP) loss`, which they describe as a dissimilarity metric between two word embeddings. Yin and Shen validate their theoretical accounts on the Text8 corpus [27]. We leave finding the optimal dimensionality for our corpus to future work and adapt the empirically well-proven dimensionality of 300 for our current word embeddings.

Word embedding vectors were not trained during model training, but generated using the *fastText*[4] library [28]–[30]. The word embeddings were pre-trained on the same corpus of 308 million medical reports that we later used for training the FutureType model. Embeddings were trained for 25 epochs using default parameters tailored to the `Skip-gram` model for learning word representations, with the exception that we created a vector representation for all tokens in the vocabulary, including punctuation and misspelled words. Using the default settings, only words with a minimum corpus frequency of 5 are represented as vectors. In total, we trained 1 555 341 `Skip-gram` word vectors. We chose the `Skip-gram` architecture since it yields better representations of rare words [31], which we expected to deal with in our medical corpus. During model prediction, out-of-vocabulary words were assigned a special word embedding, consisting of only zeros.

Regarding architecture, the FutureType model consists of a single bidirectional long short-term memory network (LSTM) layer [32], allowing simultaneous forward and backwards processing of the input sequence. The bidirectional LSTM layer has 2×100 nodes with hyperbolic tangent activation. The dense output

[3] https://www.offconvex.org/2016/02/14/word-embeddings-2/, last accessed 2020-28-02
[4] https://fasttext.cc, last accessed 20-02-2020

layer had 39 074 nodes with softmax activation, each output node corresponding to a word in the output vocabulary. Figure 3 shows a schematic overview of the baseline FutureType model.

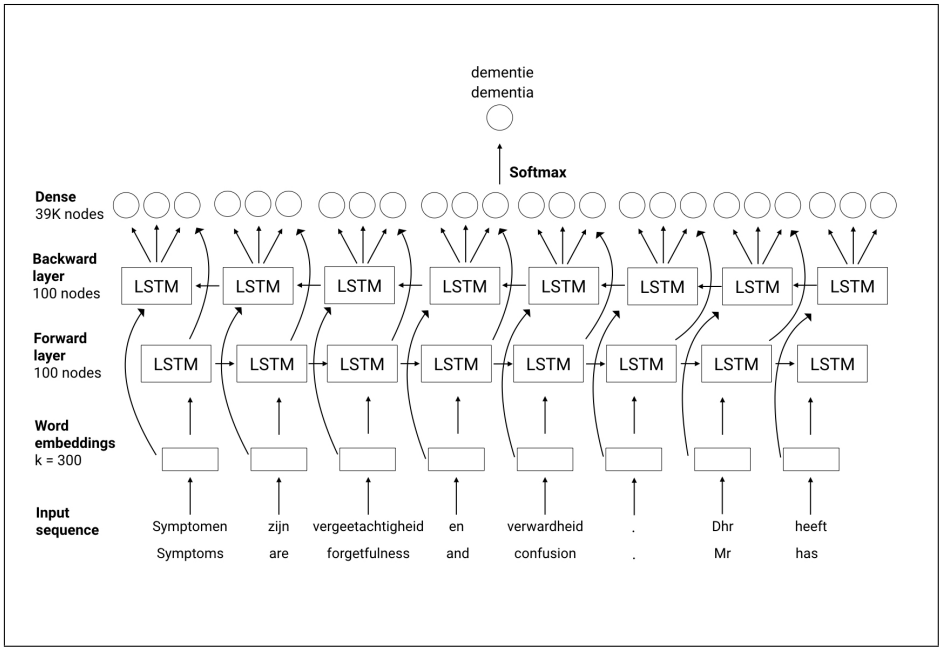


Figure 3: Architecture of the baseline FutureType model. The model takes an input sequence of eight words and predicts a single next word given the input sequence. Word embeddings had a dimensionality `k` of 300. The bidirectional LSTM layer had 2×100 nodes and the dense output layer had 39 074 nodes.

Data

We randomly sampled 1 000 000 documents from the database of Nedap Healthcare to train the FutureType model. Such a document was an entry in a real medical report written by employees using the `Ons` software. Since the baseline FutureType model takes sequences of eight words as input to predict a ninth word, we randomly sampled 20 million sequences of length 9 from the corpus of 1 million documents. A sequence could be any nine consecutive words in any of the sampled documents. Specifically, we sampled sequences by randomly determining 20 million target words that needed to be predicted and then retrieving their preceding eight words, including punctuation. We padded sequences at the beginning of a document so that we always sampled sequences of length 9, even though the target word may not have been preceded by eight words in the document. The 20 million sampled sequences were split into training, validation and test set with a 99.5%/0.25%/0.25% split, where 99.5% of all sequences were assigned to the training set, and 0.25% respectively to validation and test. As discussed in detail in section 2.2, additional filtering was performed to ensure that the sampled sequences did not include misspelled words as prediction targets. The filtering reduced the final training, validation, and test sets to

3 845 683, 9523 and 9757 sampled sequences respectively. Table 1 summarizes descriptives for the dataset used to train and evaluate the FutureType baseline.

Table 1: Descriptives of the 20M sampled sequences, each consisting of 9 words, from a random sample of 1 000 000 medical reports managed by Nedap Healthcare. Descriptives were calculated after filtering, which reduced the final number of sampled training, validation and test sequences. Filtering and only affected the target word, which means that sampled word contexts, that is the model input, could contain words with less than six letters, even if they did not contain special characters such as a trema (*ä*).

	Training	Validation	Test
N samples	3.845M	9523	9757
N words	34.611M	85 707	87 813
Median length all words	5	5	5
Median length target words	8	8	8
Vocab size (tf 1*)	146 961	8622	8843
Vocab size (tf 2)	111 716	3642	3711
Vocab size (tf 5)	57 356	1541	1608
Vocab size (tf 10)	38 863	868	887
Vocab size (tf 20)	26 040	463	465
Vocab size (tf 100)	9928	101	105

* `tf` stands for `minimum term frequency` and means that all tokens included in the vocabulary occurred at least `n` times. For `tf 1`, all tokens in the corpus are counted.

Model training

The training pipeline for FutureType contained a number of preprocessing steps. Most notably, the size of the output vocabulary was reduced by filtering based on word frequencies and a customized spelling correction algorithm. Reducing the size of the output layer has a number of advantages, including a faster inference time in real application settings and a significant reduction in training parameters, which speeds up model training. Frequency filtering also reduced the chance of predicting privacy sensitive words, such as names. Spelling correction was targeted at preventing FutureType from suggesting misspelled words.

Output vocabulary reduction

As a first step towards optimizing the output vocabulary, we chose to only predict words that occurred more than 1000 times in the training set. Second, all words shorter than 6 characters were removed. In terms of keystroke savings, most can be gained by predicting long words. Next, Dutch first and last names were filtered. Two public data sets [33] of the 10 000 most frequent Dutch first and last names were used for this purpose.

Finally, a custom spellcheck was performed. Spelling correction was based on dictionary look-ups and (frequency based) heuristics. Unfortunately, we failed

to actually apply the spelling correction as to the output layer vocabulary of FutureType due to a bug. However, we did identify spelling mistakes using the algorithm and will apply the spelling correction to future re-trained FutureTyoe models.
At the beginning of the spellcheck, misspelled words are identified by checking the corpus against the Hunspell dictionaries for the Dutch language[5].
For words that did not exist in the dictionary, we generated a spelling update. Updating misspelled words included finding close matches in the corpus using the Python library `difflib`[6] for comparing two sequences. `Difflib` sequence matching is based on the Ratcliff/Obershelp pattern recognition algorithm [34], [35]. The minimum similarity had to be 0.8 out of 1. We executed a number of additional checks to find the best of all close matches, including

- The first two characters of a misspelled word and any close match identified by the `difflib` had to be the same
- The candidate match's prevalence in the corpus is higher
- The misspelled word had a minimum length of 4 letters

If all of the above conditions were satisfied, we checked whether the conversion from misspelled to correctly spelled match was a matter of adding accents (e.g., $\hat{a}$, $\ddot{a}$, $\grave{a}$). If it was, we added the accents. Otherwise, we checked whether a single letter transformation was possible based on the Levenshtein distance metric [36]. If it was possible, the candidate was chosen as correct match.
If the algorithm failed for all close matches we extracted using `difflib`, the misspelled word was simply removed from the vocabulary, without a substitute. The spellcheck algorithm was combined with three other manual filtering conditions so that all words that did not satisfy minimally one of the following additional conditions were also removed from the output vocabulary.

- The word consists of minimally 7 characters so that long words are kept in the vocabulary.
- The word contains tremas (¨) or other special characters. Words that do contain special signs were considered the less frequent, but correct variant of words that did not pass the `String` length filter.
- The word was matched to a spelling mistake as the correct form of the word[7].

The snippet below shows misspellings of the word `medicatie` (Engl. `medication`) that our custom spelling correction mapped to the correct spelling of the word.

[5]http://hunspell.github.io
Downloaded from https://github.com/elastic/hunspell/tree/master/dicts/nl_NL, last accessed 25-03-2020

[6]https://docs.python.org/3/library/difflib.html, last accessed 25-03-2020

[7]While reviewing the code, we came across a noteworthy bug in the implementation of these last filtering steps. The bug is related to how the filtering logic was applied (a chain of `OR` operators). Unfortunately, we failed to only filter spellings which were earlier identified as correct which led to the inclusion of misspelled words in the output layer despite our efforts to identify misspellings. Since we already removed words that were shorter than six letters at the very beginning of the output layer filtering pipeline, they all had a length of minimally six letters.

```
mediciatie ( 2773 ) => medicatie ( 22392278 )
mediacatie ( 1387 ) => medicatie ( 22392278 )
medicaie   ( 1825 ) => medicatie ( 22392278 )
medicstie  ( 3548 ) => medicatie ( 22392278 )
medicarie  ( 1327 ) => medicatie ( 22392278 )
```

Listing 1: Spelling mistakes of the word `medicatie` (Engl. `medication`) that were caught by our custom spelling correction and mapped to the correct spelling of the word. The frequency counts in the training set of each misspelling and the correct variant of the word are included in brackets.

Note that FutureType does not perform spelling correction in real-time while the user types. Spelling correction was performed for model training only.
The applied filtering strategies and (attempted) spelling correction shrank the output vocabulary size from 1 555 342 to 39 074 words. The spelling correction was "attempted" because a bug in the filtering logic we applied led to the accidental inclusion of misspelled words, despite all our previous and very fruitful efforts to identify misspellings. In the end, the filtering caused words that were shorter than six letters **and** did not contain a special character like a trema (¨) to be excluded from the output layer. This reduced the final training, validation and test samples to 3 845 683, 9523 and 9757 sampled sequences respectively. We thus tried to exclude misspellings from the output layer **and** the set of possible training, validation and test target words.

As further explanation for our decision, at first glance, excluding misspellings may sound dangerously as if we had intended to overestimate prediction performance during intrinsic evaluation. The logic behind this decision was that we did not want to punish our model for predicting correct words when the model was trained on data containing misspelled variants of the same word. Above all, we did not want the model to learn frequent misspellings, such as `client` instead of the correct `cliënt` in Dutch. Spellchecking was only intended for the output vocabulary, misspellings did have a word embedding representation and could be fed to the model as input. In the future, the model will be re-trained with the spellchecking described above and the bug removed.

Training with default parameters
The baseline FutureType model was trained for 20 epochs with default parameters for LSTM models. We used a batch size of 256, categorical crossentropy as loss and Adam as optimizer. We reduced the base learning rate of 0.001 whenever validation accuracy did not improve for 3 consecutive epochs with a factor of 0.1. We stopped early after 5 epochs if no improvement in validation accuracy took place. A dropout of 0.4 was added between the word embedding input and the LSTM layer. Between the LSTM output and the dense output layer a dropout of 0.2 was added. No systematic hyperparameter optimization was conducted for the baseline model.
Table 2 documents the accuracy and perplexity the trained model achieved on 1000 randomly sampled sequences from the validation and test set. Accuracy and perplexity were calculated for varying prefix lengths which mimics how users interact with word completion. A user may not accept a suggestion right away, but continue typing. Table 2 shows that there is a huge increase in both, prediction accuracy and perplexity, when three instead of two letters of the target word are known at the moment of prediction. With each additional known letter, the

set of possible predictions becomes smaller and predictions become more accurate. On the basis of Table 2, we see that from five letters onward the increase in performance flattens. It is likely that the set of possible predictions is already small when the first five letters of the target word are known which explains why typing additional letters does not increase the model's performance.

Table 2: Prediction accuracy (ACC) and perplexity (PPL) on 1000 randomly sampled sequences from the validation and test set at different prefix lengths. A prefix length of 2 means that the prediction was performed when two letters of the target word were known.

Prefix length	Validation		Test	
	ACC	PPL	ACC	PPL
2	0.583	814.37	0.574	915.82
3	0.710	292.03	0.713	297.07
4	0.810	121.11	0.794	131.81
5	0.874	60.23	0.858	67.75
6	0.912	35.03	0.903	39.28
7	0.941	22.58	0.927	25.60

2.3 Meta feature evaluation

On the basis of earlier work [17], [23], we expect that document meta information, such as the `gender` of the patient about whom a report is written, carries useful information for word prediction tasks. In this section, this assumption is tested using the same random sample of 1 million medical records that was used to train the baseline FutureType model.
The extent to which the selected meta information can distinguish the usage of words in our data set is examined in two ways. For both methods, we treat the set of reports that belongs to each level of a meta feature as an individual text corpus. For example, for the `employee gender` feature, texts written by female employees are merged into one text corpus and reports written by male authors into another corpus. Our first method calculates weighted log-odds ratios for each word in the combined corpus of two (or multiple) feature levels to identify words that are characteristic for their respective vocabularies. Using weighted log-odds ratios to identify distinctive words was proposed by Monroe, Colaresi and Quinn [10] (Section 2.3.1). The second method measures the similarity between two or more word probability distributions by calculating the Jensen-Shannon divergence (JSD) [11] between these distributions (Section 2.3.2).

Before going into the details of weighted log-odds ratios and the JSD, we provide a brief description of our meta data. Tables 3 and 4 descriptives for our candidate meta features. Our candidate features include

- Patient gender
- Employee gender
- Healthcare sector
- Employee expertise
- Healthcare organisation
- Report type
- Patient age
- Employee age

The meta data descriptives are spread across several tables because descriptives for categorical as well as numerical features were difficult to combine in one clear table. The same was true for categorical meta features with just a few feature levels, such as `patient gender`, and features with hundreds of levels, such as `employee expertise`. We summarized categorical features with few levels in Table 3 and features with a large number of levels in Table 4. For categorical meta features, we summarize the number of distinct groups, their frequency and the number of missing values. For the two continuous variables `patient age` and `employee age`, the median and standard deviation are reported in Table 5, as well as the number of missing values.

Some descriptive trends visible in Table 3.1 have the same underlying reason. More than 99% of the sampled reports originate from `elderly care`, indicating that the `healthcare sector` feature is extremely imbalanced. Only 0.25% and 0.17% respectiveley originate from the other two sectors, `mental and disabled care` and `regional protected living` (`RPL`). This extreme class imbalance correlates with the distribution of sectors across all customers of Nedap Health-

care. Ons is indeed mainly used by care organizations working in elderly care. This is also reflected in the median patient age of 85.
Table 3 further documents an extreme class imbalance for the employee gender feature, with more than 86% of all employees being female. The extreme class imbalance for the distribution of report types can be explained by how the medical dossier application is used by end users. The Text report type serves the needs of most everyday reports. The rest of the types is for documenting specific information which is reported less frequently. For instance, the medical type is intended for doctors to write down medical content.

The employee expertise feature has a large number of feature levels. This is because customer organizations of Nedap Healthcare define their own expertise titles. As a result, several expertise titles in the data set describe the same expertise. For instance, the top three most occurring expertise titles are Verzorgende IG, Niveau 3 and 3 VIG. They all describe the same organizational function, but they are called and written differently by different customer organizations, yielding distinct expertise entries in our data set. Normalizing expertises was beyond the scope of this thesis.
In this section, the expertise feature is evaluated using a number of selected expertise clusters. Clusters were formed by selecting expertise Strings naively based on substrings. For instance, the doctor cluster contained medical reports written by employees whose expertise title contained the substring arts, either with a capital or lowercase letter. Table 6 summarizes which expertise clusters were examined and the substrings used to identify them. Using substrings for forming expertise clusters has its drawbacks. The main disadvantage is that the precision and recall with which this simple method identifies expertise titles that belong to a certain expertise cluster is difficult to evaluate. In addition, the method neglects the subtle variety of expertises that contain a certain substring. For example, Klinisch psycholoog and Psychosociaal medewerker are both included in the psychologist cluster, although it can be expected that they have different responsibilities and therefore, they are likely to report on different matters, using different vocabulary. Another drawback is that the same expertise may be included in two or more clusters because it contains more than one of the characteristic substrings, as in the case of Sociaal psychiatrisch verpleegkundige. Nevertheless, the simplicity of this filtering approach speaks for itself and the obtained expertise clusters were deemed sufficient for the purpose at hand. The clusters were chosen based on prevalence in the data set and assumptions about differences in language between groups. For instance, most employees in our data set have a background in nursing. We expected their language to differ from doctors and psychologists because they report on different subjects.

The two age features were the only non-categorical features and were transformed into age groups for subsequent analyses. Earlier research has shown that it is impractical to work with exact ages in predictive tasks based on text data [23], [37]. For client age, three age groups were formed. The first contained reports on clients aged between 0 and 30 years, the second between 31 and 60 years, and the last contained reports written on clients older than 60 years up to the filtering limit of 114 years. The rationale for these age groups was to form

three groups that span comparable age ranges and capture individuals that, on the basis of their age, are likely to have similar health problems. For example, it can be expected that pregnancy related health problems are unlikely to occur in the age groups 0 to 30 and 60 to 114, but we expect them to be most prevalent in the middle-aged group. For `employee age`, three age groups were formed: "young professionals" including reports written by employees between the age of 12 and 30, "professionals" aged 31 to 50, and "senior professionals" aged 51 or older up to the filtering limit of 99 years. We formed age groups on the basis of simple heuristics and assumptions and their validity should be checked in the future. Table 5 summarizes descriptives for each age group.

