

Master-Thesis

**“Building a Text Classification Model  
in a sparse and noisy Data Environment”**

Andreas Barth  
Matrikelnummer: 88408

Hochschule Albstadt Sigmaringen  
Fakultät für Informatik  
Studiengang Data Science  
Poststraße 6, 72458 Albstadt

In Kooperation mit  
BMW Financial Services - BMW Bank GmbH

Erstprüfer: Prof. Dr. Goran Glavaš (Universität Mannheim)  
Zweitprüfer: Prof. Dr. Simone Paolo Ponzetto (Universität Mannheim)

26.02.2021

### **Eigenständigkeitserklärung**

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet, sowie die aus fremden Quellen direkt oder indirekt übernommenen Stellen und Gedanken als solche kenntlich gemacht habe.

Diese Arbeit wurde noch keiner anderen Prüfungskommission in dieser oder einer ähnlichen Form vorgelegt. Sie wurde bisher auch nicht veröffentlicht.

Ich bin damit einverstanden, dass die vorliegende Arbeit von den Prüfern in elektronischer Form mit entsprechenden Softwaretools auf Plagiate überprüft werden kann.

Donauwörth, 26.02.2021

Andreas Barth

## Table of Content

1	Introduction .....	1
1.1	Use Case: BMW Financial Services .....	1
1.2	Challenges .....	2
1.3	Problem definition .....	3
1.4	Outline.....	4
2	Theoretical Background .....	5
2.1	Vector Representations of Text Data.....	5
2.1.1	Weighted Words.....	5
2.1.2	Word Embeddings .....	7
2.2	Selected Machine Learning Models for Text Classification .....	11
2.2.1	k-Nearest-Neighbor Classification .....	11
2.2.2	Logistic Regression.....	12
2.2.3	Support Vector Machines.....	13
2.3	Selected Deep Learning Models for Text Classification.....	15
2.3.1	Key Concepts applied in Deep Learning Models .....	16
2.3.1.1	Loss Function for Multi-Class Classification .....	16
2.3.1.2	Optimizers .....	17
2.3.1.3	Strategies to prevent Vanishing Gradients .....	18
2.3.1.4	Batch Normalization .....	20
2.3.1.5	Regularization strategies.....	22
2.3.2	Recurrent Neural Nets .....	23
2.3.3	Long Short-Term Memory.....	26
2.3.4	Bidirectional Recurrent Neural Networks.....	29
2.3.5	Convolutional Neural Nets .....	29
2.4	Self-Attention & Transformer Networks.....	31
2.4.1	Positional Encodings .....	33
2.4.2	Scaled Dot-Product Attention .....	34
2.4.3	Multi-Head Attention.....	35

<b>3</b>	<b>Data &amp; Preprocessing .....</b>	<b>37</b>
3.1	Procurement of Training Data .....	37
3.2	Anonymization of sensitive Data .....	38
3.3	Explorative Data Analysis (EDA).....	40
3.3.1	Document Types (Labels).....	40
3.3.2	Other Meta Data.....	41
3.3.3	Text Data .....	43
<b>4</b>	<b>Experiments.....</b>	<b>46</b>
4.1	Evaluation & Metrics.....	46
4.2	Machine Learning Approaches.....	46
4.2.1	Baseline Classifiers.....	47
4.2.2	Linear Models .....	48
4.2.3	Non-linear Classification Methods .....	49
4.3	General Training Architecture .....	50
4.4	Deep Learning approaches.....	52
4.4.1	Bidirectional Long Short Term Memory.....	52
4.4.2	Convolutional Neural Nets .....	53
4.4.3	CNN with pre-trained Embeddings.....	53
4.5	Bidirectional Encoder Representations from Transformers .....	54
<b>5</b>	<b>Results .....</b>	<b>56</b>
5.1	Machine Learning Models with BOW/TF-IDF vectorization .....	56
5.1.1	Top Line Results of the Machine Learning Classifiers.....	56
5.1.2	Exploration on the Linear SVM performance.....	58
5.2	Deep Learning Models leveraging Embeddings .....	62
5.3	Transformer Model: BERT .....	64
5.4	Linear SVM versus 2-layered CNN .....	66
5.5	Synthesis – Combining individual Strengths .....	69
5.6	Limitations & possible Improvements .....	70
<b>6</b>	<b>Conclusions.....</b>	<b>74</b>
	<b>Bibliography .....</b>	<b>77</b>

## List of Abbreviations

- 2L-CNN *2-layered Convolutional Neural Net*  
ANN *Artificial Neural Net*  
BERT *Bidirectional Encoder Representations from Transformers*  
BiLSTM *Bidirectional Long Short Term Memory*  
BN *Batch Normalization*  
BoW *Bag of Words*  
CBOW *Continuous Bag-of-Words*  
CNN *Convolutional Neural Net*  
DLP *Deep Learning Platform*  
DNN *Deep Neural Net, Deep Neural Net*  
GloVe *Global Vectors*  
GPU *Graphical Processing Unit*  
GTA *General Training Architecture*  
HSDAP *High Security Data Analytics Platform*  
IDF *Inverse Document Frequency*  
 $k$ -NN  *$k$ -Nearest-Neighbor Classifier*  
LinSVM *Linear Support Vector Machine*  
LogReg *Logistic Regression Classifier, Logistic Regression Classifier*  
LSTM *Long Short Term Memory*  
MVP *Minimum Viable Product*  
NER *Named Entity Recognition*  
NLP *Natural Language Processing*  
OCR *Optical Character Recognition*  
OH *One-Hot-Encoding*  
OOV *Out of Vocabulary*  
OVR *One versus Rest*  
ReLU *Rectified Linear Unit*  
RNN *Recurrent Neural Net, Recurrent Neural Net*  
SVM *Support Vector Machine*  
TF *Term Frequency*  
TF-IDF *Term Frequency Inverse Document Frequency*

## List of Equations

Equation 2-1: Term Frequency-Inverse Document Frequency (TF-IDF) .....	6
Equation 2-2: Skip-gram Objective Function .....	8
Equation 2-3: Loss Function of the GloVe Model .....	10
Equation 2-4: Posterior Probability of a Class C1 & C2 and the Logistic Function $\sigma(\cdot)$ .....	12
Equation 2-5: Cross Entropy Error Function .....	13
Equation 2-6: Hard Margin Linear SVM Classifier Objective Function.....	14
Equation 2-7: Soft Margin Linear SVM Classifier Objective Function .....	15
Equation 2-8: Softmax Activation Function & Categorical Cross Entropy .....	17
Equation 2-9: Gradient Descent.....	17
Equation 2-10: RMSProp algorithm .....	18
Equation 2-11: ReLU and Leaky ReLU .....	19
Equation 2-12: Glorot Initialization.....	20
Equation 2-13: Batch Normalization Algorithm .....	21
Equation 2-15: Forward Propagation in a multi-output RNN with a tanh-Activation .....	25
Equation 2-16: Hidden state computed from front context in a conventional RNN .....	29
Equation 2-17: Hidden state computed from front & back context in a conventional RNN ..	29
Equation 2-18: Basic Convolution .....	30
Equation 2-19: Max Pooling .....	30
Equation 2-20: Positional Encodings in the Transformers Model.....	33
Equation 2-21: Scaled Dot-Product Attention .....	35
Equation 2-22: Multi-Head Attention .....	36