Table 3: Descriptive summary of categorical meta features with maximal three feature levels in our sampled data set of 1 000 000 medical documents stored in the Ons database.

Feature	**Feature levels**	**Frequency (%)**	**Missing (%)**
Patient gender	Female	65.10	.02
	Male	34.75	
Employee gender	Female	86.38	7.51
	Male	6.12	
Health sector	elderly care	99.58	.00
	mental and disabled care	.25	
	regional protected living	.17	

Table 4: Descriptive summary of meta features with more than three feature levels in our sampled data set of 1 000 000 medical documents stored in the Ons database.

Feature	**Feature levels**	**Top 3**	**Missing (%)**
Employee expertise	3758*	Verzorgende IG (48.7K) Niveau 3 (38K) 3 VIG (33.4K)	45.5K (45.57%)
Care organisation	813	54.2K (5.42%) 32.1K (3.22%) 29.9K (2.99%)	.00
Report type	18**	Text (868.1K) (86.81%) Medical (44.7K) (4.47%) Defecation (23.2K) (2.33%)	.00

*Each customer organisation of Nedap Healthcare may define their own set of expertise titles. The reported expertises in the data set were not normalized, meaning that the same expertise is likely to occur several times in the data set under multiple titles. Normalizing or clustering `expertises` was beyond the scope of this thesis.

**To be precisely, 29 unique report types occur in the sampled data set (N = 1 000 000). There are 18 official `report types` in use nowadays. In the early days of the application, there were no clear guidelines regarding the usage of report types. This led to the manual addition of (sometimes redundant) type codes in the database. They occur infrequently (n = 1284) and can be regarded as residual artifacts.

Table 5: Descriptives for the `employee age` and `client age` meta features.

Feature	**Groups**	**Median**	**SD**	**Missing (%)**
Patient age*		85	18.39	.13
	0-30 (N = 44K)	20	7.95	
	31-60 (N = 79K)	51	8.59	
	61-114 (N = 876K)	86	8.77	
Employee age*		47	13.05	14.96
	12-30 (N = 178K)	26	3.16	
	31-50 (N = 324K)	42	6.10	
	51-99 (N = 348K)	57	4.27	

*Calculated based on filtered data sets (N = 998 695 for `patient age` and N = 850 393 for `employee age`) that excluded missing values and extreme outliers reporting ages larger than 115 for `patient age` and 100 `employee age`. For `employee age`, an additional lower bound was set to 12. Reported ages beyond these bounds were believed to represent artifacts. 12 appeared to be a reasonable lower bound for young interns to be included in the sample.

Table 6: Summary of expertise clusters in meta feature evaluation, based on a random sample of 1 000 000 medical reports. `N unique titles` counts the number of unique expertise titles in the dataset that contain the respective substring.

Expertise	**(Sub)string**	**N**	**N unique titles**	**Example**
Doctor	`arts`, `Arts`	5K	56	`Tandarts`, `7a.Huisarts`, `(Huis)arts`
Psychologist	`psy`, `Psy`	2.5K	62	`GZ-psycholoog`, `Psychomotorische therapie`, `Sociaal psychiatrisch verpleegkundige`
Daycare	`dagb`, `Dag`	3.4K	46	`Assistent begeleider wonen en dagbesteding`, `Medewerker Dagbesteding`, `Medewerker dagbehandeling`
Nurse	`verpl`, `Verpl`*	124.4K	504	`Wijkverpleegkundige`, `Coördinerend Verpleegkundige`, `3.Verpl/Verz`

*Even though the word `niveau` is often used in expertise titles for nurses it was not used as a substring because it occurs in many other expertise clusters as well, as in `Logopedist niveau 5`.

Next, we will describe the two methods we used to examine the potential value of our meta features for word prediction tasks. Using weighted log-odds ratios and the Jensen-Shannon divergence, we explore the extent to which our chosen meta features can identify differences in word usage.

2.3.1 Log-odds ratios

Monroe, Colaresi and Quinn [10] summarize a variety of techniques for visualizing the extent to which words (or other lexical features) are used differently across pairs or sets of documents. Word visualizations and lists are common in textual analyses because they offer semantic validity to automated text analysis as they intuitively show whether the employed technique captures some expected substantive meaning. If it does, the visualizations reflect word selections or a word-specific measure that characterize some semantic difference across groups, such as topics or ideology. The selection of words or the word-specific measure can also serve as input to some feed forward analysis, for example, training a classifier for unlabeled documents. The summarized techniques range from plotting word frequencies to model-based approaches that model the choice of words as a function of the group a piece of text originates from. In the process, the authors discuss the shortcomings of the reviewed techniques. Most of them, despite being popular in journalism, political science and other disciplines, fail to account for sampling variation and are prone to overfitting ideosyncratic differences between groups.

One of the two model-based techniques that Monroe et al. [10] favour are weighted log-odds ratios using an informative Dirichlet prior for regularization. The other technique uses a Laplace prior for tackling the problem of overfitting. Monroe et al. [10] model the occurence of all words in a corpus $\mathbf{y}$ with π

$$\mathbf{y} \sim \text{Multinominal}(n, \pi) \tag{6}$$

where $\mathbf{y}$ represents the raw counts in the entire corpus with $n = \sum_{w=1}^{W} y_w$ and π being a `W`-vector of multinominal probabilities. Using the multinomial logit transformation and w = 1 as reference and adopting the convention that $\beta_1 = 0$, we transform multinominal probabilities into log-odds with

$$\beta_w = \log(\pi_w) - \log(\pi_1), \text{ w} = 1, ..., \text{ W} \tag{7}$$

Equation 8 allows us to transform β estimates back to multinominal probabilities.

$$\pi_w = \frac{exp(\beta_w)}{\sum_{j=1}^{W} exp(\beta_j)} \tag{8}$$

Under π, Monroe et al. define the likelihood function `L` as

$$L(\beta|y) = \prod_{w=1}^{W} (\frac{exp(\beta_w)}{\sum_{j=1}^{W} exp(\beta_j)})^{y_w}. \tag{9}$$

`L` describes the likelihood of the odds for all words given our entire corpus `y`. Note that Monroe et al. simplified the likelihood function by ommitting the standard normalization factor $\frac{n!}{\prod_{w=1}^{w} y_w!}$ for multinomial distributions. The described likelihood is thereby no longer guaranteed to be an actual probability distribution as the individual likelihoods do not necessarily add up to 1. However, since the scaling factor is entirely based on the observed `y`, the likelihood ratios between individual words `w` are not affected. Supposedly, Monroe et al.

ommitted it because the authors were solely interested in the ratio between word likelihoods under the specified model.
The respective log-likelihood function `l` is

$$l(\beta|y) = \sum_{w=1}^{W} y_w \log(\frac{exp(\beta_w)}{\sum_{j=1}^{W} exp(\beta_j)}) \tag{10}$$

Within a topic `k`, group partitions are made salient using subscripts. In our case, we simplify `k` to be a single constant topic (i.e., a medical report). In theory, the medical reports could be further subpartitioned into topics, such as `activities of daily living` or `morning report`, or on the basis of the 18 `report types` that now form a meta feature of their own.

$$y_k^{(i)} \sim \text{Multinominal}(n_k^{(i)}, \pi_k^{(i)}) \tag{11}$$

Since we do not take topic partitions into account in the current research, the `k` index will be ommitted in the remainder of this chapter.

Due to the lack of covariates, the maximum likelihood estimation (MLE) for $\beta_w^{(i)}$ boils down to

$$\hat{\pi}^{\text{MLE}} = \text{y} \cdot (\frac{1}{n}) \tag{12}$$

and using the logit transform respectively

$$\begin{aligned} \beta_w^{MLE} &= \log(\pi_w^{MLE}) - \log(\pi_1^{MLE}) && (13) \\ &= \log(y_w) - \log(y_1), \text{ w} = 1, ..., \text{W}. && (14) \end{aligned}$$

Again, w = 1 serves as reference and $\beta_1 = 0$ is assumed.

Dirichlet prior

In Bayesian statistics, the prior probability distribution, or prior for short, expresses one's beliefs about a quantity before some evidence is taken into account. The conjugate prior of the multinominal distribution is the Dirichlet.

$$\pi \sim \text{Dirichlet}(\alpha) \tag{15}$$

where α is a vector with each $\alpha_w > 0$. α_w directly affects the posterior probability of `w` as if an additional $\alpha_w - 1$ instances of `w` were observed in the data. With the Dirichlet prior, the estimate becomes

$$\hat{\pi} = (y + \alpha) \cdot \frac{1}{(n + \alpha_0)} \tag{16}$$

where $\alpha_0 = \sum_{w=1}^{W} \alpha_w$.

Feature evaluation

Finally, within a topic `k`, we are interested in how the word usage of a word `w` by group `i` differs from the word usage by all groups, or a specific group `j`. This is captured by the log-odds ratio, which we define as

$$\delta_w^{(i)} = log(\Omega_w^{(i)}/\Omega_w) \tag{17}$$

where $\Omega_w = \pi_w/(1-\pi_w)$ denotes the probabilistic odds of word `w` relative to all other words under the multinominal model π. The point estimate for this using the appropriate subscripts is

$$\hat{\delta}_w^{(i)} = \log[\frac{(y_w^{(i)} + \alpha_w^{(i)})}{(n^{(i)} + \alpha_0^{(i)} - y_w^{(i)} - \alpha_w^{(i)})}] - \log[\frac{(y_w + \alpha_w)}{(n + \alpha_0 - y_w^{(i)} - \alpha_w^{(i)})}] \tag{18}$$

where

$$\alpha_w^{(i)} = \alpha_0^{(i)}\hat{\pi}^{\text{MLE}} = \text{y} \cdot \frac{\alpha_0}{n} \tag{19}$$

For two specific groups `i` and `j`, the point estimate is respectively given by

$$\hat{\delta}_w^{(i-j)} = \log[\frac{(y_w^{(i)} + \alpha_w^{(i)})}{(n^{(i)} + \alpha_0^{(i)} - y_w^{(i)} - \alpha_w^{(i)})}] - \log[\frac{(y_w^{(j)} + \alpha_w^{(j)})}{(n^{(j)} + \alpha_0^{(j)} - y_w^{(j)} - \alpha_w^{(j)})}] \tag{20}$$

A large positive value indicates that documents in group `i` tend to contain word `w` more often, a large negative value indicates that word `w` is more associated with documents in group `j`. Without an informative prior, equation 20 boils down to the observed log-odds ratio. Monroe et al. [10] advise to inform the choice for a meaningful prior by what we know about the actual distribution of words in an average document in the corpus. We know for instance that the word `mevr` (Engl. `Mrs.`) occurs more often than the word `bronchitis` in our example data. Therefore, $\alpha_0^{(i)}$ should be chosen in such a way that it shrinks $\pi_w^{(i)}$ and $\Omega_w^{(i)}$ to more average values for frequently occurring words. Following Monroe et al.'s example, we choose α_0 equal to the average number of tokens in a document, across all examined groups.

Finally and in accordance with Monroe et al. [10], we use z-scores of point estimates instead of using them directly for feature evaluation. This is because point estimates are prone to overfitting ideosyncratic words. At this point, we profit from having taken a model-based approach, because ideosycratic words will not only have high point estimates, but also high variance. Under the given model, we can approximate the variance σ^2 of the log-odds ratio of two groups `i` and `j` with

$$\sigma^2(\hat{\delta}_w^{(i-j)}) = \frac{1}{(y_w^{(i)} + \alpha_w^{(i)})} + \frac{1}{(y_w^{(j)} + \alpha_w^{(j)})}. \tag{21}$$

The z-scores of the log-odds ratio can then be calculated with

$$\hat{\zeta}_w^{(i-j)} = \frac{\hat{\delta}_w^{(i-j)}}{\sqrt{\sigma^2(\hat{\delta}_w^{(i-j)})}}. \tag{22}$$

2.3.2 Jensen-Shannon divergence

Identifying characteristic words depending on the level of a categorical meta feature can be framed as a corpus comparison, where the documents associated with different levels of a meta feature are treated as different text corpora. The Jensen-Shannon divergence (JSD) is a popular tool for corpus comparison and has recently been extended by Lu, Henchion, and Namee [11] for the case of more than two corpora and for simultaneous comparison of both word unigrams and bigrams.

The JSD originated from and represents an improvement over the Kullback-Leibner (KL) divergence [38] as a statistical measure that captures the difference between two probability distributions. Given two probability distributions `P` and `Q`, the Kullback-Leibner (KL) divergence is defined as

$$D_{KL}(P||Q) = \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{q_i} \tag{23}$$

where `n` is the sample size. In the context of text corpora, `n` is interpreted as the number of unique words and p_i is the probability with which word `i` occurs in `P` and q_i is respectively the probability with which `i` occurs in `Q`. Applying the KL divergence to text corpora is likely to pose problems, though, because words that only occur in one, but not the other corpus yield infinitely large values. Gallagher et al. [39] proposed to use the JSD instead and suggested a rephrased form with respect to the original JSD proposed by Lin [40]. The JSD is a smoothed, symmetrical variant of the KL divergence, defined as

$$D_{JS}(P||Q) = \pi_1 D_{KL}(P||M) + \pi_2 D_{KL}(Q||M). \tag{24}$$

The problem of infinitely large divergence is solved by introducing `M`, a mixed distribution with $M = \pi_1 P + \pi_2 Q$, where π_1 and π_2 represent weights proportional to the sizes of `P` and `Q`, with $\pi_1 + \pi_2 = 1$. A JSD score close to 0 means that the word probability distributions of the two compared corpora are similar. A JSD score of 1 indicates that there are no common words in the compared word probability distributions. Accordingly, for `n` probability distributions we can calculate the JSD with

$$D_{JS}(P_1||P_2||...||P_n) = \sum_{i=1}^{n} \pi_i D_{KL}(P_i||M) \tag{25}$$

In addition, Lu et al. [11] contributed an extension of the JSD that allows us to calculate the individual contribution of word `i` to the divergence of `n` probability distributions:

$$D_{JS,i}(P_1||P_2||...||P_n) = -m_i \log_2 m_i + \sum_{j=1}^{n} \pi_j p_{ji} log_2 p_{ji} \tag{26}$$

with p_{ij} representing the probability of word `i` occurring in corpus P_j and m_i being the probability of `i` occurring in `M`. For `n` corpora, `M` is defined as:

$$M = \sum_{i=1}^{n} \pi_i P_i \tag{27}$$

where $\sum_{i=1}^{n} \pi_i = 1$ holds and weights are proportional to the sizes of P_1 to P_n. In the context of the different levels of our meta features, we expect the words contributing most to the divergence of two word probability distributions at hand to match with the words that are identified as being characteristic for either one of the two compared groups by the log-odds ratios. If a word contributes strongly to the divergence, it can be expected to be associated with one, but not with the other corpus.

2.3.3 Results

Log-odds ratios

This section is structured as follows: first, some general results obtained from the log-odds ratios of each categorical meta feature are summarized. Second, for `Healthcare sector`, results are presented in detail to exemplify how individual features were evaluated. Details of other individual feature evaluations are included were applicable.

General results

Generally speaking, the obtained log-odds ratio results suggest that the examined meta features do affect choice of words in the sampled data set. The obtained results are in line with our intuitions about how certain levels of a meta feature influence report writing. For example, as can be seen in Figure 4, for `patient gender`, gender-specific ways of addressing the patient in a report, such as `mv` (`mevrouw`, Engl. `Mrs.`) and `heer` (Engl. `Mr.`) produce high log-odds ratio z-scores, meaning that the data suggests with fair certainty that these gender-specific forms of addressing patients are characteristc for reports written about either female or male patients. Other intuitive word usage differences depending on patient gender reflect different care needs depending on gender, with words such as `scheren` (Engl. `to shave`) and `katheter` (Engl. `catheter`) being more associated with reports written on male patients and the word `steunkousen` (Engl. `support hose`) being more characteristic for reports written on female patients.
Another general trend is the magnitude and frequency of large z-scores. As can be seen in Figure 4, the point estimates of the log-odds ratios for `patient gender` tend to produce large z-scores. This can be seen as a property of the data. Our z-scores are only meaningful when evaluated in relation to each other. We pay special attention to words that produce high z-scores compared to the rest of the vocabulary without actively interpreting statistical significance.
We can further observe that extreme class imbalance as it occurs in several meta features masks the distinctiveness of minority classes when they are combined with the majority class. This essentially turns any one-vs-rest comparison to a one-vs-majority class comparison. This is well visible in the detailed `healthcare sector` comparison depicted in Figure 7 where the mental care sector (mental healthcare) loses its distinctiveness when combined with the majority class, the elderly care sector.