## List of Figures

Figure 1-1: BMW Bank DMS.....	2
Figure 2-1: The CBOW and Skip-gram Architecture.....	8
Figure 2-2: Logistic Activation Function .....	19
Figure 2-3: Leaky Rectified Linear Unit Function .....	20
Figure 2-4: RNN with no Outputs.....	24
Figure 2-5: Archetypes of different RNN Architectures .....	24
Figure 2-6: Hidden State in a regular RNN / Memory Cell within an LSTM .....	26
Figure 2-7: Example of 1d-Convolution and Max-Pooling with 3 filters. ....	31
Figure 2-8: Transformer Model Architecture.....	32
Figure 2-9: Scaled Dot-Product Attention in the Transformer Model Architecture .....	34
Figure 2-10: Multi-Head Attention in the Transformer Model Architecture .....	35
Figure 3-1: Distribution of the Top 20 Document Types.....	40
Figure 3-2: Distributions of Numerical Features by Batch Class .....	42
Figure 3-3: Distribution of Word and Character Count.....	43
Figure 3-4: Distribution of Word Count split by Document Type .....	44
Figure 4-1: General Training Architecture.....	51

Figure 4-2: BERT Multilingual & General Training Architecture .....	55
Figure 5-1: Linear SVM: Confusion Matrix for selected Classes.....	60
Figure 5-2: Comparison 2L CNN and Linear SVM: Breakdown by Classes & Support.....	66
Figure 5-3: Comparison 2L- CNN and Linear SVM: Breakdown by Classes & Share of Noise .	68
Figure 5-4: Comparison 2L- CNN and Linear SVM: Top 15 Classes with Superiority .....	68

## List of Tables

Table 3-1: Structure of the raw Data procured from the Production System .....	38
Table 3-2: Distributions of Categorical Features.....	41
Table 3-3: Analysis of the most and least noisy Categories.....	45
Table 4-1: k-NN Classifier with Key Parameter Settings .....	47
Table 4-2: Logistic Regression Classifier with Key Parameter Settings .....	48
Table 4-3: Linear Support Vector Machine with Key Parameter Settings .....	48
Table 4-4: Stochastic Gradient Descent Classifier with Key Parameter Settings.....	49
Table 4-5: Support Vector Machine with Key Parameter Settings .....	49
Table 4-6: Gradient Boosting Classifier with Key Parameter Settings .....	50
Table 4-7: Zero Vectors from Pre-Trained Embeddings.....	54
Table 5-1: Baseline Classifiers Experiments: Weighted F1 Results .....	56
Table 5-2: Linear Model Experiments: Weighted F1 Results .....	57
Table 5-3: Non-Linear Model Experiments: Weighted F1 Results .....	58
Table 5-4: Linear SVM – F1 & Support by Document Type .....	58
Table 5-5: Linear SVM – Feature Importance for selected semantically related Classes .....	61
Table 5-6: Linear SVM – Feature Importance for selected Classes.....	62
Table 5-7: BiLSTMs – Top Line Classification Results .....	63
Table 5-8: CNNs – Top Line Classification Results.....	63
Table 5-9: CNNs with pre-trained Embeddings - Top Line Classification Results .....	64
Table 5-10: BERT Multilingual Cased & Uncased - Top Line Classification Results .....	65
Table 5-11: BERT Multilingual Cased - F1 & Support by Document Type.....	65
Table 5-12: Correlation of Classifier Performance by Class & Document Length .....	67
Table 5-13: True Positives & False Positives for selected Classes.....	69
Table 5-14: Overview Classifier Performance with Ensemble Classifier .....	70

# **1 Introduction**

Managing large volumes of inbound document traffic efficiently is an imperative skill of companies facing millions of customers and other stakeholders. To express their intentions people use various channels, such as email, postal services, facsimiles and increasingly online posts and uploads on the company website. Excellent customer service requires a fast and accurate distribution of these documents to the subject matter experts within the organization. A Document Management System (DMS) can take on this task and support it with automation logic. Artificial intelligence driven solutions can be a vital part of a DMS and potentially improve the level of automation.

## **1.1 Use Case: BMW Financial Services**

BMW Group Financial Services specializes in financing and leasing of automobiles and motorcycles for private, retail and commercial customers. It manages a portfolio of 4 million lease and credit financing contracts across 54 countries. Additional insurance and banking products complement modern mobility solutions for their customers. BMW Bank an entity within BMW Group Financial Services serves a number of regional markets, including Germany, Italy, Spain and Portugal.

BMW Bank implemented a new DMS in 2020 to support daily operations managing inbound document traffic. It seeks to process around 11 million pages per year and strives for classifying up to 80% of this volume fully automatically. The software chosen, a third party commercial solution, is customizable to specific needs and processes of the bank. A cross functional team has been fitting this system with a set of complex lookup logics and rules to enable the classification of inbound documents into document types (i.e. “Vollmacht”). Based on the assigned document type the system can subsequently extract relevant information from the document for further processing (i.e. contract number, account details etc.). Thus the correct and – ideally – fully automatic classification of a document is a vital condition for a seamless processing thereof.

The DMS deals with 320 different formats of documents and maps them to a range of more than 150 document types. Documents that fall short of automatic classification due to insufficient confidence in the mapping are not forwarded to the subsequent departments but transferred to a backlog. This backlog is cleared by a service team that takes over manual inspection and classification. Figure 1-1 schematically summarizes this process.

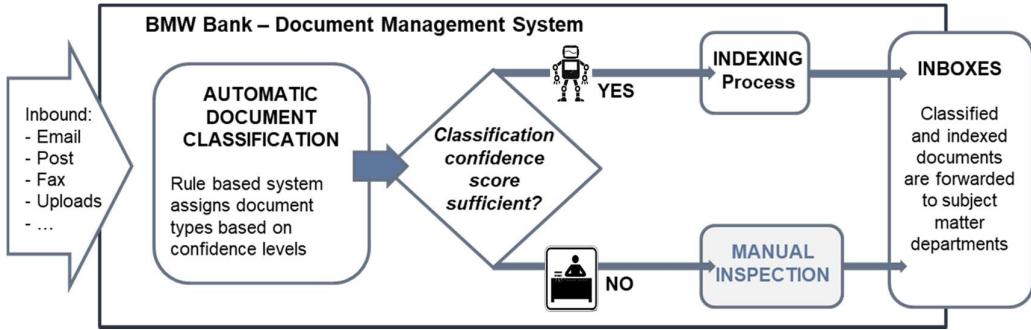


Figure 1-1: BMW Bank DMS

Despite ongoing adjustments and optimization of the underlying rule based logic it's expected at the time of writing that a substantial share of the documents (~25%) will continue to require manual intervention.

BMW Bank wants to explore a proof of concept and learn if an additional machine or Deep Learning driven document classification can mitigate the need for manual intervention, thus reducing operational expenses in a substantial scale.

## 1.2 Challenges

The nature, volume, variety in themes and technical heterogeneity of the received inputs gives rise to a number of challenges for an automatic document classifier:

- The variance in technical formats in which information is provided spans from plain emails to cluttered OCR-scans of id-documents such as passports, driver licenses, vehicle registrations etc.. Thus introducing content with a variety in length of text: From a short abstract (e.g. an email confirmation) to dozens of pages (e.g. legal correspondence or technical assessments on car damages).
- Many documents consist of uploaded forms with manual handwriting on it, introducing textual patterns that are difficult to process in a sequential fashion.
- Documents frequently make reference to previous (not included) communication. Their content makes sense in the context of previous content, making it difficult to comprehend the entire meaning of the document.
- Letterheads, email signatures and boilerplates with legal or administrative copy introduce clutter and make it difficult to determine those relevant parts of text that are most discriminative for a classifier.
- The transactional nature of this data involves a lot of personal information such as contact details, banking data, vehicle and contractual data, etc.. Sensitive personal data requires particular efforts to comply with legislative and corporate information security and data protection requirements. It is therefore necessary

to employ additional Anonymization techniques during preprocessing phase, which add even more dilution to the semantics of the content.

On top of these challenges the communication is rich in domain specific language of the automotive, banking and insurance industry. Thus containing many terms and expressions that might lead to out-of-vocabulary situations when applying general (German) language solutions to this very domain specific classification task.

The aforementioned sensitive nature of the documents with its data protection requirements also restricts preserving and storing data over longer periods. Employing supervised learning techniques requires access to sufficient amounts of labeled training data. But a migration project scaled like the DMS implementation has to be managed within tight budgets and resources. This doesn't allow for extensive additional manual labeling beyond the daily operations. Labeled training data has to be extracted from the productive system, after Go-Live. These factors lead to a limited amount of labeled training data, a sparse data environment for this project.

### 1.3 Problem definition

Automatic document classification is the process of assigning documents an estimated intent, based on a predefined set of categories (document types in this use case). It's a special form of the general text classification task. With the number of labels exceeding the binary case of two classes, it can be described as a multi-class text classification: Every document is finally assigned to one category out of a set with multiple (more than two) categories. Machine learning algorithms learn patterns that allow training data to be mapped to labels. A classifier is deemed sufficient for the given task if it generalizes well enough, thus predicting labels for new, unseen data with a low error rate.

Formally: In a given training set of labeled documents  $\mathcal{D}_{train} = \{(d_1, l_1), \dots, (d_n, l_n)\}$  every document  $d_i$  belongs to a set  $\mathcal{D}$ . Each label  $l_i$  referring to a document  $d_i$  maps to a predefined set of  $m$  categories  $C = \{c_1, \dots, c_m\}$  with  $m > 2$ .

Document classification is the induction of algorithms  $h : \mathcal{D} \rightarrow C$  that are capable of accurately classifying unseen documents, when sufficiently trained on  $\mathcal{D}_{train}$  (Bekkerman, 2004, p. 3).

Building a text classification system requires a number of strategic choices:

- Documents need transformation from unstructured text to a structured, numerical representation for a machine or Deep Learning algorithm to work the math. Many approaches for this have been developed. One can roughly categorize them into weighted word techniques or word embedding methods.

- Choosing the right classification algorithm for the task at hand is another important decision. Candidates range from the more traditional machine learning models to recent state-of-the art Deep Learning or transfer learning approaches that have shown impressive results on a number of downstream tasks.

Advances in the field of Natural Language Processing (NLP) are driven by complex neural network architectures and recently even more so by transfer learning approaches. Given the challenges mentioned above in this sparse and noisy data environment the question arises if those latest recipes automatically provide better results when applied to the task at hand. Especially the limitation on volume of available training data requires an exploration of the best approach: “*In the small data regime, [...] traditional algorithms may or may not do better. For example, if you have 20 training examples, it might not matter much whether you use logistic regression or a neural network; [...] But if you have 1 million examples, I would favor the neural network*” (Ng, 2018, p. 12).

## 1.4 Outline

This project will describe and apply three different approaches, employing several ideas of machine and Deep Learning theory to build a text classification system. It seeks to identify the best approach achieving maximum accuracy for the document classification problem. The report concludes on a deployment recommendation for the proof of concept that BMW Bank is expecting within the scope of this project.

Chapter 2 is introducing key concepts in machine and Deep Learning, providing theoretical background for the experiments applied later on. Particular focus is on the process of encoding textual data and the exploration of classification algorithms, Deep Learning models and transfer learning architectures.

Chapter 3 describes the available data environment, the data extraction and the necessary steps to ready the data for the different techniques applied.

Chapter 4 details on the experiments and the applied techniques within them.

Chapter 5 follows up with reporting the results in detail, looking at particular strengths and weaknesses of different approaches and providing ideas on potential improvements of the methods applied.

Chapter 6 concludes this report with the learnings and provides a recommendation for the “best choice”, for further implementation as a minimum viable product (MVP).

## 2 Theoretical Background

This chapter introduces briefly the theoretical fundament of the applied concepts. First different techniques of encoding text into a numerical format are explored before the different classification approaches are covered. With some abstraction the methods applied are grouped into

- (a) Traditional machine learning approaches
- (b) Deep Learning approaches
- (c) Transfer learning approaches.

For this project models stemming from all three groups are applied and benchmarked against each other. Selected representatives of these groups will be introduced to provide theoretical foundation to these concepts.

### 2.1 Vector Representations of Text Data

Machine learning algorithms apply their math on a fixed sized numerical input of data. Documents present themselves as unstructured data: Text with arbitrary length. For tasks in the NLP domain this requires transforming every textual input into a vector of digits or floating point values. Mathematically these vectors correspond to points in multi-dimensional spaces (Raaijmakers, 2021est., p. 56).

Different strategies exist for encoding text: simple approaches use weighted word schemes, while more sophisticated strategies learn dense representational vectors from the input data in the format of word embeddings (Kowsare, et al., 2019, p. 2).

#### 2.1.1 Weighted Words

One method of encoding text is to leverage word distributions in a given document or in a corpus of multiple documents. The Bag of Words (BoW) model does exactly this. First a vocabulary  $V$  listing every word in the training corpus is generated. With that a term document matrix is created: Rows represent documents and columns represent every word<sup>1</sup> contained in  $V$ . For a classification task with thousands of documents and a vocabulary size  $|V|$  in the tens of thousands the vector space can grow very high dimensional. To observe computational cost  $|V|$  commonly gets reduced to the  $k$  most

---

<sup>1</sup> Term document matrices are often described with rows accounting for words and columns representing documents (Jurafsky & Martin, 2019, p. 100). The transposed form with a matrix the shape of (documents  $\times$  words) resembles the feature space that machine learning algorithms typically expect for input.

frequent words. It's a hyper parameter that can be tuned and experimented with for better results during model training.

One-Hot-Encoding (OH) is the simplest form of a BoW representation. Every word occurrence in a document has a binary representation of one or zero in the vector representing this document.

Instead of a plain OH representation a term frequency (TF) count can add the cardinality of every words' frequency. But common words like articles or propositions are ubiquitously used and found in almost every document. Thus counting their plain frequency provides limited value for distinguishing documents from another: "*Simple frequency isn't the best measure of association between words [...] raw frequency is very skewed and not very discriminative*" (Jurafsky & Martin, 2019, p. 105)

To emphasize words with more distinguishable power, a mathematical weighting scheme combining TF with a terms' inverse document frequency (IDF) is applied (Sparck Jones, 1972).

$$w(t, d) = tf_{d,t} * idf_t$$

$$\text{with } tf_{d,t} = \text{count}(t, d)$$

$$\text{and } idf_t = \log_{10} \left( \frac{N}{df_t} \right)$$

Equation 2-1: Term Frequency-Inverse Document Frequency (TF-IDF)

Every term  $t$  in a document  $d$  is represented by the product  $w(t, d)$  of its term frequency  $tf_{d,t}$  and its inverse document frequency  $idf_t$  (Jurafsky & Martin, 2019, p. 106). The  $idf_t$  term varies inversely with the document frequency  $df_t$ , the number of documents a term is contained in a corpus of total  $N$  documents (Salton & Buckley, 1987, p. 7). Common words in the corpus that are less helpful (like propositions or articles) get their frequency  $tf_{d,t}$  discounted by multiplying the  $idf_t$  term, thus pushing their total weight towards zero.