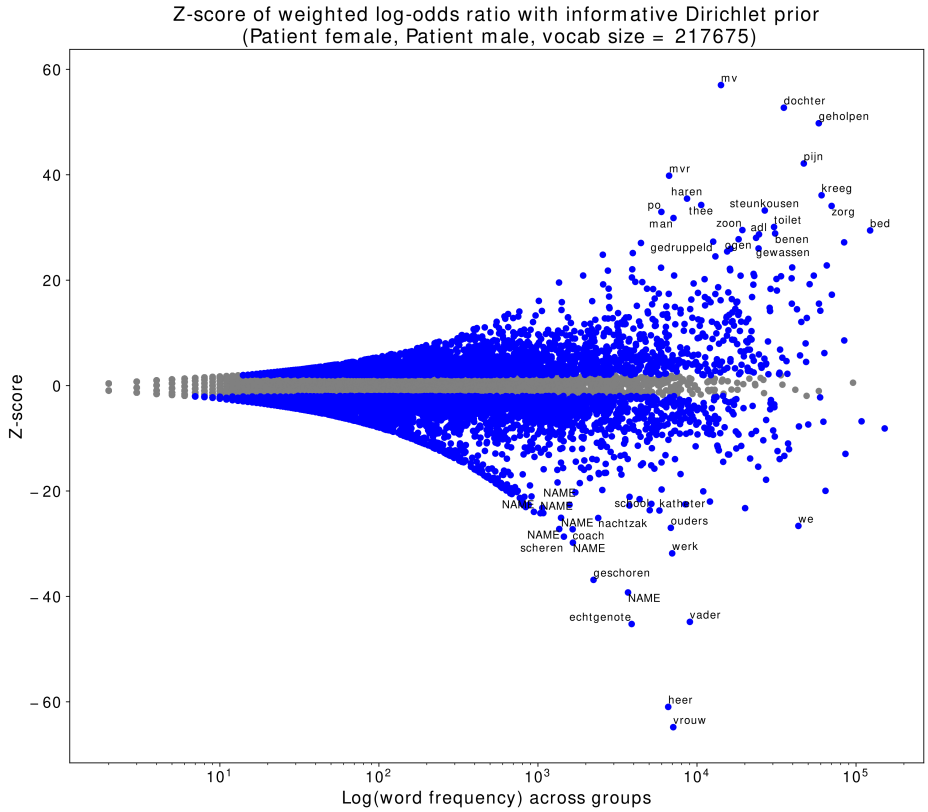


Figure 4: Word feature evaluation using z-scores of log-odds ratios for the two levels of the `patient gender` meta feature. Positive z-scores are associated with the `female` group and negative scores with the `male` group. The x-axis shows the frequency of words on a log-scale. The top 20 words that produce the largest z-scores are annotated for both groups. Z-scores larger than 1.96 are coloured blue. They indicate values that are more than |1.96| standard deviations away from the mean in a normal distribution. Person names were anonymized as `NAME` and customer names as `CUSTOMER NAME`.

Healthcare sector

Figures 5 to 8 shows for each word in a shared vocabulary of two or more healthcare sectors the z-score of the log-odds ratio using a Dirichlet prior. As can be seen in Figures 6, 7 and 8, the extreme class imbalance within the `healthcare sector` meta feature shows as the elderly class introduces extremely large raw frequency scores. In addition, as depicted in Figure 5, the shared vocabulary of the regional protected living (RPL) is much smaller when combined with the mental and disabled care sector than when it is combined with reports originating from the elderly sector. Note how the most distinctive words for the RPL class vary depending on the class it is compared against. When compared against the mental healthcare sector, adverbs of time (e.g., `vandaag` (Engl. `today`) and `vanmorgen` (Engl. `this morning`)) and words about activities of daily living (ADL) (e.g., `eten` (Engl. `food` or `meal`) and `bed`) and appointments (e.g., `gebeld` (Engl. `called`) and `uur` (Engl. `hour`)) dominate the list of most distinctive words. However, when RPL is compared against the elderly care sector, the class' by far most characteristic words are names of healthcare organisations. Intuitively, this suggests that vocabularies of reports written in RPL and elderly care are more similar than between RPL and mental healthcare. This is confirmed in Section 2.3.3 by the `Jensen-Shannon divergence` (JSD) measure of similarity between the word probability distributions of the three healthcare sectors.

The most distinctive words for the mental healthcare sector confirm intuitions, as words describing emotions and other mental states produce high log-odds ratio z-scores (e.g., `voelen` (Engl. `to feel`), `angst` (Engl. `fear`), `spanning` (Engl. `tension`)), regardless of the sector it is compared against. In addition, when compared against the elderly care sector, words characteristic for treatment in mental care produce high z-scores, such as `sessie` (Engl. `session`), `cliënte` (Engl. `(female) client`, as opposed to calling clients `patients`) and `groep` (Engl. `group`).

From Figures 5 to 8, we can further observe that when the mental healthcare sector is combined with the elderly care sector in a one-against-the-rest comparison against RPL, the words from the majority class, that is, elderly care, outweigh words characteristic for the mental healthcare sector by far, because technically, most reports of this hybrid class originate from elderly care. Essentially, the weights of words characteristic for the mental healthcare sector are trumped by the massively larger elderly care sector, which hides the distinctiveness of the mental healthcare sector.

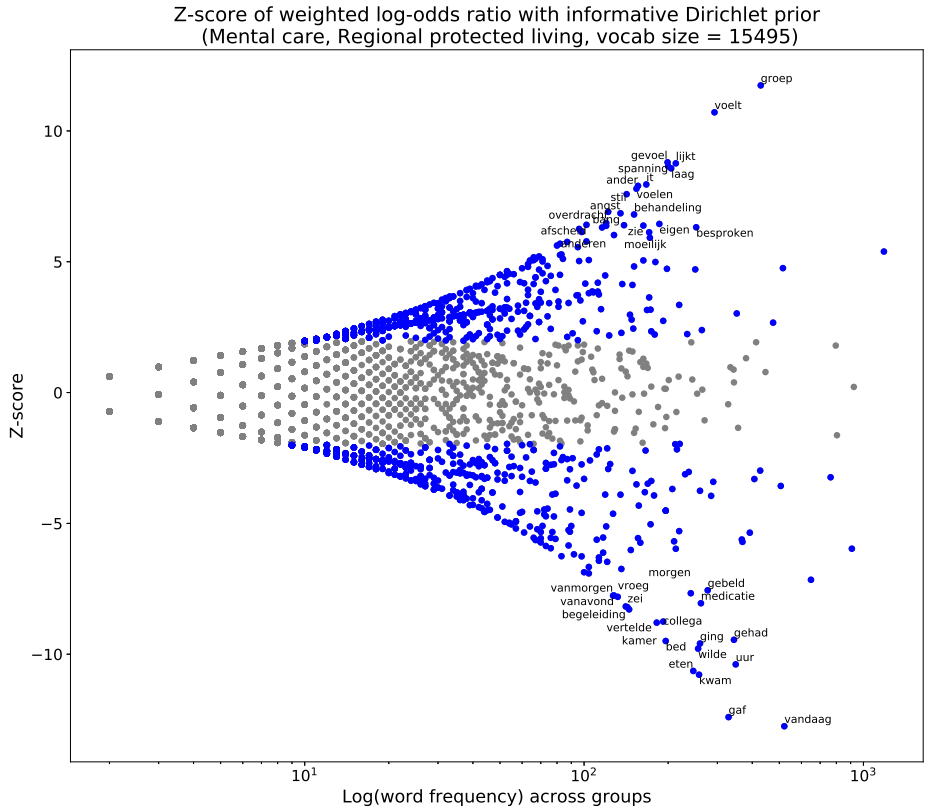


Figure 5: Word feature evaluation using z-scores of log-odds ratios for the mental healthcare sector and the regional protected living sector. Positive z-scores are associated with the `mental care` group and negative scores with the `regional protected living` group. Person names were anonymized as `NAME` and customer names as `CUSTOMER NAME`.

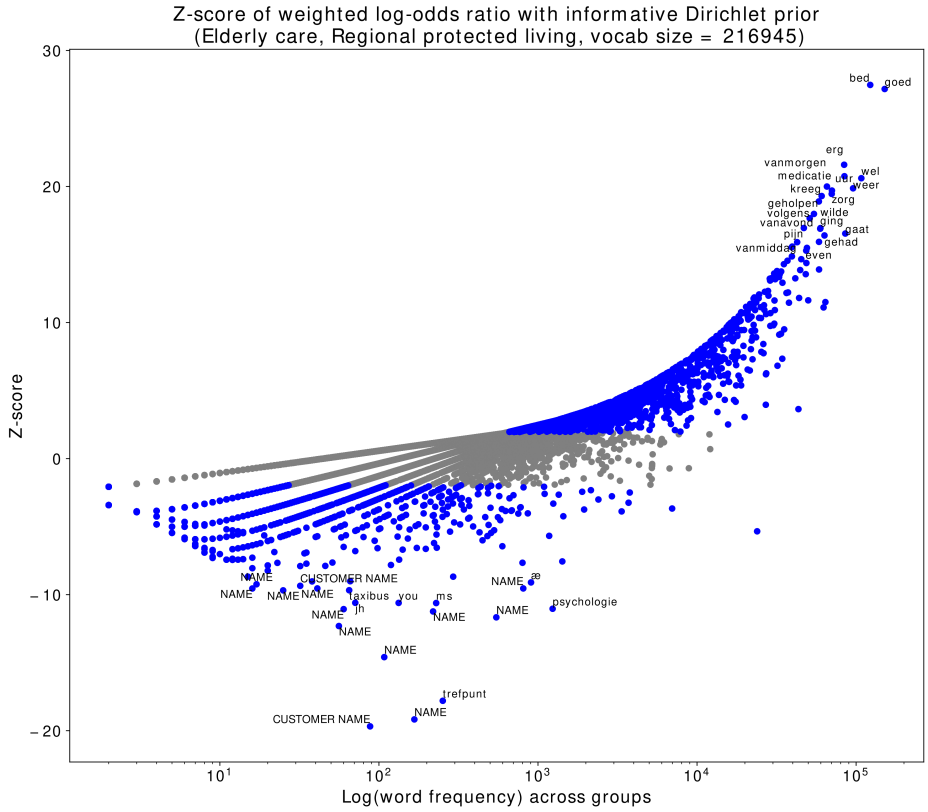


Figure 6: Word feature evaluation using z-scores of log-odds ratios for the elderly care sector and the regional protected living sector. Positive z-scores are associated with the `elderly care` group and negative scores with the `regional protected living` group. Person names were anonymized as `NAME` and customer names as `CUSTOMER NAME`.

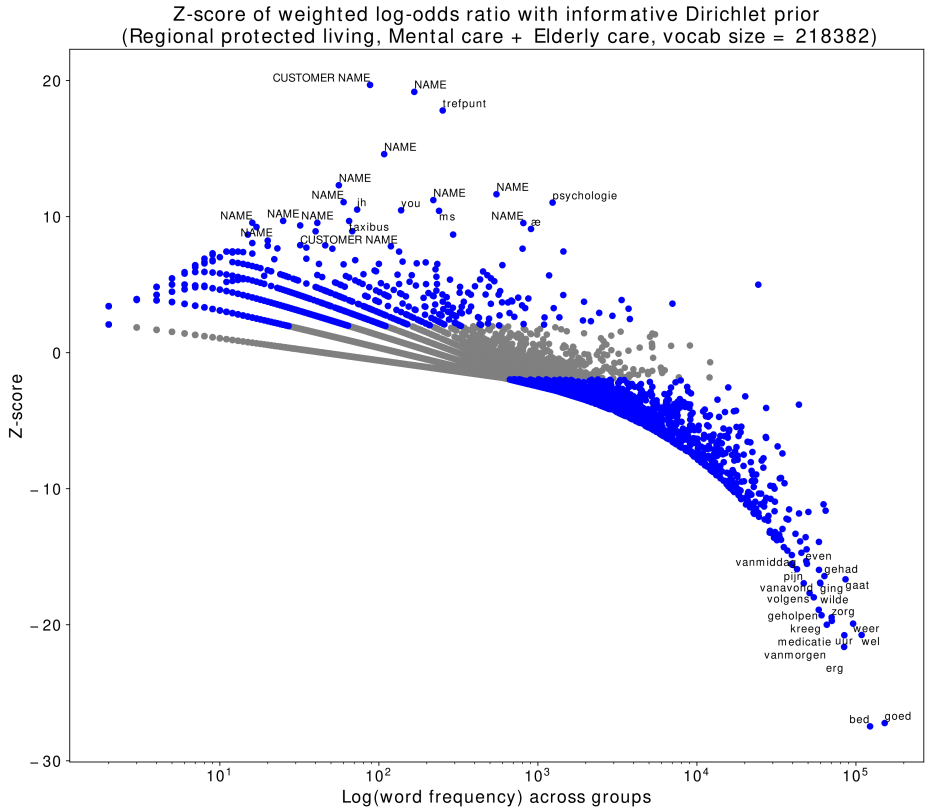


Figure 7: Word feature evaluation using z-scores of log-odds ratios for the regional protected living sector and the combined data from the mental and elderly care sectors. Positive z-scores are associated with the `regional protected living` group and negative scores with the combined mental and elderly care group. Person names were anonymized as `NAME` and customer names as `CUSTOMER NAME`.

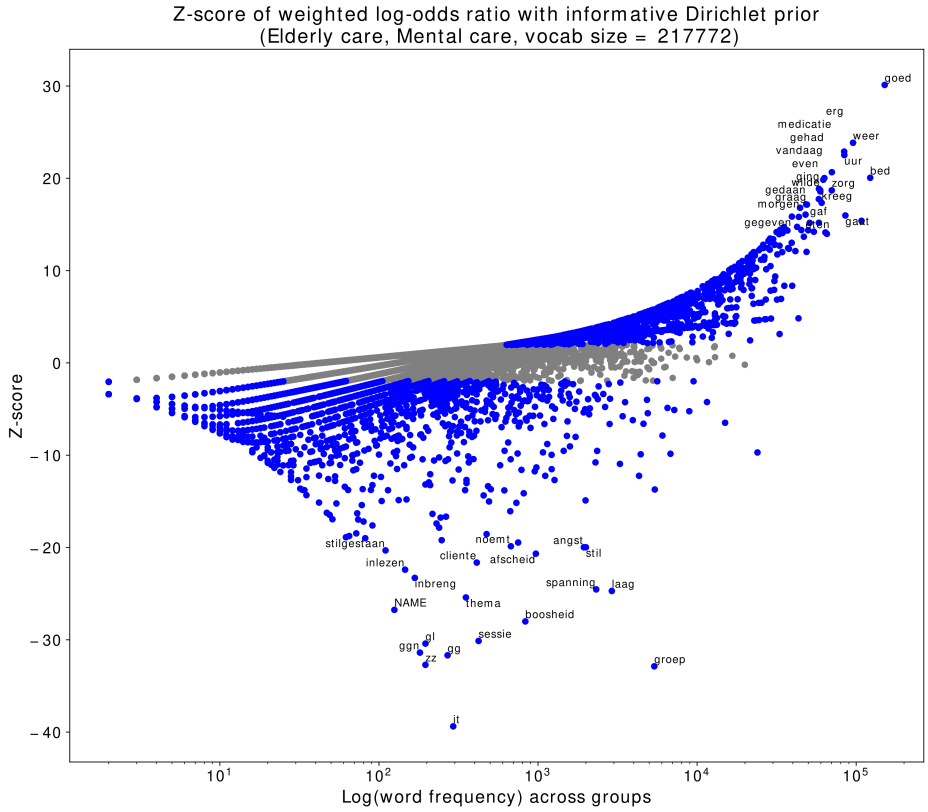


Figure 8: Word feature evaluation using z-scores of log-odds ratios for the elderly care sector and the mental care sector sector. Positive z-scores are associated with the `elderly care` group and negative scores with the `mental care` group. Person names were anonymized as `NAME` and customer names as `CUSTOMER NAME`.

Jensen-Shannon divergence

Table 7 summarizes Jensen-Shannon divergence (JSD) scores for the examined feature levels. For each JSD score, five words are listed from the joint vocabulary of the compared classes that contribute most to the obtained JSD score. The five most contributing words were determined as proposed by Lu et al. [11]. Examined customer care organisations are anonymized with captial letters `A`, `B`, and `C` respectively. According to the JSD scores, word probability distributions are most distinct between `report type` classes, followed by `employee expertise` clusters, and `healthcare sectors`. Word distributions depending on `gender` and `age`, either for patients or employees, are similar according to the obtained JSD scores, suggesting that `gender` and `age` do not influence word choice. Word distributions between the three largest customer organisations are also marginally different.

The five words that we identified to differ most between word probability distributions using the method proposed by Lu et al. [11] match well with the obtained results from the log-odds ratios. This speaks for the validity of both methods. For example, for the `healthcare sector` feature, the five most characteristic mental healthcare words on the basis of log-odds ratio z-scores are `groep, voelt, gevoel, lijkt` en `spanning` when compared against the regional protected living (RPL) sector. For the latter, `vandaag, gaf, kwam, eten` and `uur` are identified as most characteristic. Comparing these top five most characteristic words for either of the two feature classes with the top five most important words based on their JSD contribution, we find that four out of the latter five were also identified by the log-odds method. The last of the top five most JSD contributing words, `laag`, has the sixth highest log-odds z-score for the mental healthcare class when it is compared against RPL.

Table 7: Summary of Jensen-Shannon divergence (JSD) scores per meta feature. For each similarity assessment, the top 5 words are listed that contribute most to the divergence.