Even with a limited vocabulary size BoW generally results in a high dimensional but sparse vector model: Every vector presents only a small fraction of non-zero values because the majority of words listed in the vocabulary is not matched in a document.

The BoW approach brings a number of limitations to the document classification task. Classification algorithms processing BoW input have to learn parameters for every dimension in the set feature space. With tens of thousands of dimensions this can result in a highly (over) parametrized classifier inclined to over fitting the training data and consequently poor generalization to unseen data (Jurafsky & Martin, 2019, p. 111).

In a BoW representation every word of the vocabulary is individually represented atomically by its own index. This implies another serious limitation of this approach: BoW models are incapable to capture the sequential order of words in a text or similarities between words:

*“For example, [the] words ‘airplane’, ‘aeroplane’, ‘plane’, and ‘aircraft’ are often used in the same context. However, the vectors corresponding to these words are orthogonal in the bag-of-words model. This issue presents a serious problem to understanding sentences within the model”* (Kowsare, et al., 2019, p. 7).

Those shortcomings are met by another group of encoding techniques:

### 2.1.2 Word Embeddings

Simple count-based methods cannot describe the semantics of a text. But words that co-occur more often together than others can relate to the same semantic concept: *“You shall know a word by the company it keeps”* (Firth, 1957).

The concept of word embeddings is built on this insight. Word embeddings are a buildup of short (the length of hundreds of dimensions), dense vectors with the majority of values being non-zero real numbers. *“It turns out that dense vectors work better in every NLP task than sparse vectors”* (Jurafsky & Martin, 2019, p. 110).

For the document classification problem given in this project, two popular concepts are explored: Word2Vec and Global Vectors (GloVe).

**Word2Vec** (Mikolov, et al., 2013) makes use of a shallow neural network with two hidden layers. Its intuition is that the semantic of a term can be captured by the contextual words found in its neighborhood. Thus words that share the same contexts are more likely to represent a specific semantic concept. Referring to the example above the expressions “aircraft” and “plane” are commonly surrounded by the same context words (i.e. “pilot”, “flying”, or “cockpit”), thus indicating a conceptual similarity between the two.

Word2Vec implements this principle in two variations: The Continuous Bag-of-Words (CBOW) or the Continuous Skip-gram architecture (Mikolov, et al., 2013, p. 4). Both leverage a shallow neural network that is trained on a huge corpus (i.e. a Wikipedia dump), with the objective to optimize a binary classification task. Both techniques are maximizing the likelihood that a target word and some context words in a given window size<sup>2</sup> around the target word co-occur. Given the contextual input words

---

<sup>2</sup> Commonly a window size of 4-5 words is applied (Kowsare, et al., 2019, p. 8)

CBOW predicts the best target word and Skip-gram vice versa predicts the contextual words based on a given input word. Both algorithms are categorized as unsupervised learners, thus they don't require explicitly labeled data for the training. The presence of words within any given real text sequence provides the necessary learning signal.

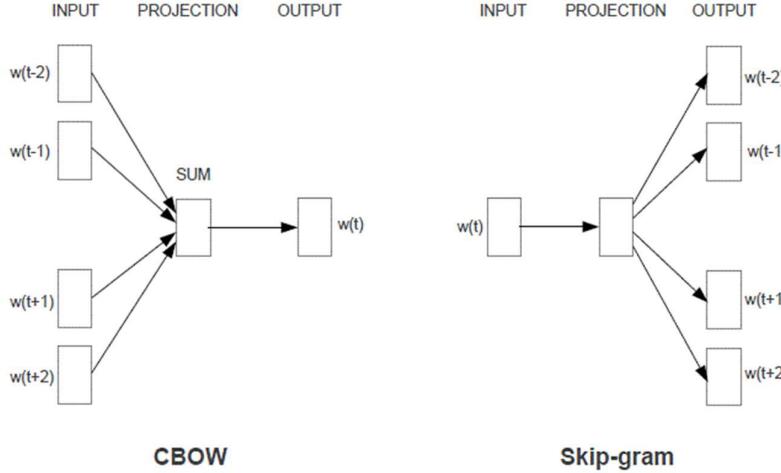


Figure 2-1: The CBOW and Skip-gram Architecture

*“The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word”* (Mikolov, et al., 2013, p. 5). To illustrate the learning process, we focus on the Skip-gram architecture: Based on an initial  $d$ -dimensional embedding vector for every word, the algorithm is optimizing for the maximum similarity between a target word  $t$  and its context word  $c$  (positive examples) and the minimum similarity between  $t$  and a non-context word randomly sampled from the corpus (the negative samples). The objective function of this learning task across the entire training set can be formally expressed as:

$$L(\theta) = \sum_{(t,c) \in +} \log P(+|t, c) + \sum_{(t,c) \in -} \log P(-|t, c)$$

Equation 2-2: Skip-gram Objective Function (Jurafsky & Martin, 2019, p. 114)

With the help of Stochastic Gradient Descent (see 2.3.1.2) the weight vector  $\theta$  is adjusted during training to optimize the sum of the log-likelihood of the positive sample word pairs  $(t, c) \in +$  and the log-likelihood of the negative examples, the randomly generated context-target word pairs  $(t, c) \in -$ . The finally learned weights represented in  $\theta$  are the interesting result. By iteratively optimizing the given objective function the algorithm is pushing the weights of contextually similar words closer to each other and

the weights of dissimilar words apart from each other. The final result is a dense  $d$ -dimensional representation for every term in the vocabulary. Again the vector dimensionality  $d$  is an architectural parameter that can be experimentally chosen or fine-tuned.

Word2Vec produces dimensions of meaning, when sufficiently trained on large enough corpuses. Pre-trained Word2Vec vectors for the German language are available and can be used for encoding training data for the document classification task. Compared to a Deep Neural Net (DNN) that would train with a randomly initialized (plain) embedding matrix the usage of a pre-trained embedding typically is expected to provide reasonable advantage in training time and convergence on better classification results.

Word2Vec delivers on the desired property of semantic learning. But it only takes local contexts into account: “*... they poorly utilize the statistics of the corpus since they train on separate local context windows instead of on global co-occurrence counts*” (Pennington, et al., 2014, p. 1532). Word2Vec streams texts sequentially during training. It doesn’t differentiate between frequent term combinations that are commonly used in conjunction (like in “it is” or “should have”) but possess only little descriptive power and words appearing in conjunction, because they relate to a specific semantic concept (e.g. “aircraft”, “flying” and “pilot”).

Amongst other considerations this drove the development of **Global Vectors**. It’s combining the strengths of the Word2Vec model with those of matrix factorization techniques encoding statistical information about an entire corpus (Pennington, et al., 2014, p. 1532).

The fundamental belief of GloVe is that co-occurrence ratios between two words in a context are strongly connected to meaning. For that GloVe establishes a co-occurrence matrix  $X$  where every value  $X_{ij}$  denotes the number of times a word  $j$  is presented in the context of a word  $i$ . With the vocabulary  $V$  represented in  $X$  the probability  $P_{ij}$  for the co-occurrence of  $i$  and  $j$  can be calculated by dividing  $X_{ij}$  with  $X_i$ , the number of times  $i$  appears in the entire corpus. With all possible  $P_{ij}$  values precomputed it’s easy to determine the ratios of them with another probe word  $k$ .

The intuition of calculating  $\frac{P_{ik}}{P_{jk}}$  with  $P_{ik}$  denoting the probability of  $k$  being in proximity of  $i$  and  $P_{jk}$  the probability of  $k$  being in context of  $j$  is that a large ratio above 1 indicates a much stronger relationship of  $k$  and  $i$  than  $k$  with  $j$ . Contrarily a ratio below 1 describes a weaker relationship of  $k$  and  $i$  than  $k$  with  $j$ . Consequently a ratio close to 1 indicates an equally strong relationship of  $k$  to both words  $i$  and  $j$  or no relationship of  $k$  to both terms. “*Compared to the raw probabilities, the ratio is better able to distinguish relevant words [...] from irrelevant words [...] and it is also better able to discriminate between the two relevant words. The above argument suggests that*

*the appropriate starting point for word vector learning should be with ratios of co-occurrence probabilities rather than the probabilities themselves*” (Pennington, et al., 2014, p. 1534).

Applying a number of constraints, mathematical conveniences<sup>3</sup> and transformations plus two additionally introduced bias terms ( $b_i, \tilde{b}_j$ ) produces the objective function  $J$  in form of a least squares problem with an additional weighting function  $f(X_{ij})$  included:

$$J = \sum_{i,j=1}^V f(X_{ij}) (w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

Equation 2-3: Loss Function of the GloVe Model (Pennington, et al., 2014, p. 1535)

The weighting function  $f(X_{ij})$  ensures some desired behaviors of the loss function:

- $f(0) = 0$ . In the event of two terms not co-occurring ( $X_{ij} = 0$ ) the  $\log X_{ij}$  is defined to negative infinity. For that case the weighing function  $f(X_{ij})$  must take on the value zero to prevent an undesired explosion of values inside the objective function. Multiplying  $f(X_{ij})$  zeroes the term inside the summation.
- Very rare and very frequent co-occurrences shouldn’t be overweighed. For that  $f(X_{ij})$  should be chosen to be non-decreasing and in the latter case be relatively small if  $X_{ij}$  takes on large values.

As with the Word2Vec technique GloVe optimizes the loss function  $J$  during training and by doing so produces vectors of weights that can be extracted. These vectors can serve as initialization weights for embedding matrices used in a neural network designed to solve a downstream task like document classification.

The authors of GloVe proof that their technique produces better embeddings much faster than CBOW or Skip-gram. It’s common practice in many real-world applications to leverage available already pre-trained embeddings (produced on large corpuses with potent computational resources). Hence training time might not be the key advantage of GloVe from a practical point of view. But GloVe is seen as a more principled approach than Word2Vec as it combines a mathematically similar derivation with the potency of global corpus statistics.

---

<sup>3</sup> Since the distance of two words should be interchangeably equal between the two the chosen function for the model should be invariant to exchanges of those roles. The properties of the exponential function  $F = \exp$  cater for this requirement (Pennington, et al., 2014, p. 1534).

However both algorithms share a limitation when applied in a practical real word scenario: Both techniques are not equipped for dealing with out-of-vocabulary (OOV) terms. That is input presenting new, unseen words not known during creation time of the embedding. Those OOV expressions result in a zero-vector within the embedding matrix and zeroed vectors don't contribute anything to the learning task. Hence an embedding matrix with a high ratio of the number of zero vectors to the number of total vectors might hinder a successful employment of these techniques.

This chapter introduced several methods to encode text into a numerical representation using either a BoW approach or leveraging word embedding techniques. Two popular embedding models were introduced to illustrate the general idea of word embedding techniques. In this project the TF-IDF encoding scheme is applied in conjunction with more traditional classification algorithms. Other experiments are using a German version of Word2Vec and Skip-Gram embeddings in combination with neural network architectures. Deep Learning models can start of a plain, randomly initialized embedding matrix and learn more meaningful weights alongside their training for a downstream task. The Deep Learning experiments in this project use this plain approach as well as the employment of pre-trained embeddings. This will be further discussed in chapter 4 (the description of the experiments) and chapter 5 (the results).

With two principal approaches to vectorize textual input for machine learning algorithms introduced chapter 2 continues to explore several algorithms of the machine learning theory (chapter 2.2), the Deep Learning field (chapter 2.3) and the world of Transformers (chapter 2.4).

## 2.2 Selected Machine Learning Models for Text Classification

### 2.2.1 k-Nearest-Neighbor Classification

The *k*-Nearest-Neighbor (*k*-NN) algorithm builds on the idea, that records of the same class share a representation in feature space within close proximity to each other. *k*-NN is a lazy learner: It simply stores the instances of the training data. To classify a new record it produces a ranking with the distances of this new instance to all existing training instances. Based on these distances the top *k* nearest neighbors are chosen and the label is assigned according to the known labels of those *k* neighbors. If different categories are presented in this selection of *k* neighbors, the most frequent category is assigned. The distance of each neighbor to the new record can be accounted for by a weighting scheme, giving higher contribution in the majority vote to training instances with a lower distance accounting for the hypothesis that those records are more predictive than instances more distant to the new record.

$k$ -NN requires a data set of stored records, a distance measure that computes the distance between records and a preset value  $k$ , determining how many neighboring instances should be evaluated for the majority vote when determining the class label.

The choice of the distance function and the  $k$  number of neighbors impact the accuracy of the classification. If  $k$  is chosen too small the model is sensitive to noise and outliers in the training data. Increasing  $k$  can mitigate this but can lead to more indecisive decision bounds. It's common practice to experiment with different distance functions and settings for  $k$  to arrive at the best solution.

$k$ -NN requires the entire data to be stored at inference time. A growing number of instances and a big multi-dimensional feature space, like a TF-IDF matrix, will increasingly drain on computational resources. The distance ranking across the feature space needs to be computed every time a new record arrives for classification. This may lead to unfavorable response times during inference if not adequately met with sufficient resources. Despite this practical limitation in a productive environment  $k$ -NN is applied as one of multiple base line models within this project.

### 2.2.2 Logistic Regression

The Logistic Regression Classifier (LogReg) is a binary classifier that computes the probability of an instance belonging to one specific (positive) class. If the estimated probability exceeds a set threshold (typically defaulted to .5) that data point is predicted to be of the positive class and vice versa if the probability is below the threshold.

Being a member of the Generalized Linear Model family Logistic Regression computes the dot product of input features  $\phi$  and their weights  $w$  and transforms this weighted sum with the Logistic Function<sup>4</sup>  $\sigma(\cdot)$  to present a posterior probability  $p(C_1|\phi)$  as result of the calculation (Bishop, 2006, pp. 205, 197).

$$p(C_1|\phi) = y(\phi) = \sigma(w^T\phi)$$

$$\text{with } p(C_2|\phi) = 1 - p(C_1|\phi)$$

$$\text{and } \sigma(w^T\phi) = \frac{1}{1 + \exp(-w^T\phi)}$$

Equation 2-4: Posterior Probability of a Class  $C_1$  &  $C_2$  and the Logistic Function  $\sigma(\cdot)$

During training of a Logistic Regression model the weight vector  $w$  containing the weights for each single feature is iteratively optimized to minimize an error function.

<sup>4</sup> The Logistic Function is frequently referred to as the Sigmoid Function or simple the Sigmoid.