Feature classes	JSD	Top 5 most contributing words
Patient gender	.026	mv, heer, mevr, vrouw, echtgenote
Employee gender	.039	vanmorgen, bed, gewassen, erg, dd
Healthcare sector		
elderly, mental healthcare	.26	bed, groep, vanmorgen, geholpen, zorg
elderly, RPL	.127	zorg, moeder, bed, we, geholpen
mental healthcare, RPL	.239	gaf, laag, vandaag, voelt, groep
All	.299	bed, groep, zorg, vanmorgen, laag
Report type		
Omaha, Text	.236	blijft, evaluatie, tevreden, vanmorgen, zorg
Omaha, Defecation	.466	ontlasting, gehad, toilet, vanmorgen, normaal
Omaha, Medical	.249	ysis, afkomstig, deskundigheid, vanuit, p
Text, Medical	.14	ysis, deskundigheid, afkomstig, vanuit, verpleegkunde
Text, Defecation	.342	def, ontlasting, gehad, toilet, normaal
Medical, Defecation	.409	def, ontlasting, gehad, toilet, incontinent
All	.529	ontlasting, gehad, toilet, normaal, incontinent
Expertise		
Doctor, Nurse	.17	dd, mg, b, p, kreeg
Doctor, Daycare	.325	dd, mg, vanmorgen, vanmiddag, mee
Doctor, Psychologist	.216	dd, mg, b, pijn, lab
Psychologist, Nurse	.206	gedrag, vanavond, wond, psycholoog, kreeg
Psychologist, Daycare	.237	vanmorgen, vanmiddag, vandaag, psycholoog, ging
Daycare, Nurse	.192	zorg, bed, mee, medicatie, we
All	.378	dd, mg, vanmorgen, vanmiddag, vandaag
Customer		
A, B	.119	vv, plan, zorg, volgens, -
A, C	.122	plan, -, zorg, volgens, zvp
B, C	.091	vv, kreeg, def, ontlasting, steunkousen
All	.158	vv, plan, -, zorg, volgens
Patient age		
0-30, 31-60	.06	moeder, school, vader, ouders, ysis
0-30, 61-114	.158	moeder, bed, geholpen, zorg, school
31-60, 61-114	.086	bed, geholpen, kreeg, we, zorg
All	.141	moeder, bed, geholpen, school, zorg
Employee age		
12-30, 31-50	.02	mv, gaf, vlgs, coach, vanochtend
12-30, 51-99	.032	mv, gaf, vlgs, med, geeft
31-50, 51-99	.021	mv, med, geeft, gaf, we
All	.034	mv, gaf, vlgs, med, geeft

2.3.4 Discussion meta feature evaluation

The current meta feature evaluation examined the extent to which word usage in reports is influenced by report meta information, such as the `healthcare sector` a report originates from. Two methods were used to evaluate meta information. For each word in a shared vocabulary of two meta feature level, log-odds ratios were calculated following the methodology proposed by Monroe et al. [10]. Using the z-scores of the log-odds ratio point estimates, we not only identified characteristic words, but also took the reliability of the point estimate into account. Second, Jensen-Shannon divergence (JSD) [11] scores were calculated to assess how similar or dissimilar word probability distributions are across the levels of a meta feature. The obtained results univocally suggest that some meta features can identify word usage differences in medical reports. According to our analysis, the `gender` and `age` features for clients and employees, as well as the `care organisation` feature are unsuitable for identifying differences in word choices. In contrast, the `report type`, `healthcare sector`, `expertise` feature exhibit remarkable distinctive value. For `employee expertise` this is even more remarkable considering the simplicity with which expertise clusters were formed in the current analysis.

The extreme class imbalance in the examined meta information is likely to pose a problem for learning from explicitly provided meta information. Especially for the log-odds ratio method we saw that otherwise distinctive words for smaller feature classes are masked when a smaller class is combined with the majority class, essentially turning any decision involving, but not exclusively involving the majority class, into a decision whether a word is more distinctive for the class at hand or the majority class.

One important drawback of the employed methodology is its focus on main effects, and therefore, its ignorance of interaction effects between levels of meta features. For instance, according to the obtained JSD scores for the `care organisation` feature, the word probability distributions of the three most occurring customer organisations differ marginally. However, all three organisations work in the elderly care sector. As a result, their clients have similar care needs and sector-specific ways of report writing are standardized. However, it is still possible that the `care organisation` feature interacts with another feature, say the `healthcare sector` feature, because two customer organisations specialize in caring for specific client groups, such as adult clients compared to child clients.

A limitation of how we employed the log-odds evaluation is its focus on differences. We emphasized distinctive word tokens in our plots using overlayed colour coding for z-scores within the 95% confidence interval, which overemphasizes the extent to which the two compared text corpora are distinct. This is because the colour coded data points occlude some of the other data points. The actual fraction of words identified as reliably distinct is much smaller than our plots may suggest. For instance, for the `patient gender` feature, only 0.052% of all z-scores are larger than 1.96. As a result, our employed log-odds method does reliably identify distinctive words, but is less suitable for giving an impres-

sion on how different the word distributions are as a whole. Fortunately, the calculated JSD scores complement the log-odds method in this regard and we refer to them for getting an impression on how different two or multiple text corpora are in general.

Another limitation of the employed methodology is that we looked at sub-samples, instead of the full picture due to the high complexity of most meta features. For instance, instead of looking at all 18 report types, only four were covered in the current meta feature evaluation. One the one hand, this increases the chance that we missed some important trends regarding the capacity of meta information to distinguish word usage in our medical reports. On the other hand, focusing on sub-samples enabled us to analyze the differences between levels of meta features with greater detail, that is, on the level of word tokens. In addition, the high dimensionality of the meta information makes it challenging, if not unfeasible, to include all information in one analysis while still providing meaningful results.

2.4 Concluding remarks and future work

This chapter set out to examine **RQ1**, the extent to which meta information about medical records in the `Ons` database can be useful for **FutureType**, a text prediction model that is currently under development at the Dutch technology company Nedap. To this end, we reviewed relevant literature on how recurrent neural network (RNN) language models such as FutureType can be enriched with additional meta input. To introduce FutureType formally, we described its architecture and training in detail. Finally, before setting out to build a meta-feature enhanced FutureType model, we evaluated a set of candidate meta features with regard to their capacity to distinguish word usage in medical report writing. For this we employed two methods, log-odds ratios [10] and Jensen-Shannon divergence scores (JSD) [11].

We saw that the model's performance in terms of prediction accuracy and perplexity increases sharply and then flattens depending on the number of known letters of the target word. Our evaluation of FutureType as described in this chapter is limited in an important way. We only trained and evaluated the model for target words that consisted of at least six letters. This means that FutureType cannot predict words with less than six letters. We chose to bias the model towards long words because we anticipated that the model's predictions are most beneficial in terms of preventing spelling mistakes and saving keystrokes for long, complex and possibly infrequent medical terms. However, our dataset statistics show that the median length of all words in the training, validation and test sets was five. This means that many words in our medical report database are shorter than six letters and cannot be predicted by FutureType by design. Since we trained and evaluated FutureType only for words with minimally six letters, the performance we report is likely higher than the performance FutureType would achieve if it were used to predict words of all lengths. In a downstream scenario with end users, this can mean that FutureType often suggests irrelevant words, also named `unnecessary distractions (UDs)`. If the majority of words in medical reports is indeed shorter than six letters and we impose no further restrictions on word suggestions, the user will often get a wrong, long word suggestion simply because FutureType is incapable of predicting shorter words.

Our results from the meta feature evaluation indicate that some, but not all, candidate meta features can distinguish differences in word choice. The three meta features that have the most dinstinguishing power are the type of the report, the healthcare sector a medical document originates from and the expertise of the author of the document. Meta features with negligible distinctive value are the `age` and `gender` of the patient about whom the report is written and the author of the report. Explicitly providing the healthcare organisation from which a report originates also appears to be a futile effort. We discussed drawbacks of the methodology we employed. One drawback is that we are unable to observe interaction effects between multiple meta features because we only compared a selected set of levels of one and the same feature, such as the the vocabulary of `male` compared against the vocabulary of `female` authors. Some systematic differences may only be meaningful if a set of meta features is considered at the same time. However, the employed methodology is unsuitable

for visualizing such interaction because of the large number of feature levels of some of our meta features, especially the `employee expertise` feature. Considering that we applied only simple filtering startegies to obtain expertise clusters and still, the feature showed remarkable distinctive power, we recommend for future work to put effort into deriving sound expertise clusters from the 3758 unique expertise titles in the `Ons` database. The `employee expertise` feature is a strong candidate for complementing FutureType's predictive capacity.

Even if future work succeeds in identifying meaningful interactions and provides a set of well-interacting and meaningful meta features with proven distinctive power, it remains unclear whether FutureType will profit from their inclusion in the model. It may implicitly extract any distinctive value of the provided meta features from a large enough text corpus because any differences in word usage are implicit in the text data. In that case, adding meta information explicitly may be redundant. We established for some meta features that they are suitable for detecting word usage differences, the next step is to see whether explicitly providing this information to FutureType does indeed improve its predictive performance.

As a first step, we started re-organizing the code repository at Nedap Healthcare for training FutureType and sampled a new training, validation and test set. The re-organized and newly sampled data set includes 308 649 554 (308, 649M) randomly sampled documents, the same documents that were sampled for the initial training of FutureType. Each document is a medical report under the administration of Nedap Healthcare. A major difference between our previous and current sampling strategy is that we no longer sample word sequences from a document subset, but we sample documents from the entire set of 308 million available medical reports. The dataset was divided into training, validation and test set using a 90/5/5 split, where 90% of the documents were assigned to the training set, and respectively 5% to validation and test set each.

To further reduce data leakage, training, validation and test documents were sampled in a stratified fashion. While splitting the dataset into training, validation and test samples ensures that the model does not encounter any documents during training that are shared with the test set, information from the test set can still leak into the training process. That is because we enrich the model with meta information that is shared across training and test samples. For example, out of two randomly sampled reports, one is assigned to the training set and the other report to the test set. The two reports are distinct documents covering different text content. However, they were written by the same employee. While we do not explicitly feed employee identifiers to the model, the model can in theory still learn from implicitly leaked information between training and test documents using the set of meta features it is provided with. In fact, given enough data, the model could learn a representation of individual employees or clients. While the latter would be ethically problematic, any data leakage from test to training documents sheds doubt on the reliability of performance evaluations. Therefore, we used `customer code` and `employee id` combinations as an additional splitting variable, ensuring that no reports written by the same employee are shared between the training and test data. Due to computational restrictions, the training set was downsampled to 1% for model

training, denoted as `training downsampled` in Table 8.

Table 8: Descriptives of the re-organized and newly sampled data set from 308 649 554 medical reports in the database of Nedap Healthcare.

	Training	Validation	Test	Training (downsampled)
N documents	275.752M	15.221M	15.480M	2.661M
N tokens*	8.103B	452.791M	459.466M	79.546M
Median N tokens per doc	16	17	16	17
Vocab size (tf 1**)	1.353M	663 487	662 525	289 661
Vocab size (tf 2)	1.353M	425 293	423 104	157 019
Vocab size (tf 5)	1.353M	226 147	225 068	82 659
Vocab size (tf 10)	837 395	147 689	146 693	54 458
Vocab size (tf 20)	537 758	98 664	98 250	36 742
Vocab size (tf 100)	206 420	39 897	40 032	14 975
N employees	109 667	9938	9939	2351

* Here, `token` means a unigram word.
** `tf` stands for `minimum term frequency` and means that all tokens included in the vocabulary occurred at least `n` times. For `tf 1`, all tokens in the corpus are counted.

Regarding the architecture for a baseline meta-enriched FutureType, we recommend implementing the simplest form of adding meta information to a recurrent neural language model, the approach proposed by Mikolov and Zweig [13], the `ConcatCell` model. The ConcatCell model can optionally be implemented with a `SoftmaxBias`, another popular technique for adjusting the weights to information outside the immediate text context. Once a first meta-feature enriched FutureType model is implemented, future work can be devoted to further explore the interaction of our meta features and alternative ways of incorprating meta information in the model, such as an extension of the `ConcatCell` architecture to a `FactorCell` approach, or by giving more attention to how meta features are represented in an additional feature vector, similar to the research efforts of Zheng, Chen, Huang, Liu, and Zhu [23].

Interaction effects between the chosen set of meta features can be approximated by making their inclusion in the model a part of the hyperparameter space that is searched during hyperparameter optimization. Alternatively, systematic model ablation [5] can be applied to a meta-enriched FutureType model to experiment with the inclusion and exclusion of specific meta features. Spithourakis, Peterson and Riedel [5] ablated their meta-enriched medical text prediction model to examine the added value of additional patient data, including the gender of the patient and several numerically-encoded medical test results. An example for such a medical test result are end diastolic and systolic volumes for the left and right heart ventricles as measured through magnetic resonance imaging (MRI). These volumes capture the amount of blood that is in the ventricles before and after the heart contracts and are a measure of the health of a person's heart.

Ultimately, if the inclusion of explicit meta features turns out to be of little additional value with regard to FutureType's performance, for the sake of par-

simony [41], [42], they should be abandoned and future efforts should focus on improving FutureType in alternative ways. In a way, we already took a step into that direction by conducting a large scale pilot study with real end users, as the next chapter describes.

3 FutureType pilot

While the previous chapter generated recommendations with regard to the recurrent neural network (RNN) language model underlying **FutureType**, i.e., the technical backend of the text prediction application, the current chapter focuses on a second pillar of software development, early and (preferably) iterative user testing. In her blog post on reasons why usability testing can save software companies money, Collete Stumpf stresses the paramount importance of early user testing:

> "Fixing an error in a product can be up to 100 times more expensive than it would have been to implement early-stage testing."[8]

Indeed, it is well documented in literature that early involvement of end users in software development is crucial for system success (e.g., [7], [9]). According to Damodaran [43] and Kujala [44], empirical results show that user involvement in system design yields the following benefits:

1. Overall improved system quality because user requirements are captured more accurately
2. Costly system features that the user does not want or use can be recognized early and avoided
3. Higher user acceptance of the system
4. More effective use because the user better understands the system
5. Increased participation in future feature decision-making

The above empirically grounded benefits are somewhat focused on `participatory` user testing, where users actively contribute to the system's design. However, Kujala [44] confirms in her review on the benefits and challenges of user involvement that benefits 1 to 3 do apply to other user testing approaches. In the current research, we chose A/B testing as our user testing method. A/B tests are a form of online controlled experiment [45]. The simplest A/B test setup involves two conditions: the control condition `A`, which is the default or current version of a feature, and a treatment condition `B`, which is the change in the feature to be tested. Since A/B tests are a form of controlled experiments, they allow us to establish causal relationships between a system change and its effect on observable user behaviour [46]. A/B testing is also called *split testing* when more than two conditions are involved. Traditionally, A/B tests have been employed to increase revenue and more recently for usability testing [7]. In the current research, we employ a split test involving 8 different conditions.

Our pilot study represents the first user test of FutureType with its target group, healthcare professionals whose daily activities inlcude writing medical reports. Next to documenting FutureType's first large scale user evaluation, the current chapter explores how well common intrinsic evaluation metrics align with how end users receive a text prediction application in *downstream* evaluation.

[8] https://www.surgeforward.com/top-5-reason-why-usability-testing-can-save-you-money/, last accessed 2020-03-01

Unfortunately, the term *downstream* evaluation is misleading in natural language processing (NLP), since it suggests that language models are evaluated under conditions that are comparable with real world applications. However, what is meant with *downstream* is the evaluation on a number of benchmark tasks that are still distant from real world applications. As a result, the question arises to what extent the results obtained from such benchmark studies can be transferred to real world applications. There has been evidence that evaluation metrics used in common downstream tasks in NLP may align poorly with the demands of real world applications. Cumulative evidence indicates that perplexity, the most common benchmarking evaluation metric for language models [19], poorly aligns with other text prediction metrics, such as mean reciprocal rank (MRR), which is the multiplicative inverse rank of the correct word in a suggestion list [5], and prediction accuracy [19]. Ultimately, it is likely that the evaluation of language models that are designed for use beyond benchmarking their capabilities can better opt for a holistic evaluation approach that captures both, the system's `utility` and `usability` [9].
In the current study, we evaluated FutureType instrinsically with prediction accuracy and perplexity. It remains unclear how well our chosen metric translates to differences in user satisfaction and performance in real world settings, although there is evidence that the quality of predictions, that is, the system's utility, greatly impacts user experience [3], [47]. Intrinsic model evaluation is certainly less costly and more efficient than elaborate user testing. Therefore, it is an important task to establish the transferability of intrinsic results to external settings.

The current chapter is organized as follows. In section 3.1, we first review relevant literature about keystroke analysis, a common method for evaluating text prediction systems with end users. Secondly, we introduce model ablation, our chosen method for generating multiple versions of FutureType that score differently on prediction accuracy. Section 3.2 describes in detail how we set up the FutureType pilot and how we conducted the keystroke analysis. The remaining sections document and discuss the results we obtained from the keystroke data.

3.1 Related work

Keystroke analysis

Analyzing keystrokes, either collected from real end users or by simulating the typing behaviour of potential users, is a common user testing method for text prediction applications [5], [48]. Text prediction applications originated in the field of augmentative and alternative communication (AAC) for motor and speech impaired people (e.g., [49], [47]), where each saved keystroke matters to reduce communication effort for the user and to increase the user's communication rate, i.e., producing more text faster and with less effort.

Recently, Dhakal, Feit, Kristensson, and Oulasvirta [48] analyzed keystroke patterns from 168 000 online volunteers and show that typing behaviour is subject to individual differences. Main insights from their research include the identification of eight groups of typists that differ in typing accuracy, performance,

hand and finger usage, and `rollover`. Rollover is the phenomenon of pressing the next key before the previous is released. Rollover was found to be a strong predictor of typing speed (Pearson correlation coefficient r = 0.73). Contradictory to common belief, formal typing training appears to be no prerequisite for fast, error-free typing, though fast typists ($65 - 68$ words per minute (WPM)) do make less errors than slow typists ($46 - 48$ WPM). The authors claim to have captured modern typing behaviour by including untrained typists and multiple keyboard layouts (physical, on-screen, laptop) in their sample.
Dhakal et al. further review a number of standard typing evaluation metrics in keystroke analysis, including `typing speed` in words per minute (WPM) or characters per minute (CPM), `error rate`, `inter-key intervals` (IKIs; the difference in timestamps between two consecutive keypress events), `keypress durations`, `keystrokes per character` (KSPC; i.e. number of keystrokes divided by the number of characters in the final string produced), and `error corrections` (i.e. the percentage of keypresses using the `Backspace` or `Delete` key during typing). The authors further argue that current modelling assumptions behind the design of text entry applications need to be updated as our knowledge of typing proficiency has recently been extended with metrics such as rollover, consistent key-to-finger mapping, and reliance on visual information [50].