This error function can be expressed by summation over the negative logarithm of the likelihood for each prediction. For every label  $t$  with  $t_n \in \{0, 1\}$  and  $y_n = p(C_1|\phi)$  the function is calculating the cross entropy  $E(w)$ :

$$E(w) = -\ln p(t|w) = -\sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

Equation 2-5: Cross Entropy Error Function (Bishop, 2006, p. 206)

When predicted probabilities diverge from actual labels the cross entropy (aka log loss) increases. The log loss shown in Equation 2-5 uses a mathematical convenience to combine the two loss functions of both possible cases  $t_n = 0$  and  $t_n = 1$  into one formula: By multiplying each loss term with  $t_n$  and  $(1 - t_n)$  one of the two loss terms always cancels out and the negated summation across the entire training set of size  $N$  produces the total error.

There is no closed form to calculate the optimal state of the values in  $w$ . Thus the algorithm adjusts  $w$  incrementally in multiple iterations to find the optimum delivering the smallest error. The cost function is convex, thus guaranteeing that an optimizing algorithm (like Gradient Descent) will find the best solution provided given enough training time and a sufficient learning rate (Géron, 2019, p. 144).

Logistic Regression strictly is a binary classifier. To counter this limitation when presented a multi-class problem ( $m$  classes with  $m > 2$ ) the chosen software implementation needs to adapt a special strategy under the hood to use Logistic Regression anyway. The “One versus Rest” (OVR) strategy can solve for this case: For every class  $m_i$  a separate Logistic Regression classifier is trained to predict whether a record belongs to  $m_i$  or not. Thus decomposing the multi-class task to an ensemble of  $m$  binary classifiers. The final label for a given record is determined by the classifier yielding the highest probability (Géron, 2019, p. 100).

### 2.2.3 Support Vector Machines

Another supervised classification technique well suited for text classification problems is the Support Vector Machine (SVM). Like Logistic Regression it's a model designed for binary classification but OVR and other strategies can be applied to overcome this binary constraint if a multi-class solution is needed.

For a linearly separable data set “*there may of course exist many such solutions that separate the classes exactly. [...] The support vector machine approaches this problem through the concept of the margin, which is defined to be the smallest distance between the decision boundary and any of the samples, ...*” (Bishop, 2006, p. 326). In  $m$ -dimensional feature space a SVM is fitting a  $(m-1)$ -dimensional hyperplane to the

training data, so that a perpendicular distance between the decision boundary and the nearest data points is achieved. Hence the decision function is defined only by a subset of the data points, aka the support vectors. They define the maximum margin possible for the hyperplane that is fit. This property of a maximum margin classifier is particularly useful as it may reduce the generalization error when the model is applied to new and unseen data later.

A Hard Margin SVM constructs a margin with no allowance for any errors. The hyperplane constructed forces every record to be on the correct side of the decision boundary. For a given set of labeled training data  $\{(t_i, x_i), \dots, (t_n, x_n)\}$  with  $t_n \in \{-1, 1\}$  denoting the positive (+1) and negative class (-1) the decision function must take on values  $> 1$  for all positive instances and  $< 1$  for all instances labeled negative. This can be written as a one line constraint and finding the maximum margin decision boundary can then be expressed as an optimization problem for the weight vector  $w$  and the bias term  $b$  (Géron, 2019, p. 166):

$$\underset{w,b}{\text{minimize}} \frac{1}{2} w^T w$$

subject to:

$$t_n(w^T x_n + b) \geq 1, \quad \text{with } n = 1, \dots, N.$$

Equation 2-6: Hard Margin Linear SVM Classifier Objective Function

A hard margin classifier is very sensitive to outliers in the data and many real world problems present data that isn't strictly linear separable. Thus a hard margin classifier cannot solve such a problem sufficiently. Those situations require a variation of the constraints formulated above: Some misclassifications should be allowed as long as the remaining data points can be separated in an optimal fashion. This intuition is met by the Soft Margin Classification: Some data points are allowed to be on the "wrong side" of the decision boundary but are increasingly penalized with growing distance to that boundary. Formally: To relax the constraints imposed by the Hard Margin Classifier a set of slack variables  $\xi_n$  is introduced with  $\xi_n \geq 0$ . Every data point is mapped to one slack variable, serving as a penalty for margin violations and scaling this effect proportionally with the distance from the margin (Bishop, 2006, p. 332):

- An instance  $x_n$  on the correct side of the margin is mapped to a slack variable  $\xi_n$  with zero value.
- An instance on the wrong side of the margin gets assigned a slack value  $0 \leq \xi_n \leq 1$  if it still respects the decision boundary. This instance still is correctly classified but violates the maximum margin set by the classifier.
- For a misclassified instance  $x_n$  the slack value takes on a value of  $\xi_n > 1$ .

With the introduction of the slack variables the optimization problem now contains two conflicting forces: On the one hand the maximum margin still needs to be found, whereas the margin violations expressed by the sum over the slack variables should be minimized. To manage this trade off a new parameter  $\mathcal{C}$  is introduced into the equation controlling the regularization by scaling the penalizing effect of the slack variables.

Including these conceptual additions the optimization problem for the Soft Margin Classifier is expressed as:

$$\begin{aligned} & \underset{w,b}{\text{minimize}} \frac{1}{2} w^T w + \mathcal{C} \sum_{n=1}^N \xi_n \\ & \text{subject to:} \\ & t_n(w^T x_n + b) \geq 1 - \xi_n \quad \text{and } \xi_n \geq 0 \quad \text{for } n = 1, \dots, N. \end{aligned}$$

Equation 2-7: Soft Margin Linear SVM Classifier  
Objective Function (Géron, 2019, p. 167)

Another powerful feature of SVMs to solve non-linear data situations efficiently is their ability to apply the “Kernel Trick”. It describes the process of applying kernel functions that map non-linear data input to higher dimensions in order to find better conditions for a linear separation there. Mathematically this implies exchanging every dot product in the optimization problem with a non-linear kernel function.

*“SVMs are very universal learners. In their basic form, SVMs learn [a] linear threshold function. Nevertheless, by a simple “plug-in” of an appropriate kernel function, they can be used to learn polynomial classifiers, radial basis function (RBF) networks, and three-layer Sigmoid neural nets”* (Joachims, 1998, p. 138).

With all this SVMs are well equipped for document classification tasks. They deal well with the high dimensional input space resulting from the usage of weighted word schemes like TF-IDF: *“One remarkable property of SVMs is that their ability to learn can be independent of the dimensionality of the feature space. SVMs measure the complexity of hypotheses based on the margin with which they separate the data, not the number of features. This means that we can generalize even in the presence of very many features ...”* (Joachims, 1998, p. 139).

## 2.3 Selected Deep Learning Models for Text Classification

The classifiers discussed so far have demonstrated their strengths in many practical applications of NLP over a long time. But the advent of Deep Learning has drawn a lot of attention: *“Deep Learning has emerged in the last decade as the vehicle of the latest wave in AI. Results have consistently redefined the state-of-the-art for a plethora of*

*data analysis tasks in a variety of domains. For an increasing amount of Deep Learning algorithms, better-than-human (human-parity or superhuman) performance has been reported ... ”* (Raaijmakers, 2021est., p. 12).

This rise of Deep Learning spawned various model architectures particularly suited for tasks of the NLP domain. Amongst other properties these models convince with their capability to process sequential data like texts in documents. “*Theoretical results [...] suggest that in order to learn the kind of complicated functions that can represent high-level abstractions (e.g. in vision, language, and other AI-level tasks), one may need deep architectures*” (Glorot & Bengio, 2010, p. 249).

This chapter will introduce some of those ideas and provide a brief description of their theoretical foundation. The focus is on model architectures that are selected and applied to the document classification task during the experiments described later (chapter 4).

### **2.3.1 Key Concepts applied in Deep Learning Models**

Some principal concepts are applied to the design and training of every Deep Neural Net (DNN): An adequate loss function and optimizer needs to be chosen, and different strategies can ensure a more robust training, convergence and good generalization.

#### **2.3.1.1 Loss Function for Multi-Class Classification**

In a supervised learning context an Artificial Neural Net (ANN) produces estimations that are compared against the target values of the training instances. A loss function computes the deviance between the predictions and the target values. Training a neural network is solving an optimization problem with regards to this function: “*Given neural network parameters  $\theta$ , find the value of  $\theta$  that minimizes the cost function  $J(\theta)$* ” (Goodfellow, 2015). Various loss functions cater for different needs. Choosing an adequate loss function for a given learning task is a key decision for the project.

In multi-class classification records are assigned mutually exclusive to classes. Hence the target label distribution of one instance can be expressed as a sparse vector with targets  $t_k \in \{0,1\}$ , in a *1-of-K* coding scheme.

The cross entropy function (see Equation 2-5) introduced earlier for the binary case ( $K=2$ ) of Logistic Regression can be extended to a multi-class task with  $K>2$ . In a neural network this implies that for every document the output layer needs to generate a result vector of  $K$  probabilities  $y_k$  that can be compared against the target vector: “*If we have  $K$  separate binary classifications to perform, then we can use a network having  $K$  outputs each of which has a logistic Sigmoid activation function*” (Bishop, 2006, p. 235).

A Softmax function  $y_k(x, w)$  normalizes the result vector of single probabilities (the outputs of the Sigmoid functions) into a probability distribution to ensure that the networks' predicted single probabilities add up to 1 for every document.

$$y_k(x, w) = p(t_k = 1|x) \text{ with } \sum_k y_k = 1$$

$$y_k(x, w) = \frac{\exp(a_k(x, w))}{\sum_j \exp(a_j(x, w))} \text{ with } a_k = w_k^T x \text{ and } 0 \leq y_k \leq 1$$

$$E(w) = - \sum_{n=1}^N \sum_{k=1}^K t_{nk} \ln y_k(x_n, w)$$

Equation 2-8: Softmax Activation Function & Categorical Cross Entropy (Bishop, 2006, pp. 235-236)

The loss  $E(w)$  for the multi-class case is expressed by the Categorical Cross Entropy computed between the distribution of actual targets and the predicted probability distribution of classes summed over  $N$  training instances. The optimization problem, the objective of the training process, is to find the weight vector  $w$  that minimizes the Categorical Cross Entropy.

### 2.3.1.2 Optimizers

With a loss function  $E(w)$  established backpropagation computes the partial derivative for every weight in  $w$  with regard to the loss function, determines the gradient  $\nabla E(w)$  at learning step  $\tau$  and incrementally updates  $w^\tau$  in the opposite direction of the gradient (going downhill), resulting in the new weights vector  $w^{\tau+1}$ . The learning rate  $\eta$  controls the impact of the update by scaling the gradient step. The process is formally expressed (Bishop, 2006, p. 240):

$$w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E(w^{(\tau)})$$

Equation 2-9: Gradient Descent

The learning rate  $\eta$  is an important hyper parameter. It's driving the speed of the training process and plays a major role in supporting the algorithm to find the global optimum and not to get stuck in local optima: While it will generally accelerate training a too high  $\eta$  might lead to overshooting, in that the global minimum of the cost function is not found because the algorithm overshoots when in the local area of the best solution and perhaps even deteriorates away. A too small  $\eta$  on the other hand might work better for that matter but could conversely prolong the network training substantially and

potentially run the risk of not converging because of it being trapped in a local minimum, unable to get out and continue its search for the global optimum.

Batch Gradient Descent makes use of the entire training set. Variations thereof like Stochastic Gradient Descent (randomly picking just one instance) and Mini-Batch Gradient Descent (using smaller mini batches for every update step) add improvements in terms of training speed and memory management (Hinton, et al., 2012, pp. 4-6).

Numerous optimization methods have been developed. For this project the RMSProp algorithm developed by Geoffrey Hinton and Tijmen Tieleman (Hinton, et al., 2012, p. 29) is chosen. RMSProp computes the gradient update by considering the velocity of recent gradient upgrades. This optimizer is able to modify the learning rate and adapt it to the current local environment of the optimization problem.

RMSProp applies two computational steps: First it computes a moving average  $s$  using the squared gradients  $\nabla_{\theta}J(\theta) \otimes \nabla_{\theta}J(\theta)$  multiplied with a decaying factor  $\beta$ . Then the gradient of the current learning step  $\nabla_{\theta}J(\theta)$  is normalized by the root mean squared average  $\sqrt{s + \epsilon}$ :<sup>5</sup>

$$s \leftarrow \beta s + (1 - \beta)\nabla_{\theta}J(\theta) \otimes \nabla_{\theta}J(\theta)$$

$$\theta \leftarrow \theta - \eta \frac{\nabla_{\theta}J(\theta)}{\sqrt{s + \epsilon}}$$

Equation 2-10: RMSProp algorithm (Géron, 2019, p. 356)

### **2.3.1.3 Strategies to prevent Vanishing Gradients**

The training of a neural network generally is prone to an unfavorable misbehavior: The “Vanishing Gradients” phenomena.

For a deep network with many layers, neurons and weights in between them backpropagation translates into calculating a large number of partial derivatives. Computing derivatives implies using the chain rule of calculus<sup>6</sup>. Most of the activation functions used back in the day yield small output values near zero. Computing the gradients of a deep neural net means multiplying lots of these small numbers. Because

<sup>5</sup> A small smoothing term  $\epsilon$  is added to prevent zero division

<sup>6</sup> This chaining of operations can also produce the unfavorable exploding of the gradients problem. That is when values larger than 1 are pushed through thousands of multiplications scaling the upgrades of the ANN to exponentially large portions.

of this the gradients in a deep net with many layers can grow smaller and smaller. The error signal traveling backwards through the network reduces exponentially until it finally vanishes. Thus preventing an effective upgrade of  $\theta$  for the lower layers of the network. With no further change induced at the lower layers the network cannot learn anymore. The training stops long before a satisfying result is reached.

Effective strategies to counter this behavior were developed in 2010 (Glorot & Bengio, 2010). It was found that a poor choice of the activation functions strongly contributed to the problem. Inspired by the natural design of biological neurons, it was very common to use the Logistic Function at that time. Figure 2-2 illustrates the unfavorable behavior of the Sigmoid in this context: Its output saturates with large valued inputs (negative or positive) to either 0 or 1. Computing the derivate in these areas results in values of or near zero.

*“Thus when backpropagation kicks in it has virtually no gradient to propagate back thought the network; and what little gradient exists keeps getting diluted as backpropagation progresses down through the top layers, so there is really nothing left for the lower layers”* (Géron, 2019, p. 332).

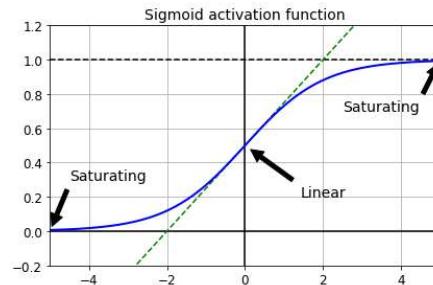


Figure 2-2: Logistic Activation Function Saturation (Géron, 2019, p. 333)

Based on this insight better suited activation functions for this purpose are required. The Rectified Linear Unit (ReLU) is commonly chosen because it's not saturating for positive input values  $z$  and with its mathematical simplicity (see below) it's also fast to compute. *“This activation function is the default activation function recommended for use with most feedforward neural networks”* (Goodfellow, et al., 2016).

$$\text{ReLU}(z) = \max(0, z)$$

$$\text{LeakyReLU}_\alpha(z) = \max(\alpha z, z)$$

Equation 2-11: ReLU and Leaky ReLU (Géron, 2019, p. 335)

But ReLU still has a limitation in being not differentiable at zero and in that it produces strictly a zero valued derivative for any negative input. If a majority of the neurons in

the network produces negative valued outputs it would result in gradients filled with zeros. A behavior described as “Dying ReLUs”.