Using keystroke analysis, a number of benefits and challenges of text prediction applications have emerged. On the one hand, text prediction has been shown to save keystrokes and prevent spelling mistakes [3], [49]. On the other hand, while auto completion has been found to increase typing speed for impaired people using on-screen keyboards [47], [49], results are less promising for able-bodied people using physical keyboards [3], [51], [52]. Intuitively, keystroke savings are negatively proportional to time spent on typing. However, using a text prediction application is bound to introduce cognitive overhead by exposing typists to word suggestions [47], [52]. Typists are forced to scan through predictions and to shift their focus. Trnka, McCaw, Yarrington, and McCoy [47] developed a mathematical model that directly captures the trade-off between keystroke savings and input rate (i.e. the time needed to enter a single keystroke, measured in `seconds per keystroke (SPK)`). The model can be used to identify cases in which text prediction is slower (or faster) than letter-by-letter text entry. The model represents speedup (or slowdown) as the ratio between communication rate (i.e. output rate or typing speed, number of characters produced per time unit) with and without text prediction

$$speedup = \frac{1}{(1 - \frac{\text{actual keystroke savings}}{100}) * \frac{\text{SPK}_{\text{aided}}}{\text{SPK}_{\text{unaided}}}} \tag{28}$$

where $\text{SPK}_{\text{unaided}}$ and $\text{SPK}_{\text{aided}}$ represent seconds per keystroke for letter-by-letter text entry and text entry using text prediction, respectively. We leave examining speedup vs. slowdown experiments involving FutureType to future work and focus on gathering initial usage statistics and comparing our eight ablated FutureType models in the current research.

For FutureType's first evaluation with its target users, healthcare professionals who write medical reports as part of their daily work activities, we chose keystroke analysis as our user testing method. By analyzing keystrokes from

users who use FutureType, we not only get an insight into how our users type, but we can monitor the usage of FutureType remotely in an unobtrusive fashion and under naturalistic usage conditions.

Model ablation

Our second research question is about the extent to which model performance differences as measured by intrinsic performance metrics such as perplexity and prediction accuracy impact end users. To this end, we compare user acceptance and performance differences between eight versions of FutureType. Our chosen method for generating multiple versions of FutureType that score differently on prediction accuracy is `ablation`. The term originally stems from the field of clinical neuropsychology and describes the removal or destruction of body tissue or anatomical structures by means of surgery, disease or other physical or energetic processes. Ablation is employed as treatment or to study the function of bodily parts [53].
In June 2018, a tweet post by François Chollet, the primary author of the Keras deep learning framework, re-directed attention to the term `ablation` in the machine learning community. In the post, Chollet stresses the utility of ablation for studying causal effects in deep learning models:

> "Ablation studies are crucial for deep learning research – can't stress this enough.
>
> Understanding causality in your system is the most straightforward way to generate reliable knowledge (the goal of any research). And ablation is a very low-effort way to look into causality."[9]

In the context of machine learning, the term `ablation` is used to describe the systematic removal of parts of a complex neural network to gain more insight into the network's behaviour.
For example, in their influential paper in the field of object detection, Girshick, Donahue, Darrel and Malik [54] present a large and complex neural model that consists of three modules. The first identifies regions in an image within which to search for objects. It feeds its information forward into a large convolutional neural network (CNN) with 5 convolutional layers and 2 fully connected layers. The CNN extracts features from those regions in the image that were marked as candidates for object search. The final module takes those features as input and feeds them to a set of support vector machines to perform classification. At the time of publication, the authors report significant improvements on the PASCAL VOC 2010 benchmark challenge [55] for object detection. To get a better understanding of their complex model, the authors performed an ablation study where they removed different parts of their system. To their surprise, Girshick et al. found that removing one of the two large fully connected layers of the CNN led to no performance loss at all despite having ablated 29%, about 16.8 million, of the CNN's parameters. Removing both fully connected layers led to a small drop in performance. The authors concluded that most of the CNN's power is rooted in its convolutional layers and that the much larger fully connected layers are somewhat dispensable.

[9] https://twitter.com/fchollet/status/1012721582148550662?lang=en, last accessed 2020-03-03

In the field of clinical text prediction, Spithourakis, Peterson and Riedel [5] ablated their meta-enriched text prediction model to examine the added value of additional and especially numerical patient data. The authors included one categorical (`patient gender`) and 19 numerical meta features that represented the results of medical tests. The authors found that their ablated models were on par and occasionally even outperformed their meta-enriched counterparts marginally with regard to perplexity and keystroke savings in a word completion task. Qualitative inspection of word suggestion ranks produced by meta-enriched as well as ablated models revealed that enriching the model with meta information changed the order in which suggestions appeared. Most of the time the correct suggestion was included in the top 5 predictions, but not on rank 1.

Recently, model ablation has been established as an important tool at the disposal of machine learning researchers to study the behaviour of their neural models. Next to other more elaborate and sophisticated methods such as `quantization` [56], [57] and `distillation` [58], [59], ablation is a popular method for downscaling huge models to make them more suitable for production and to meet the low latency constraints of many real world applications. In the current pilot study, we employ ablation for yet a different purpose. We use `weight pruning` [60], [61] to intentionally decrease FutureType's performance in a controlled fashion to study the effect of prediction accuracy on user reception of FutureType and user performance. In `weight pruning`, individual weight connections between layers are removed. A popular variant of weight pruning is `weight magnitude pruning` which removes weights closest to 0, using weight magnitude as a measure of importance. Connections are removed by setting the respective elements of a weight matrix manually to 0. This procedure does not affect the size of the weight matrix or computational speed, but it does affect model performance. Since we aimed to intentionally downgrade the performance of our FutureType model, we did not need to pay attention to only removing weights that were close to 0 anyway. We pruned a systematic proportion of weights, but we chose the pruned weights randomly.

3.2 Pilot method

Study design

To investigate end user reception of the FutureType prototype and the influence of internally measured model performance on user experience, an experimental between-subjects design was employed. The experiment followed the setup of an A/B test [7]. Participants were assigned to one of eight experimental groups using a custom assignment algorithm described in section 3.2. The independent variable that distinguished one group from another was the FutureType model that provided word suggestions. The dependent variables were calculated from keystroke logs and included: word suggestion acceptance (either using the `Tab` key or by clicking on the suggestion), typing speed, keystroke savings (KS), and unnecessary distractions (UD). With the exception of UDs, the dependent variables measured typing behaviour. UDs are a measure of how distracted typists get by the text prediction feature. The current study received ethical approval from the ethics committee of the Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) at the University of Twente,

Enschede, The Netherlands.

Data collection

The dependent variables for analysis were calculated from raw log event data. Table 9 shows the content of a raw `json` log. Each log falls into one of three event types: a keypress, toggling FutureType (enabled/disabled) or showing a suggestion. `Application` logged the type of Ons application in which the log event was triggered. `Client id` uniquely identified the healthcare patient in whose medical record the typing occurred. `Context` captured completed words in a window of eight preceding words. `Prefix` logs the preceding letters of the current word when the log event was triggered. The `user agent` is a characteristic `String` that lets the server know the application, operating system, vendor, and/or version of a requesting user agent. A user agent is a computer program that represents a person. In the current web context of FutureType, the user agent is a browser.

Table 9: Raw logged keystroke data

Log variable	Data type	Example
Client id	`int`	`1234`
Employee id	`int`	`5678`
Customer code	`String`	`DF1111`
Application	`String`	{`dossier`, `agenda`, `groupcare`}
Timestamp	`int`	`1576561169834`
Event type	`String`	{`keypress-down`, `show-prediction` `toggle-futuretype`}
Key	`String`	`a`, `Tab`, `ArrowRight`, `AltGraph`
Context	`String`	`vannacht onrustig geslapen. Dhr heeft last van`
Suggestion	`String`	`kriebelhoest`, `''`
Prefix	`String`	`kri`
User agent	`String`	`Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:73.0) Gecko/20100101 Firefox/73.0`

Participants

In total, 7063 healthcare professionals participated in the FutureType pilot study. We gathered no demographics on our participants because they were irrelevant for the purpose of the study. No inclusion or exclusion criteria were applied, except for the implicit inclusion criterion that all participants were employed at a healthcare organisation that uses the Ons software package of Nedap Healthcare. In total, 10 customer organisations participated in the pilot, working in three different healthcare sectors: elderly care, disabled care and mental care.

Procedure

Customer organisations took the initiative in joining the FutureType pilot. The pilot was advertised in two ways. First, an announcement was placed simultaneously in two web environments: a) Basecamp[10], an online environment used by the account management department of Nedap Healthcare and its support staff to communicate with customer healthcare organisations about new features and b) the support portal of Nedap Healthcare. The FutureType pilot was added to the `Pilot` section of the portal. Both communication channels included a description of FutureType and some usage instruction. The information brochure is included in Appendix A. Second, the pilot was announced at the customer day, an annual gathering of Nedap's customer healthcare organisations at Nedap. During the customer day, software developers present their current and future ideas and discuss the latest software releases with end users and application managers. FutureType was pitched during the 2019 edition of the customer day. Except for the information brochure, no direct communication took place between researchers and research participants. We have no further knowledge of whether and how customer organisations discussed participating in the pilot with their employees internally. When a customer organisations took the initiative to join the pilot, we assumed that the employees at the organisation consented to participation.
Once a pilot customer joined the pilot, FutureType was enabled on their production environment, meaning that it was enabled in the software environment that is used in real work settings. Since FutureType was designed to disrupt the normal workflow of healthcare professionals as little as possible, no explicit briefing sessions were scheduled. Once FutureType was activated on production, the text input fields in the Nedap Ons Dossier, Agenda and Groepszorg applications were enhanced with the feature for all employees.
Due to performance considerations with respect to the backend of FutureType, pilot customers were added gradually. The first two customers joined the pilot in late September 2019. On January 17th 2020, FutureType was deactivated for all production environments. In total, the pilot was conducted during 102 days (14 and a half weeks). However, the runtime of the pilot varied per pilot organisation.

FutureType models

Eight different FutureType models were used in the pilot. None of them were enriched with meta information described in chapter 2. We ablated the FutureType model described in chapter 2. In particular, we systematically set proportions of its weights to 0. The proportions we tested varied between 0% and 56% with a step size of 1%. The ablated weights were chosen randomly. Only weights between the LSTM output nodes and the last dense output layer were ablated. The effect of ablation on model accuracy was examined for predicting the next word based on a context window size of 8 preceding words and a word prefix of length 2 for the current word. The ablation experiments were conducted on the same 1000 randomly sampled sequences from the test set that were used to evaluate the trained FutureType model as described in chapter 2.

[10] https://basecamp.com, last accessed 2020-03-27

Table 10: Accuracy scores of the eight ablated FutureType models that were used as experimental conditions in the FutureType pilot. Accuracy scores were calculated for predicting the next word given a context window of eight preceding words and a word prefix length of 2. Ablated accuracy scores were calculated for the same test set as the FutureType model was originally evaluated on (see chapter 2).

	Ablation fraction (%)	Test accuracy
$M_{0.57}$ (Baseline)	0	0.574
$M_{0.55}$	0.09	0.545
$M_{0.52}$	0.17	0.516
$M_{0.48}$	0.24	0.476
$M_{0.45}$	0.29	0.454
$M_{0.42}$	0.35	0.424
$M_{0.39}$	0.42	0.386
$M_{0.37}$	0.46	0.365

The best performing ablated model had an accuracy of 57.4%, the worst performing model an accuracy of 32%. For the experimental pilot groups, we chose eight models with linearly descreasing accuracy in the range of 58% to 38%. We deemed a difference of 20% in model accuracy as an ethically responsible performance range for investigating the effect of internally measured performance differences, without actively disturbing the work of healthcare professionals too much. The linear step size for choosing the "next worst" model was a rounded decrease of 0.03 in accuracy and ablated models were chosen on the nearest fit. Table 10 shows an overview of the final set of experimental FutureType models. It is remarkable how robust FutureType is to ablation, given that accuracy "only" dropped by about 20% when 46% of all weights were randomly set to 0.

An algorithm assigned each pilot participant one of the eight models. Once assigned, the same model was used for the participant throughout the pilot. Based on `employee ids`, the algorithm distributed the eight models for each pilot customer. From a reputational and marketing perspective, we wanted to give as many end users as possible the best performing model. At the same time, we had to ensure that sufficiently many participants were assigned the less well performing models to be able to compare experimental groups during analysis. The following pseudo code illustrates how 72% of all employees in an organisation were assigned the best performing model (`baseline`), while each of the ablated models were assigned to 4% of all users. Essentially, a `batch` number is calculated by taking a participant's `employee id` modulo 100 and then checking the percentile in which the resulting number falls.

```
algorithm get_model(employee_id)
  """ Returns a FutureType model based on the employee number """

  batch <- employee_id mod 100

  if batch < 72
      model <- baseline
  else if batch < 76
      model <- model_2
  else if batch < 80
      model <- model_3
  else if batch < 84
      model <- model_4
  else if batch < 88
      model <- model_5
  else if batch < 92
      model <- model_6
  else if batch < 96
      model <- model_7
  else
      model <- model_8

  return model
```

Listing 2: Pseudo algorithm for assigning a model to a research participant

Plan of analysis

Data preprocessing

We preprocessed the raw keystroke logs in several ways. First, all logs, distributed over three servers, were collected in one single `json` file. Logged events were either a keystroke, activating/deactivating FutureType, or showing a prediction. Second, logs had to be sorted based on `user id` and `timestamp`. A unique intermediate sorting key was used to sort the raw logs, a `String`, consisting of `user id`, `customer code` and `timestamp`. `User ids` were unique per customer, but not across customers. Therefore, it was necessary to incorporate `customer codes` to form unique sorting keys. Once the raw logs were sorted, they were ready for simplification. Preprocessing included

- Finding suggestion acceptances that were accepted by `clicking` on the suggestion rather than using `Tab`. Extracting `click` acceptances was based on shown suggestions and the expected next word context (i.e., the word context at the previous timestep plus the shown suggestion).
- Extending the set of relevant event types. After simplification, log event types included `special`, `key`, `show suggestion`, `no suggestion available`, `delete`, `accept suggestion (click)`, `accept suggestion`, `navigation`, `toggle off`, `toggle on`. The `key` event was assigned to keys of length 1, such as alphanumeric characters and punctuation, but also symbols like ¶ were included by filtering on the length of the keypress event. `Special` was assigned to keypress events that did not fall into any other event type. In addition, it was assigned whenever the key that triggered the log event could not be identified, literally producing the `String Unidentified`. `No suggestion available` occurred when the server did receive a client request for a suggestion and it did reply with a

suggestion, but the suggestion was intercepted. Pruned suggestions were intercepted.
- Reducing the amount of `String` objects to minimize the size of the final data frame. Only `contexts`, `prefixes` and `suggestions` were kept that occurred in a `show suggestion` event.
- Assigning a `session id` and `word id` to each log event. A typing session was defined as any continuous period of typing, interrupted by a gap in consecutive event timestamps of more than 60 seconds. `Word ids` enabled analyses such as counting the number of suggestions received for one and the same word. They also simplified counting words.

Preprocessing was performed in steps of pilot customers. All logs originating from the same customer were preprocessed at a time. The preprocessed logs were saved on disk, in one `csv` file per customer. To analyze the keystroke log data, all customer `csv` files were read into a single `Pandas`[11] data frame, in total amounting to 64.8GB of keystroke data. Each row in the data frame represented a (simplified) log event and contained the following information:

- Timestamp
- Model
- Application
- User id
- Type
- Customer code
- Employee id
- Client id
- User agent
- Key
- Session id
- Word id
- Context
- Prefix
- Suggestion

Preprocessing was performed from within a Jupyter Notebook[12], running on a secured GPU server owned by Nedap Healthcare, using Python 3.6.7[13] and Debian/Linux [14] for intermediate sorting of log events[15].

A major filter that was later applied to the preprocessed keystroke data was based on whether a log event had occurred on a mobile device or not. Filtering for mobile devices was performed by filtering the case insensitive `user agent String` for markers of mobile user agents. The following `substrings` were used as markers: `mobile`, `tablet`, `android`, `iphone`, `ipad`, and `blackberry`. If a `user agent` field contained any of these substrings, the log event was classified

[11] https://pandas.pydata.org/, last accessed 2020-03-27
[12] https://jupyter.org/, last accessed 2020-03-27
[13] https://www.python.org/, last accessed 2020-03-27
[14] https://www.debian.org, last accessed 2020-03-27
[15] Here, I would like to acknowledge the major support I received from my colleagues at the Data Science team of Nedap Healthcare in preprocessing the raw log data - many thanks, this would not have been possible without your help!

as originating from a mobile device. It is possible that this rather simple filter missed some mobile log events. Our `substring` choice was based on regular expression suggestions on Github[16] for identifying mobile user agents.

Comparing ablated FutureType models

The primary goal of the keystroke analysis is to compare how multiple FutureType models that differ in their intrinsically evaluated performance impact the typing and usage behaviour of end users.
One simple way of comparing our eight ablated models is to look at the amount of accepted suggestions they yielded. Note that this comparison can only be made if our experimental groups are comparable with regard to the amount of time people spent typing using each model and the amount of users that were assigned to our models. Since our model assignment was biased to assign the best performing model to most pilot participants, we extracted a random sample of 300 users that were assigned the `baseline` model in our between-model comparisons. The sampled subset is denoted as $\mathrm{M}_{0.57}(\textit{downsampled})$ or `baseline (downsampled)` when it is directly compared to the full `baseline` dataset. In all other cases, `baseline` and $\mathrm{M}_{0.57}$ refer to the downsampled dataset in subsequent analyses. The rest of the experimental groups did not need to be downsampled.

A formal way of assessing the performance of a text prediction model in a real world task is to look at its `precision` and `recall` when used by end users. Spithourakis, Petersen, and Riedel [5] and Bickel, Haider, and Scheffer [62] note that `keystroke savings` (KS), the percentage reduction in key presses compared to character-by-character text entry, corresponds to a `recall` metric. Likewise, the average number of unaccepted character suggestions that the user has to scan before completing a word, called `unnecessary distractions` (UD), corresponds to a `precision` metric. KS and UD are calculated as follows:

$$\mathrm{KS} = \frac{\mathrm{keys}_{\text{unaided}} - \mathrm{keys}_{\text{with prediction}}}{\mathrm{keys}_{\text{unaided}}} \tag{29}$$

$$\mathrm{UD} = \frac{\sum \text{string length}_{\text{accepted suggestions}}}{\sum \text{string length}_{\text{all suggestions}}} \tag{30}$$

Typing performance

In real world settings, models can also be compared by how they impact end users on relevant metrics. In the context of text prediction models, typing performance is a relevant metric. We calculated one measure from the raw keystroke logs to approximate typing performance for our participants: typing speed in `words per minute (WPM)`. In accordance with Wobbrock [63], typing speed is calculated based on all keystrokes that contribute to the text *result*, not to the *process* of writing text, like so

$$\text{Words per minute (WPM)} = \frac{|T| - 1}{S} \times 60 \times \frac{1}{5} \tag{31}$$

[16] https://gist.github.com/dalethedeveloper/1503252, last accessed 2020-03-27

Hence, $|T|$ in equation is the length of the final transcribed `String` that may be composed of letters, numbers, punctuation, spaces and other printable characters, but not for instance backspaces. S is the number of seconds starting from entering the first character to the last, including keystrokes related to the text entry *process*. The -1 in the numerator is crucial since the preparation time for executing the very first keystroke of the entered text is not included in S as it occurs *before* the first keystroke is executed [63]. Multiplying with 60 and dividing by $\frac{1}{5}$ converts the measure from characters per seconds (CPS) to WPM, the latter being more commonly reported in literature. The conversion is according to Wobbrock [63] and MacKenzie [64] and subdivides the string of characters into words of length 5 as a rule of thumb.

For calculating WPM, the keystrokes and total time for the unit of time of interest, in our case typing sessions, were summed and equation 31 was applied to the summed keystrokes and time in seconds, as if dealing with one very long typing session. A typing session was defined as all subsequent keystroke events cut off from the next event by a break of more than 60 seconds. That way, only actual typing time was included in the calculation. Sessions with only one keystroke event were excluded from the calculation, as were sessions that did not contain printable characters, i.e. letters or punctuation, but only `special` keys like `Shift`, `Unidentified`, or `ArrowDown`. In our calculation of typing speed, we explicitly distinguished between keystrokes gathered on mobile devices and keystrokes gathered on laptops or desktops. While keystroke data collected on mobile devices with an attached physical keyboard are comparable with keystroke data on laptops and desktops, keystrokes collected on touch screens are not comparable. Since we could not reliably distinguish between mobile users with physically attached keyboards and mobile users typing on a touch screen, we excluded all mobile keystrokes from calculating typing speed.

3.3 Results

Descriptives

Table 11 summarizes basic descriptives on the keystroke log data. The high number of accepted suggestions by clicking is remarkable. About 98% of all accepted suggestions were accepted by clicking on the suggestion. The number of acceptance clicks by users who used FutureType on a mobile device cannot solely explain why so few suggestions were accepted using the intended method, the `Tab` key. Not all mobile device keyboards have a `Tab` key. Therefore, one would expect most suggestion acceptances using the click method to originate from keystrokes logs on mobile devices. The data does not confirm the expectation as only about 20% of all click acceptances were performed on a mobile device. This means that 80% of all suggestion acceptances using the click method originate from laptops or desktops. One explanation is that users did not know that word suggestions could be accepted using the `Tab` key and therefore, mostly used the click method.

Table 11: Keystroke log descriptives. The subset of logs that was gathered on mobile devices is highlighted to show what proportion of suggestion acceptances were collected on mobile devices.

	N Total	N Mobile	%
Number of users	7063	2486	35.2
Number of log events	176.320M		
Number of keystrokes	108.490M		
Number of words	17.3748M		
Number of sessions*	365 274		
Number of hours spent typing	13 607.9		
Number of suggestions shown	52.853M	9.388M	17.76
Number of accepted suggestions	483 843	95 548	19.75
Number of accepted suggestions (`Tab`)	7331	811	11.06
Number of accepted suggestions (click)	476 512	94 737	19.88
Number of keystroke savings	1.711M	200 688	11.73

* A session is defined as a period of continuous typing. Sessions are interrupted by pauses. The threshold for determining the end of a session was set to 60 seconds.

Comparing ablated FutureType models

Table 12 shows basic descriptives for the keystroke logs gathered for each of the eight ablated FutureType models. `Baseline` represents the best performing model and the rest of the models the next best performing model in descending order. Since the model assignment algorithm was biased to assign the `baseline` model to most pilot participants, subsequent analyses were executed on a random sample of 300 users for the `baseline` group, denoted as $M_{0.57}$downsampled in Table 12. Once the `Baseline` group is downsampled, all model groups are comparable with regard to the number of users and the amount of typing. Hence, experimental groups are suitable for comparison.

Table 12: Keystroke log descriptives dependent on model

	N users	Number of keystrokes	Number of words	N hours spent typing
$M_{0.57}$ (Baseline)	5113	77.831M	12.408M	9795
$M_{0.57}$ (downsampled)	300	4.694M	744 985	613.8
$M_{0.55}$	306	4.108M	663 690	529.2
$M_{0.52}$	272	4.157M	669 701	494.6
$M_{0.48}$	278	4.748M	766 529	590.7
$M_{0.45}$	275	4.554M	737 660	565.3
$M_{0.42}$	255	4.025M	651 007	513.2
$M_{0.39}$	285	4.656M	771 355	564.8
$M_{0.37}$	279	4.408M	706 188	555.0

Table 13: Word suggestion acceptances dependent on model

	N users	N suggestions shown	N accepted suggestions	N accepted suggestions (Tab)	N accepted suggestions (click)	N median accepted per user	Keystroke savings (recall)	Unnecessary distractions (precision)
$M_{0.57}$	300	2.298M	21 841	248	21 593	38.5	0.434	0.009
$M_{0.55}$	306	2.031M	18 813	108	18 705	32.5	0.398	0.009
$M_{0.52}$	272	2.122M	17 643	406	17 237	35.5	0.387	0.008
$M_{0.48}$	278	2.320M	20 245	92	20 153	35.0	0.424	0.009
$M_{0.45}$	275	2.263M	18 683	238	18 445	37.0	0.397	0.008
$M_{0.42}$	255	1.881M	17 897	44	17 853	29.0	0.405	0.009
$M_{0.39}$	285	2.308M	19 717	299	19 418	38.0	0.387	0.008
$M_{0.37}$	279	2.244M	18 798	557	18 241	38.0	0.367	0.008

Accepted word suggestions and deactivating FutureType

Table 10 shows statistics on how many word suggestions were shown and accepted for each model. In addition, Table 6 shows how many suggestions were accepted on average by a user in each model group (`N median accepted per user`). There seems to be no remarkable difference between models with regard to the total number of word suggestions accepted. However, in some groups, slightly more suggestions are accepted using the `Tab` key than in other experimental groups. The least `Tab` acceptances occur in the group of $M_{0.42}$ ($N = 44$) and most in the group of $M_{0.37}$ ($N = 557$). Still, compared to the total number of word suggestions accepted in both groups, these differences are small.

Further analysis investigates the individual suggestion acceptance behaviour of participants, depending on their assigned model group. Figure 9 shows boxplots of the distribution of accepted suggestions per user, normalized by the amount of words a user has typed. For this analysis, only users that have typed at least 200 words in total were included. Participants who typed less than 200 words were regarded as unrepresentative for the sample: healthcare professionals whose daily work activities include medical report writing. Three extreme outliers were excluded from Figure 9 for the sake of readability. One outlier had an $\frac{\text{suggestion acceptance}}{\text{typed words}}$ ratio of more than 1, the other two of more than 0.8. These extreme ratio scores are possible if a user has typed more words than the final report included. Our calculation of the number of typed words only takes words into account that are in the final result, not words that were typed as part of the typing process. If a user deleted a large proportion of their typing in the process, the number of acceptances were still counted, but words were only counted for the final text product. Figure 9 shows that the average $\frac{\text{suggestion acceptance}}{\text{typed words}}$ ratio per user lies around $2 - 3\%$. This shows that most users accepted few word suggestions with regard to the amount of typing they did throughout the pilot. However, for each model, a small group of users (between $2 - 5\%$) produces more or less extreme outliers, with $\frac{\text{suggestion acceptance}}{\text{typed words}}$ ratios ranging between 10% and 55%. This trend is constant for all models.

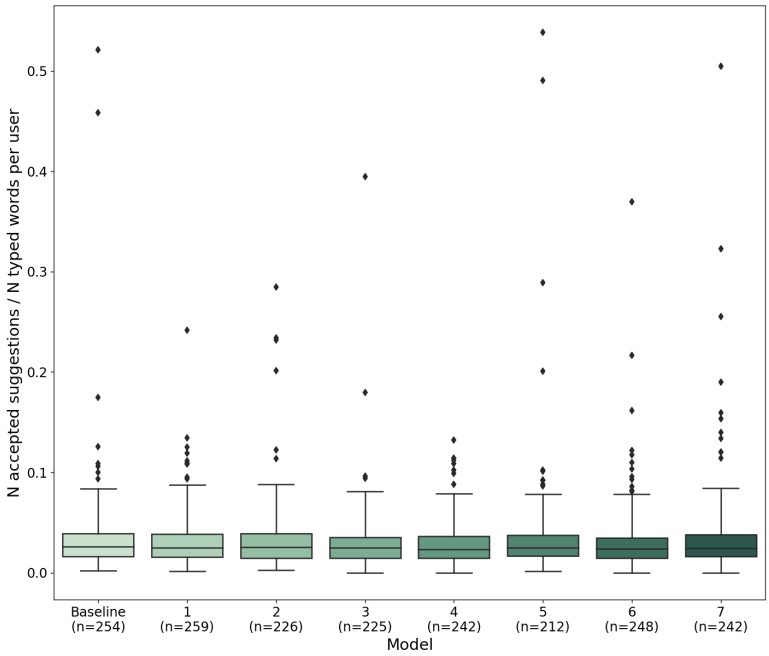


Figure 9: Number of word suggestion acceptances per user for each FutureType model, normalized by the amount of words a user typed. Except for the baseline model, all models were represented as numbers in the plot. 1 corresponds to $M_{0.55}$, 2 to $M_{0.52}$ and so on. Boxplots were drawn with box borders representing the 75th (Q3) and 25th (Q1) percentile respectively. The whiskers were drawn according to Hoaglin and Iglewicz [65], they indicate data points at Q3+2.2∗IQR and $Q1 - 2.2 * IQR$ respectively, with IQR (inter quartile range) = Q3 − Q1. Only users that had a total typed word count of at least 200 were included in the analysis.

If there is a select group of enthusiast FutureType users, there may also be a comparable group at the other end of the spectrum: users that deactivate FutureType extremely more often compared to the average pilot participant. Figure 10 shows boxplots of the distribution of the number of deactivation events per user for each model. Again, only users who typed at least 200 words during the pilot were included in the analysis. No outliers were removed. For all model groups, the median number of `toggle off` events is very small, either 1 or 1.5. The $75th$ percentile of all groups lies at about 16 deactivations per user. However, for each model group, there is a large subset of users (between $16 - 22\%$ of all participants) that deactivates FutureType a lot more often than the average participant.

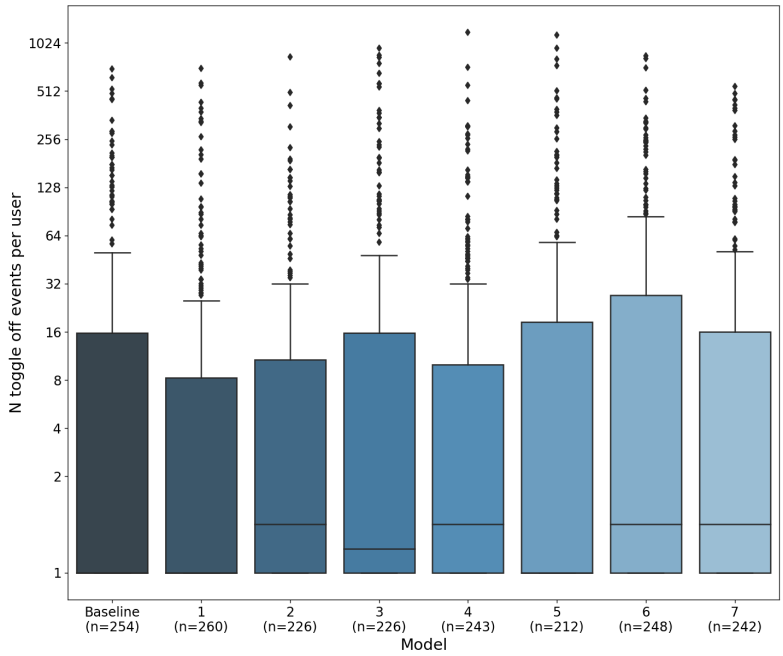


Figure 10: Number of `toggle off` (deactivation) events per user for each FutureType model. Except for the baseline model, all models were represented as numbers in the plot. 1 corresponds to $M_{0.55}$, 2 to $M_{0.52}$ and so on. Boxplots were drawn with box borders representing the 75th (Q3) and 25th (Q1) percentile respectively. The whiskers were drawn according to Hoaglin and Iglewicz [65]. Only users that had a total typed word count of at least 200 were included in the analysis.

Further analysis shows that participants from two out of the ten participating customer organisations deactivated FutureType more often than participants from other customers. Figure 11 shows boxplots of the number of `toggle off` events per user for each participating customer organisation. Again, only users who typed at least 200 words during the pilot were included in the analysis.

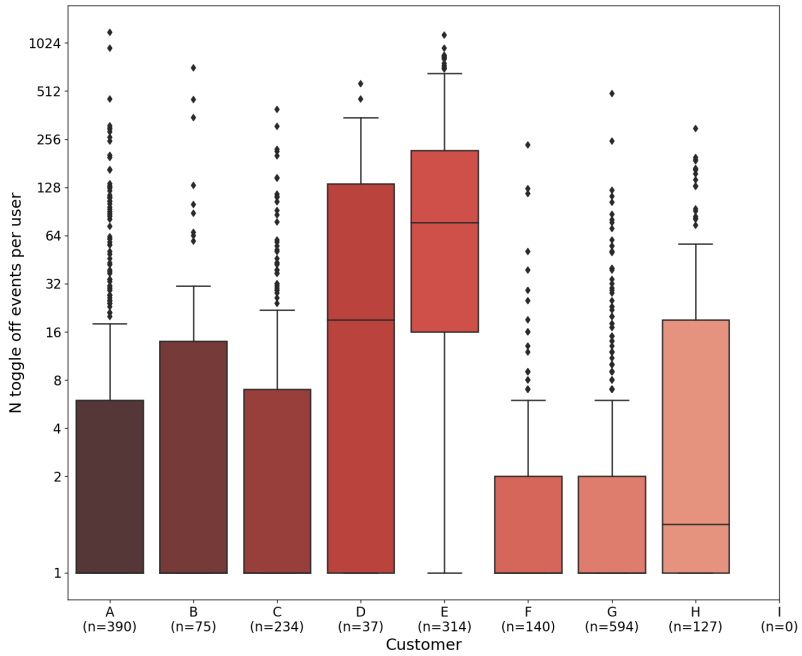


Figure 11: Number of `toggle off` (deactivation) events per user for each customer who participated in the pilot. Customers were anonymized with uppercase letters. Boxplots were drawn with box borders representing the 75th (Q3) and 25th (Q1) percentile respectively. The whiskers were drawn according to Hoaglin and Iglewicz [65]. Only users that had a total typed word count of at least 200 were included in the analysis. As a result, the two only participants from customer `I` were filtered and no boxplot could be constructed for the organisation.

It appears that participants at customer organisations `D` and `E` systematically deactivated FutureType more often than participants working at the rest of the pilot customers. Apart from that, the same pattern is visible as when comparing the number of `toggle off` events between models. The median number of deactivation events is very small, either 1 or 1.5, while each customer group has a distinct group (about $15 - 19\%$, for customers `D` and `E` about 8%) of outliers that activated FutureType a lot more often.

Finally, about half of all participants neither `toggled` FutureType `off`, nor did they `toggle` FutureType `on`. Table **??** shows for each model, how many participants did neither activate or deactivate FutureType during the pilot. For this overview, only participants who typed at least 200 words throughout the pilot were included. This suggests that half of all pilot users did not experiment with FutureType at all. It may also offer an explanation for the low median number of deactivations for all model groups and customers depicted in Figures

10 and 11. Participants who did not use the toggle feature were evenly spread across experimental groups. Their behaviour is thus unlikely to be related to the model they were assigned.

Table 14: Number of participants per model who made no use of the `toggle on`/`toggle off` feature. Only participants who typed at least 200 words during the FutureType pilot are included in the overview.

	N total	N no toggle events	%
$M_{0.57}$ (Baseline)	300	162	54.0
$M_{0.55}$	305	162	53.1
$M_{0.52}$	272	144	52.9
$M_{0.48}$	277	159	57.4
$M_{0.45}$	275	137	49.8
$M_{0.42}$	255	146	57.3
$M_{0.39}$	285	141	49.5
$M_{0.37}$	279	140	50.2

User performance

Apart from comparing usage data depending on the FutureType model they were assigned, the impact of the ablated models on a user's typing performance is examined. Here, typing performance is approximated with typing speed. Figure 12 shows boxplots of the typing speed in words per minute (WPM) for each model. When summing the total typing time and typed characters across typing sessions, as we do in our calculation for typing speed (see Section 3.2), there appears to be no difference in speed between model groups. For each model group, the median WPM is about 22 WPM, and the fastest typists within the $75th$ percentile typed about 50 words per minute.

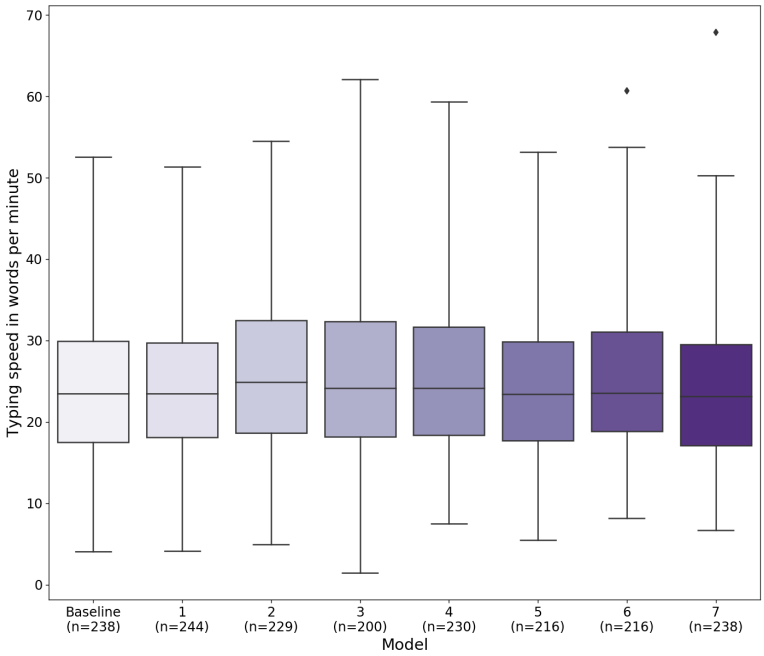


Figure 12: Typing speed of individual pilot participants in words per minute (WPM) for each model. Except for the baseline model, all models were represented as numbers in the plot. 1 corresponds to $M_{0.55}$, 2 to $M_{0.52}$ and so on. Boxplots were drawn with box borders representing the 75th (Q3) and 25th (Q1) percentile respectively. The whiskers were drawn according to Hoaglin and Iglewicz [65]. Only users that had a total typed word count of at least 200 were included in the analysis. Keystrokes typed on mobile devices were excluded from the analysis as explained in Section 3.2.

3.4 Discussion

For a software developing company such as Nedap Healthcare, one way of supporting healthcare professionals in their evermore demanding work due to an aging population and the lack of healthcare professionals on the Dutch (and global) job market, is building their software for efficient and ease of use. To this end, Nedap Healthcare puts effort into developing FutureType, a textual auto completion tool for medical records.
Motivations for the FutureType pilot described in this chapter were rooted in commercial as well as scientific interests. The pilot had two main objectives. First, FutureType needed validation in real work settings, with real users. Our second research question (**RQ2**) concerns the impact of internally determined performance differences of text prediction models on end users. If internally measured performance differences using common metrics such as prediction accuracy turn out to matter little for the end user, the suitability of evaluation tasks and metrics researchers use to evaluate text prediction models may be challenged. Their evaluation methods may match poorly with what real application settings demand of text prediction models. After all, text prediction applications in real life settings are likely to profit from a more holistic evaluation process, taking both, a system's utility and usability into account [9].
To this end, we set up a FutureType pilot involving ten customer organisation of Nedap Healthcare. All customers were healthcare organisations providing either intra- or extramural care in one of three healthcare sectors in the Netherlands: elderly care, disabled care, and mental care. We devised eight experimental conditions, featuring eight ablated FutureType models in a large scale A/B test, spanning 14 and a half weeks in total. Keystroke log events were collected and formed the basis for analysis.

The conducted analyses yielded two main insights. First, pilot participants likely did not know how to use FutureType properly. Second, our eight ablated FutureType models performed equally "well" in the examined analyses, suggesting that their internally determined performance differences did not matter for our end users.

FutureType usage throughout the pilot

Usage of the `toggle` functionality and how users accepted word suggestions suggest that our pilot participants did not know how to use FutureType properly. The results show that, independent of the model they were assigned, half of all participants never used the designated `thunder symbol` in the user interface to activate or deactivate FutureType. While this can in theory mean that half of all participants were satisfied enough with FutureType as it is, our combined results suggest otherwise. In particular, we see that most accepted word suggestions were accepted by `clicking` on the suggestion. In the current FutureType release, this way of accepting suggestions was intended for users on mobile devices because mobile keyboards do not necessarily include a `Tab` key. That is, accepting suggestions using `Tab` is cumbersome, if not impossible on certain mobile devices, when no physical keyboard is attached. However, accepting by `clicking` was never intended as the main way of accepting suggestions when a physical keyboard is available. `Clicking` is much less efficient, even detri-

mental for typing performance because the user has to switch attention and finger movements from keyboard to a mouse cursor. Since acceptance using `clicks` is less efficient, one would expect users that had a physical keyboard at their disposal to refrain from using them. Unless they did not know how to accept suggestions otherwise. Further analysis confirmed this intuition. The proportion of suggestion acceptances using the `click` method that originated from mobile devices could not fully account for the unexpected high number of `click` acceptances. Only a fifth originated from mobile devices and thus, it is very likely that our pilot participants did not know that they could accept suggestions with `Tab`. However, it would be interesting to investigate how often a word suggestion that was not accepted was typed over by a user. The suggestion may not have been accepted because the user did not know how to accept it or because pressing the `Tab` key or `clicking` on the suggestion was perceived as more effortful. However, in this way FutureType would still contribute to reducing the amount of spelling mistakes. We name this until now unconsidered use case for FutureType `FutureType as a spellchecker`. FutureType as a spellchecker should definitely be explored in the future as yet another potential use case for the word prediction tool.

Since accepting word suggestions with a `click` is less efficient, its high prevalence in our pilot data may partly explain why our users typed much slower compared to a recent and very deatiled study of 136 million keystrokes by Dhakal, Feit, Kristensson, and Oulasvirta [48]. Despite the undeniable differences between our study and the typing tasks devised by Dhakal et al., differences in average typing speed in words per minute (WPM) are striking. Dhakal et al. investigated typing performance in brief transcription tasks and their sample was based on teenagers and young adults that volunteerd to take part in a commercially advertised typing test. On average, their participants typed 51.56 WPM with a standard deviation of 20.20 WPM. Our fastest typists typed about 50 WPM, and the average 22 WPM of our sample aligns with the slowest 10% in Dhakal et al. [48]. Clearly, the methodological differences make a direct comparison difficult, but the magnitude of the difference is striking. The inefficient use of FutureType by accepting suggestions with `clicks` instead of `Tab` may have contributed to the slow typing that is prevalent in our sample. However, by choosing a cut-off of 60 seconds between typing sessions, we may also have contributed to the slow typing speed in our sample. 60 seconds was a safe, but possibly excessive cut-off to distinguish periods of continuous typing from a typing break. By failing to reliably identify the onset of typing breaks, we may have included time when people already stopped typing in the calculation of their typing speed. Consequently, it is possible that we underestimated the typing speed of our participants. Future work should experiment with shorter session cut-offs. Shorter session cut-offs produce more sessions, but are less likely to unjustly include time when people take a break from typing in the calculation of speed. We could find no theoretical or empirically based guidelines regarding cut-offs between typing sessions and breaks. It is likely that such an ideal cut-off differs from sample to sample and between typing tasks.

It remains unclear why our users did not know that they could accept word suggestions with `Tab`. How to accept word suggestions and how to (de)activate Fu-

tureType with the `thunder symbol` was explained in the information brochure. It appears that most end users did not read the instructions in the manual. In general, we do not have any insight into how participation in the FutureType pilot was communicated within customer organisations, whether the pilot and the usage of FutureType was openly discussed or simply decided by people at the top of the organisational hierarchy. This observation is included and further discussed in a set of general practical observations with regard to conducting a large scale A/B test with real end users at the end of this thesis report. Since it is unclear how using a new feature is communicated downward within customer organisations, for future releases of FutureType, we highly recommend making its usage (even more) self-explaining so that instructions outside the direct application context are rendered superfluous. In practice, this means that the application context should explain how to use FutureType. This can be achieved by putting very brief textual instructions, such as `Accept word suggestions with Tab`, close to text fields for which FutureType is enabled or as default into the text field and remove the default once the user starts typing. Alternatively or additionally, during the first one or two visits of the application after FutureType is activated, a pop up with a short instruction video could explain the two main functionalities of FutureType visually: accepting suggestions with `Tab` and (de)activating the feature for a text field and as a global setting.

Impact of FutureType model ablation

On the basis of the normalized distribution of word suggestion acceptances across experimental conditions, the usage of the (de)activation feature and user performance, our results suggest that there is no main effect of experimental condition. In other words, it appears that it did not matter which ablated FutureType model we assigned to an end user.

Our results revealed promising `keystroke savings (KS)`, regardless of the assigned model. The calculation of KS may have profitted from the complexity of the accepted word suggestions. That is, if users tend to write many complex, long words as it is the case in medical reports, achieving promising keystroke savings is easier, especially when users accept suggestions after having typed only a few characters. Even if relatively few suggestions are accepted, if accepted suggestions are long words, the count of saved keystrokes increases fast. If our accepted suggestions are indeed mainly complex, medical terms, our results emphasize the utility of a good auto completion tool for medical reports. Follow-up analyses should confirm whether accepted word suggestions tend to be complex. Our ablated FutureType models performed much worse with regard to the second formal metric we used to evaluate model performance, `unnecessary distractions (UD)`. While the relation between KS and UD is as much a balancing act as the relation between `recall` and `precision` in virtually all machine learning settings, the FutureType models we tested in the pilot scored strikingly poorly on UD. One explanation for this was given in chapter 2. FutureType is biased towards predicting long words and in fact, is unable to predict words that are shorter than six letters by design. If our users typed many words that are shorter than six letters, they were bound to receive many irrelevant suggestions. Apart from enabling FutureType to also predict shorter words which comes with its own complications such as a much larger output

layer, additional optimizations for presenting and timing word suggestions can help to reduce the amount of UDs.

One way of reducing UDs is putting more emphasis on an ideal delay for presenting word suggestions. Word suggestions should only be delivered when needed, otherwise they impose more cognitive load on users than they can possibly redeem by saving keystrokes. As it is now, it appears that our pilot participants were bombarded by unnecessary suggestions. As a follow-up, we recommend investigating the typing speed of our users more closely in terms of characters per `milliseconds`, preferably under conditions where a) additional overhead from accepting suggestions with `clicks` is minimized and b) shorter cut-offs are employed to reliably distinguish between typing sessions and breaks. By doing so, suggestion delays can be optimized depending on the typing speed of our users. In addition, UDs can be reduced by investigating the moment when users are most likely to accept a suggestion, that is, the prefix length at which users are most likely to accept a suggestion. We expect that such a tendency is closely related to the model's prediction accuracy at different prefix lengths. As we saw in chapter 2, the performance of the baseline FutureType model (depicted as $M_{0.57}$ in the current chapter) increases sharply when three instead of two letters of the target word are known. Consequently, we expect that users are more likely to accept suggestions at a prefix length of three than when two letters of the target word are known. However, there are probably other factors at play as well. If FutureType fails to predict the correct word early enough, that is, at small prefix lengths, the effort of switching from spelling the word to pressing `Tab` outweighs typing the word oneself. This is even more relevant when users accept suggestions with a mouse `click` because clicking on a suggestion represents a greater and more effortful switch than hitting the `Tab` key. Yet another way of reducing UDs is to employ a probability threshold for next word suggestions which enforces a minimum certainty before suggesting a word. This way, less incorrect suggestions should be presented to the user. Reducing unnecessary distractions is crucial for limiting the cognitive overhead introduced by FutureType for our users.

Our results show that the performance of our tested ablated FutureType models is comparable with regard to keystroke savings, unnecessary distractions, and user reception. On the one hand, this could mean that the evaluation methods researchers use without involving end users in meaningful downstream tasks align poorly with the requirements for real application settings. On the other hand, one can argue that the discriminative power of the tested ablation factions was too small. As outlined, while chosing an appropriate maximum for degrading the performance of FutureType, we had to take ethical considerations into account. Degrading the performance of a software feature to the extent that it obstructs healthcare professionals in their daily work would have been unacceptable from an ethical point of view. In addition, FutureType's robustness to ablating as much as 46% of weights connecting the LSTM output with the dense output layer and the results we obtained from the pilot suggest that our chosen method of ablation may be ineffective. This may be the case because the output layer of FutureType has a very high dimensionality (39 074), enabling it to predict many words. However, not all words that FutureType can predict may be used by our end users. Some words are used much more frequently than

others, which limits the detrimental effect of removing infrequent words from the output layer. In addition, if the words that were removed have frequently occurring synonyms, their removal is also likely to go unnoticed by our end users. As a result, pilot participants may have noticed very little of the performance differences as determined by internally measured prediction accuracy.

In addition, it is entirely possible that we could not find a difference in user reception and performance depending on intrinsic model performance differences because our users struggled too much with other elements, such as the user interface (UI) of FutureType. We set up the A/B test under the assumption that our users would be able to use FutureType as intended, by using the `Tab` key to accept suggestions and knowing how to switch the feature off and on. Since we know that our participants did not understand the feature well, conclusions can only be drawn with great caution.

Limitations

Our collected keystroke data has a number of limitations that caution us to draw conclusions with overly great confidence. The data is flawed in relevant ways. First, we cannot reliably identify periods when FutureType was activated or deactivated on the basis of the log events. While we did log `toggle` events, we discovered that some customer organisations have a policy for deleting browser histories when closing a browser. Due to initial technical difficulties, we saved user preferences for deactivating FutureType in the browser settings, rather than in their profile. As a result, some participants were forced to deactivate FutureType whenever opening a new browser, for which FutureType was activated by default. We have no logs of FutureType activations by virtue of opening a new browser, ergo, we cannot reliably determine based on our logged `toggle` events whether FutureType was active or not. Ideally, to reliably evaluate the impact of FutureType on user performance, we would have wanted to only include periods when FutureType was indeed activated in our calculations. However, this was impossible for our collected log data.

In addition, our inability to cache user preferences regarding FutureType (de)activation was likely the source of major frustration for a select group of pilot participants. It also impaired the extraction of a control group from the keystroke logs. Ideally, we would have enlisted an additional customer as a control group to compare user performance with and without FutureType in a clean fashion. When this turned out to be unfeasible, first due to commercial considerations and later due to time restrictions, we planned to extract a group of pilot participants from the initial pilot group. We would have argued that users that decided that FutureType is not for them and therefore, deactivated the feature early on, were suitable enough as a control group. However, since we cannot reliably distinguish between periods when FutureType was activated or deactivated, we had no means of extracting a control group from the log data. In the future, from a scientific perspective, a control group should be secured by all means prior to the pilot kick off.

Another limitation of our collected log data is that we cannot reliably distinguish between mobile devices with an attached physical keyboard and mobile devices

with no attached keyboard. This makes grouping keystroke events based on (non-)mobile devices less reliable and forced us to exclude all mobile agents from performance analyses. Luckily, we could minimize the impact of this flaw on the reliability of our conducted analyses by excluding all mobile user agents and by virtue of having collected large amounts of data, which made log events collected on mobile devices somewhat dispensable for drawing the bigger picture of user performance depending on experimental condition.

3.5 Concluding remarks FutureType pilot

Testing software in parallel with development in an iterative fashion is crucial for building user-friendly and meaningful software products. The current chapter documents the first validation FutureType, a word completion tool for medical records which is currently under development at the Dutch technology company Nedap. A team of developers is processing the insights from this pilot as the current thesis report is written. Our pilot study had two main objectives. First, validating FutureType with end users in real work settings. Second, investigating the suitability of common text prediction evaluation metrics used as a proxy for model performance in downstream tasks with real end users. Regarding the first objective, at this point, we can conclude that future iterations of FutureType will profit considerably from putting usage information directly into the application context of the feature. Regarding the second objective, results suggest that evaluation metrics used for internal model evaluation capture the demands of real application settings for text prediction insufficiently. Indeed, our study participants struggeled so much with the user interface of FutureType that drawing reliable conclusions about how our eight ablated FutureType models performed is difficult, which only confirms that a model that performs well on intrinsic metrics is not guaranteed to be useful for end users. As a result, we attribute only more importance to Nielsen's advice to view a *useful* system as a two-sided coin, including the system's `utility` and `usability` [9]. We acknowledge that conclusions can only be drawn with utter caution since our collected keystroke log data is flawed and since it is likely that the devised method for degrading model performance was somewhat inefficient for the studied downstream task, the composition of medical record entries. The conducted pilot contributes important insights for future pilots with regard to improved logging and pitfalls of large scale A/B testing for scientific purposes.

4 Conclusion

The current thesis research set out to support the development of FutureType, a word completion tool for medical report writing, from the persepctive of both pillars that make up a system's usefulness: `utility` and `usability`. The first chapter of this research focuses on exploring extensions to the recurrent neural network (RNN) architecture of FutureType, with the aim of improving the model's prediction performance. The quality of predictions is an important measure of the `utility` of a text prediction system. We demonstrate the capacity of a number of meta features concerning medical reports, such as the expertise of the employee who writes the report and the type of the report, for identifying systematic vocabulary choices. We thereby conclude with respect to **RQ1**: "What is the potential of meta information about the patient, the author or the medical report itself to increase FutureType's prediction accuracy for word completions?" that some of our examined meta features have high potential for enriching FutureType, while others appear to be less suitable. One meta feature with high potential is the `type` of the report, while two features with less potential are the `age` and `gender` of the patient about whom the report is written. We leave it to future work to investigate whether feeding meta information explicitly to FutureType increases its predictive capacity, but we did provide a roadmap for incorporating meta features into RNN language models.

The second chapter of this thesis focuses on FutureType's `usability`. In addition, it investigates **RQ2**, the extent to which common intrinsic evaluation methods for system `utility` align with downstream utility when the system is used by end users. To this end, we conducted a large scale end user evaluation, collecting keystroke data from ten customer organizations of Nedap Healthcare and more than 7000 healthcare professionals. The results from the FutureType pilot perpetuate the importance of approaching the development of software from a utility as well as from a usability perspective. Our participants struggled much with the usability of FutureType, which made assessing the impact of intrinsically measured utility differences on our end users difficult. We draw any conclusions with great caution, but our results suggest that differences in prediction accuracy had little influence on how FutureType was used by our pilot participants. We provide recommendations for the future development of FutureType regarding its utility and usability and we conclude this thesis report with a number of practical observations that we believe to capture some of the pitfalls of conducting research involving large scale user testing and big data analysis in the healthcare domain.

5 Practical observations

While working on the current thesis research, we experienced a number of complications along the journey and some plans turned out differently than expected. We gained some important insights about conducting applied research in healthcare and the specific methods we employed. Apart from our recommendations for the future development of FutureType at Nedap Healthcare and the scientific contribution of this thesis, we summarize a number of practical observations that we hope will guide future researchers that choose similar research methods.

- A first set of observations is about conducting large scale pilots with real end users as part of a thesis project. As a developer of a software feature which may well be at the heart of your thesis research, you may want to develop in close cooperation with end users. In reality, this is may be difficult because real end users have busy schedules and organisational policies may not work in your favour. An example is a scaffolded feedback loop between developers and end users, involving account managers at your own software company and application managers at customers. To prevent ineffective feedback loops and misunderstandings, you should do your very best to make sure that your product design incorporates all relevant information for usage so that indirect communication lines are used as little as possible.
- Research and industry do not always align well. While working on FutureType, we often noticed that building text prediction models for research purposes is not necessarily the same as building models for production. For example, from a language model perspective and for the sake of comparability with benchmarks, FutureType should be evaluated in specific ways which may be completely irrelevant for its use at Nedap Healthcare. At times, this puts additonal workload on you because you will want to satisfy the demands of both worlds. Setting up a user study is one example of conflicting interests between industry and research. One example from the current research is how we assigned participants to one of the eight ablated FutureType models. From a scientific perspective, you would want to randomly assign participants to their condition and to assign equally many participants to each condition. From a marketing and ethical perspective, it is unwise to assign more end users than necessary an intentionally inferior product. Often, a compromise or clever alternatives need to be found as we did in the current study. We assigned the best performing model to most participants and downsampled the overrepresented experimental condition during analysis.
- A final set of observations is about collecting keystroke log data on a large scale and the subsequent analyses. If keystroke log data is part of your project, you need to know exactly what to log in advance. Moreover, having an idea what to log is unfortunately not enough, you should sit down and try to conduct the analyses you intend to do with a set of dummy log data that exactly represents the logs that you would be collecting, based on the set of log requirements you came up with. While this may sound overly careful, it really is the only way to figure out whether you

generate logs for all events you need for your analysis. Also consider how logging is integrated in the service of interest. Consider where you save your user dependent settings and how you can protect logs and settings alike from sources of failure from outside your logs. We overlooked external sources of failure in the current FutureType pilot which eventually made it impossible to reliably infer periods in which FutureType was (in)active. Keystroke log data is messy which is why you need to chose your filters wisely. Make sure you collect all necessary information for applying your chosen filters later on and document your filtering decisions well. They can make the difference between a reliable and an unreliable analysis.

Be aware that there is little you can conclude with certainty from keystroke logs with a single analysis. Keystroke logs are puzzles. You will often need to cross-check the results you retrieve from one analysis with results from another analysis. For example, our log data showed that half of all users did not use the `toggle` feature. Log data however, does not tell you why users did not use the feature. They might love FutureType so much that they never felt like switching it off. But it is more probable that they simply did not know how to use the `toggle` button since it better fits the general picture of the log data, i.e. users clearly did not know how to accept suggestions with `Tab`, ergo, it is probable that they did not understand the `toggle` button either. Analysing keystroke log data is very time-consuming because one always needs to cross-reference conclusions by conducting several analyses that together form a tiny piece of the puzzle. It also makes it difficult to not lose yourself in details. Our advice is to conduct keystroke analyses in small steps. Start with very simple descriptives to get a feeling for the data and the answers you can and cannot get by analysing your data. Sometimes, you will need to abandon a planned analysis because the result of your data exploration is that you cannot conduct the intended analysis reliably with the data you collected. But that does not mean that your data does not contain interesting results. Sometimes, you will find patterns in your data that you did not expect. Therefore, to a certain extent, let the data guide your analysis, depending on the questions it can and the questions it cannot answer.

References

[1] M. Hanekamp, S. Heesbeen, I. v.d. Helm, and R. Valks, "Administratieve belasting langdurige zorg 2019 [online]", Berenschot, 2019, Available at https://www.berenschot.nl/actueel/2019/september/administratieve-belasting/, last accessed 2020-03-05.

[2] N. R. Greenbaum, Y. Jernite, Y. Halpern, S. Calder, L. A. Nathanson, D. A. Sontag, and S. Horng, "Contextual autocomplete: A novel user interface using machine learning to improve ontology usage and structured data capture for presenting problems in the emergency department", *BioRxiv*, p. 127 092, 2017.

[3] Y. Gong, L. Hua, and S. Wang, "Leveraging user's performance in reporting patient safety events by utilizing text prediction in narrative data entry", *Computer methods and programs in biomedicine*, vol. 131, pp. 181–189, 2016.

[4] M. Sevenster, R. van Ommering, and Y. Qian, "Algorithmic and user study of an autocompletion algorithm on a large medical vocabulary", *Journal of biomedical informatics*, vol. 45, no. 1, pp. 107–119, 2012.

[5] G. P. Spithourakis, S. E. Petersen, and S. Riedel, *Clinical text prediction with numerically grounded conditional language models*, 2016. arXiv: 1610.06370 [cs.CL].

[6] A. Yazdani, R. Safdari, A. Golkar, and S. R. N. Kalhori, "Words prediction based on n-gram model for free-text entry in electronic health records", *Health information science and systems*, vol. 7, no. 1, p. 6, 2019.

[7] S. Firmenich, A. Garrido, J. Grigera, J. M. Rivero, and G. Rossi, "Usability improvement through a/b testing and refactoring", *Software Quality Journal*, vol. 27, no. 1, pp. 203–240, 2019.

[8] J. Eng and J. M. Eisner, "Informatics in radiology (info rad) radiology report entry with automatic phrase completion driven by language modeling", *Radiographics*, vol. 24, no. 5, pp. 1493–1501, 2004.

[9] J. Nielson, "Usability 101: Introduction to usability [online]", 2012, Available at https://www.nngroup.com/articles/usability-101-introduction-to-usability/, last accessed 2020-03-01.

[10] B. L. Monroe, M. P. Colaresi, and K. M. Quinn, "Fightin'words: Lexical feature selection and evaluation for identifying the content of political conflict", *Political Analysis*, vol. 16, no. 4, pp. 372–403, 2008.

[11] J. Lu, M. Henchion, and B. MacNamee, "Extending jensen shannon divergence to compare multiple corpora", in *25th Irish Conference on Artificial Intelligence and Cognitive Science, Dublin, Ireland, 7-8 December 2017*, CEUR-WS. org, 2017.

[12] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult", *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[13] T. Mikolov and G. Zweig, "Context dependent recurrent neural network language model", in *2012 IEEE Spoken Language Technology Workshop (SLT)*, IEEE, 2012, pp. 234–239.

[14] J. R. Bellegarda, "Exploiting latent semantic information in statistical language modeling", *Proceedings of the IEEE*, vol. 88, no. 8, pp. 1279–1296, 2000.

[15] N. Coccaro and D. Jurafsky, "Towards better integration of semantic predictors in statistical language modeling", in *Fifth international conference on spoken language processing*, 1998.

[16] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank", 1993.

[17] J. Li, M. Galley, C. Brockett, G. P. Spithourakis, J. Gao, and B. Dolan, "A persona-based neural conversation model", 2016. arXiv: 1603.06155 [cs.CL].

[18] X. Chen, T. Tan, X. Liu, P. Lanchantin, M. Wan, M. J. Gales, and P. C. Woodland, "Recurrent neural network language model adaptation for multi-genre broadcast speech recognition", in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[19] A. Jaech and M. Ostendorf, "Low-rank rnn adaptation for context-aware language modeling", *Transactions of the Association for Computational Linguistics*, vol. 6, pp. 497–510, 2018.

[20] ——, *Personalized language model for query auto-completion*, 2018. arXiv: 1804.09661 [cs.CL].

[21] A. B. Dieng, C. Wang, J. Gao, and J. Paisley, *Topicrnn: A recurrent neural network with long-range semantic dependency*, 2016. arXiv: 1611.01702 [cs.CL].

[22] J. Tang, Y. Yang, S. Carton, M. Zhang, and Q. Mei, "Context-aware natural language generation with recurrent neural networks", *arXiv preprint arXiv:1611.09900*, 2016.

[23] Y. Zheng, G. Chen, M. Huang, S. Liu, and X. Zhu, "Personalized dialogue generation with diversified traits", *arXiv preprint arXiv:1901.09672*, 2019.

[24] H. Zhou, M. Huang, T. Zhang, X. Zhu, and B. Liu, "Emotional chatting machine: Emotional conversation generation with internal and external memory", in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[25] Z. Yin and Y. Shen, "On the dimensionality of word embedding", in *Advances in Neural Information Processing Systems*, 2018, pp. 887–898.

[26] S. Arora, *Word embeddings: Explaining their properties*, https://www.offconvex.org/2016/02/14/word-embeddings-2/, Accessed 2020-28-02, 2016.

[27] M. Mahoney, "Large text compression benchmark", 2011, Accessed 2020-01-03.

[28] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information", *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[29] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, "Advances in pre-training distributed word representations", *arXiv preprint arXiv:1712.09405*, 2017.

[30] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov, "Learning word vectors for 157 languages", *arXiv preprint arXiv:1802.06893*, 2018.

[31] T. Mikolov, Q. V. Le, and I. Sutskever, "Exploiting similarities among languages for machine translation", *arXiv preprint arXiv:1309.4168*, 2013.

[32] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[33] M. I. Amsterdam, *Netwerk naamkunde*, http://www.naamkunde.net, Accessed: 2020-20-02.

[34] J. W. Ratcliff and D. E. Metzener, "Pattern-matching-the gestalt approach", *Dr Dobbs Journal*, vol. 13, no. 7, p. 46, 1988.

[35] P. E. Black, *Ratcliff/obershelp pattern recognition*, in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed. Available from: https://xlinux.nist.gov/dads/HTML/ratcliffObershelp.html, accessed 2020-20-02, 2004.

[36] D. Jurafsky and J. H. Martin, *Speech and language processing*. Pearson London, 2014, vol. 3.

[37] P. Eckert, "Age as a sociolinguistic variable", *The handbook of sociolinguistics*, pp. 151–167, 2017.

[38] S. Kullback and R. A. Leibler, "On information and sufficiency", *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[39] R. J. Gallagher, A. J. Reagan, C. M. Danforth, and P. S. Dodds, "Divergent discourse between protests and counter-protests:# blacklivesmatter and# alllivesmatter", *PloS one*, vol. 13, no. 4, e0195644, 2018.

[40] J. Lin, "Divergence measures based on the shannon entropy", *IEEE Transactions on Information theory*, vol. 37, no. 1, pp. 145–151, 1991.

[41] E. Sober, "The principle of parsimony", *The British Journal for the Philosophy of Science*, vol. 32, no. 2, pp. 145–156, 1981.

[42] J. Vandekerckhove, D. Matzke, E.-J. Wagenmakers, *et al.*, "Model comparison and the principle of parsimony", *Oxford handbook of computational and mathematical psychology*, pp. 300–319, 2015.

[43] L. Damodaran, "User involvement in the systems design process-a practical guide for users", *Behaviour & information technology*, vol. 15, no. 6, pp. 363–377, 1996.

[44] S. Kujala, "User involvement: A review of the benefits and challenges", *Behaviour & information technology*, vol. 22, no. 1, pp. 1–16, 2003.

[45] R. Kohavi and R. Longbotham, "Online controlled experiments and a/b testing.", *Encyclopedia of machine learning and data mining*, vol. 7, no. 8, pp. 922–929, 2017.

[46] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne, "Controlled experiments on the web: Survey and practical guide", *Data mining and knowledge discovery*, vol. 18, no. 1, pp. 140–181, 2009.

[47] K. Trnka, J. McCaw, D. Yarrington, K. F. McCoy, and C. Pennington, "User interaction with word prediction: The effects of prediction quality", *ACM Transactions on Accessible Computing (TACCESS)*, vol. 1, no. 3, pp. 1–34, 2009.

[48] V. Dhakal, A. M. Feit, P. O. Kristensson, and A. Oulasvirta, "Observations on typing from 136 million keystrokes", in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–12.

[49] S. K. Kane, J. O. Wobbrock, M. Harniss, and K. L. Johnson, "Truekeys: Identifying and correcting typing errors for people with motor impairments", in *Proceedings of the 13th international conference on Intelligent user interfaces*, 2008, pp. 349–352.

[50] A. M. Feit, D. Weir, and A. Oulasvirta, "How we type: Movement strategies and performance in everyday typing", in *Proceedings of the 2016 chi conference on human factors in computing systems*, 2016, pp. 4262–4273.

[51] X. Liu and J. P. Bagrow, "Autocompletion interfaces make crowd workers slower, but their use promotes response diversity", *arXiv preprint arXiv:1707.06939*, 2017.

[52] D. Anson, P. Moist, M. Przywara, H. Wells, H. Saylor, and H. Maxime, "The effects of word completion and word prediction on typing rates using on-screen keyboards", *Assistive technology*, vol. 18, no. 2, pp. 146–154, 2006.

[53] E. Wong, "Ablation", in *Encyclopedia of Clinical Neuropsychology*, J. S. Kreutzer, J. DeLuca, and B. Caplan, Eds. New York, NY: Springer New York, 2011, pp. 6–6, ISBN: 978-0-387-79948-3. DOI: 10.1007/978-0-387-79948-3_3. [Online]. Available: https://doi.org/10.1007/978-0-387-79948-3_3.

[54] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[55] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge", *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[56] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding", *arXiv preprint arXiv:1510.00149*, 2015.

[57] R. Cheong and R. Daniel, "Transformers. zip: Compressing transformers with pruning and quantization", Technical report, Stanford University, Stanford, California, 2019. URL https ..., Tech. Rep., 2019.

[58] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network", *arXiv preprint arXiv:1503.02531*, 2015.

[59] R. Tang, Y. Lu, L. Liu, L. Mou, O. Vechtomova, and J. Lin, "Distilling task-specific knowledge from bert into simple neural networks", *arXiv preprint arXiv:1903.12136*, 2019.

[60] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network", in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[61] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks", *arXiv preprint arXiv:1902.09574*, 2019.

[62] S. Bickel, P. Haider, and T. Scheffer, "Predicting sentences using n-gram language models", in *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, 2005, pp. 193–200.

[63] J. O. Wobbrock, "Measures of text entry performance", in *Text entry systems: Mobility, accessibility, universality*, San Francisco: Morgan Kaufmann, 2007, pp. 47–74.

[64] I. S. MacKenzie, "A note on calculating text entry speed", *Unpublished work. Available online at http://www.yorku.ca/mack/RN-TextEntrySpeed.html*, 2002.

[65] D. C. Hoaglin and B. Iglewicz, "Fine-tuning some resistant rules for outlier labeling", *Journal of the American statistical Association*, vol. 82, no. 400, pp. 1147–1149, 1987.

Appendix A: FutureType information brochure

Tekstvoorspelling in rapportages (FutureType) - Pilotfunctionaliteit

Tekstvoorspelling in rapportages (FutureType) - Pilotfunctionaliteit

Deze functionaliteit zit nog in de pilotfase, daarom is dit nog niet voor iedereen zichtbaar. Neem contact op met je accountmanager voor meer informatie over deelname aan de pilot.

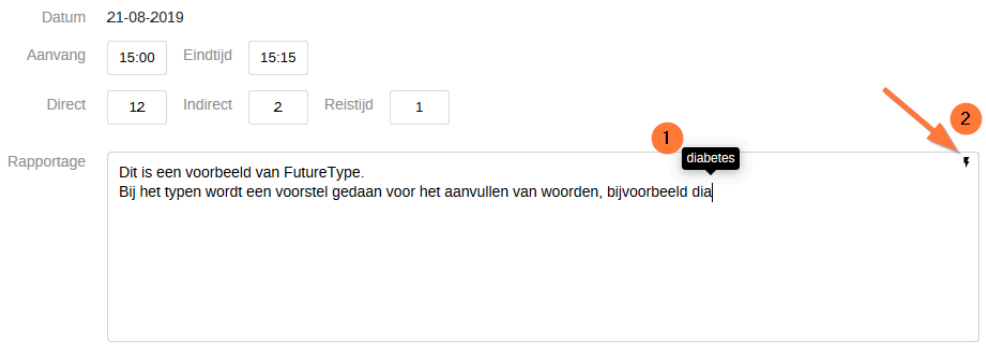
Ons FutureType is een hulpmiddel voor het eenvoudiger invoeren van tekst.

Gebruikers worden geholpen met het invoeren van complexe medische en (zorginhoudelijke) terminologie. De functionaliteit lijkt op het slimme toetsenbord van een smartphone. Ons FutureType wordt geactiveerd in Dossier, Agenda en Groepszorg.

De suggesties worden gebaseerd op de bestaande rapportages in Ons. Er is geen koppeling met een (medisch) woordenboek.

De voorgestelde woorden zijn altijd een suggestie. Als je niet actief de suggestie selecteert, zal er niets veranderen in het rapporteren.

Bij het typen van een rapportage of registratie in een veld met Ons Futuretype zullen suggesties voor het schrijven van woorden verschijnen:



Appendix A: FutureType information brochure

Tekstvoorspelling in rapportages (FutureType) - Pilotfunctionaliteit

Zodra je begint met typen, kunnen voorspellingen verschijnen (1). Door op de **tab-toets** te drukken wordt je woord automatisch aangevuld op basis van de suggestie.

Op een touchscreen werkt het aanvullen van de suggestie door er op te klikken.

Uitgangspunten:

- Suggesties worden gebaseerd op basis van bestaande rapportages.
- De suggesties houden rekening met de context. In verschillende rapportages kunnen andere suggesties getoond worden bij dezelfde beginletters.
- Suggesties kunnen per gebruiker verschillen, ook als dezelfde tekst ingevoerd wordt.

Het is mogelijk om FutureType per applicatie uit te zetten. Klik hiervoor op het icoontje rechtsbovenin (2).

Het aan- of uitzetten van FutureType wordt opgeslagen in je browser. Als je in een andere browser (op een ander apparaat) een rapportage bewerkt, kan de instelling anders zijn.