This is countered with a variation of ReLU the Leaky ReLU: Given an input value  $z < 0$  it multiplies  $z$  with a small scaling factor  $0 < \alpha < 1$  and typically sized  $\alpha = 0.01$  (Géron, 2019, p. 335).

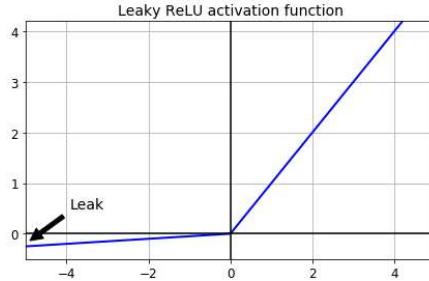


Figure 2-3: Leaky Rectified Linear Unit Function (Géron, 2019, p. 336)

In their 2010 paper Glorot and Bengio also recognized that the initialization scheme of an ANN was an important contributor to the saturating gradients phenomena (Glorot & Bengio, 2010). A normal distribution  $\mathcal{N}(\mu = 0, \sigma = 1)$  was commonly used to initialize the network. It was noted that the variance of the outputs of every layer should be equal to the inputs’ variance of that layer and during backpropagation the same should hold for the reverse direction. Input and output variance of the gradients should be kept equal for every layer. To approximate a solution they developed the “Glorot initialization” scheme factoring in the average number of inputs (fan-in) and outputs (fan-out) used:

$$\mathcal{N}(\mu, \sigma) \text{ with } \mu = 0 \text{ and } \sigma = \frac{1}{\text{fan}_{\text{avg}}}$$

$$\text{and } \text{fan}_{\text{avg}} = \frac{(\text{fan}_{\text{in}} + \text{fan}_{\text{out}})}{2}$$

Equation 2-12: Glorot Initialization (Géron, 2019, p. 334)

Other initialization schemes were subsequently developed like an initialization scheme optimized for the usage with ReLUs called “He-Initialization” (He, et al., 2015). But the search for methods to improve the training process of DNNs also revealed other techniques beyond the choice of the initialization scheme.

#### 2.3.1.4 Batch Normalization

In a 2015 paper Ioffe and Szegedy proposed a technique called Batch Normalization. “*Training Deep Neural Networks is complicated by the fact that the distribution of*

*each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as internal covariate shift, and address the problem by normalizing layer inputs*" (Ioffe & Szegedy, 2015, p. 1). In a DNN where every layer takes its input from lower layers and channels its output to higher layers applying Batch Normalization smoothens the various input distributions and prevents an unfavorable high variance between inputs and outputs of a layer.

The key idea of Batch Normalization is a normalization step built into the network architecture, ideally adding one Batch Normalization layer before every activation of the network. This way every mini-batch<sup>7</sup> of inputs gets zero centered with a unit variance and then linearly transformed by a scale and a shift parameter vector  $\gamma$  and  $\beta$ . The model learns these two vectors for every layer<sup>8</sup>.

For a mini-batch of inputs  $\mathcal{B} = \{x_1, \dots, x_m\}$  the algorithm computes the mini-batch mean  $\mu_{\mathcal{B}}$  and variance  $\sigma_{\mathcal{B}}^2$  and uses them to normalize<sup>9</sup> every  $x_i$  into  $\hat{x}_i$ :

$$\begin{aligned}\mu_{\mathcal{B}} &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad \text{and} \quad \sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \\ \hat{x}_i &\leftarrow \frac{(x_i - \mu_{\mathcal{B}})}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}\end{aligned}$$

$\hat{x}_i$  is linearly transformed using the scaling vector  $\gamma$  and the shifting vector  $\beta$ , resulting in the rescaled and shifted output vector  $y_i$

$$y_i \leftarrow \gamma \hat{x}_i + \beta$$

Equation 2-13: Batch Normalization Algorithm (Ioffe & Szegedy, 2015, p. 3)

The authors demonstrate that Batch Normalization can significantly improve DNN training: The vanishing gradients phenomena is kept at bay, the network is less sensitive to the choice of activation function. With more stability and robustness in training the applied learning rates can be increased, thus accelerating the training process considerably.

<sup>7</sup> During training the entire input of a training set is portioned into equally sized batches. Within one training step an entire batch is fed to the network. When all batches have been processed, the network has completed one epoch. ANNs get trained over many epochs.

<sup>8</sup> For every layer specific input  $\gamma$  contains one scaling parameter and  $\beta$  one shifting parameter. Every input into the layer is scaled and shifted by its respective parameter set.

<sup>9</sup> Note that in the denominator a smoothing term  $\epsilon$  is added to avoid zero-division.

### 2.3.1.5 Regularization strategies

A DNN with tens of thousands or even millions of parameters is prone for overfitting the training data. “*If the relationship between the input and the correct output is complicated and the network has enough hidden units to model it accurately, there will typically be many different settings of the weights that can model the training set almost perfectly, especially if there is only a limited amount of labeled training data*” (Hinton, et al., 2012). Given this freedom of enough adjustable parameters the model learns residual noise as if it was a structural property of the data. For that matter it fails to predict with sufficient accuracy when it is exposed to new and unseen data at inference time. The model generalizes poorly. Different regularization strategies can prevent overfitting and reduce the test error:

Well established techniques in traditional machine learning theory like  **$L_1$**  and  **$L_2$  regularization** (a penalty term depending on the amount of parameters used is added to the loss function) are likewise applicable to DNNs (Goodfellow, et al., 2016, pp. 227-231).

**Batch Normalization** in addition to its described positive contributions (see above) also imposes regularization to a DNN, thus reducing the need for other regularization techniques (Ioffe & Szegedy, 2015, p. 5).

Another common strategy is the application of **Dropouts**. Implemented into the architecture of the network Dropouts randomly switch off neurons of the connected layer during model training. Every neuron (excluding the final output layer) in the respective layer will be temporarily dropped with a probability  $p$  in one training step. With a sufficient dropout rate  $p$  assigned to every layer the network learns to generalize because it cannot rely any longer on the predictive power of all the neurons. Given the large number of permutations with neurons being switched on or off, adding Dropouts can be seen as sampling one version from an exponentially large set of different “thinner” neural networks at every training step. “*Random dropout makes it possible to train a huge number of different networks in a reasonable time. There is almost certainly a different network for each presentation of each training case but all of these networks share the same weights for the hidden units that are present*” (Hinton, et al., 2012, p. 2).

Of course the neurons are only dropped during training of the net. “*At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods. We show that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and*

*computational biology, obtaining state-of-the-art results on many benchmark data sets.”* (Srivastava, et al., 2014, p. 1929)

Another strategy to prevent overfitting is **Early Stopping**. During training the model is periodically applied to a held-out validation set of unseen data. The key idea is that a low error measured on the validation set translates into a low error on the test set. If a chosen metric (i.e. loss or accuracy) on the validation set improves the current weights of this training step are stored. The training process terminates when the validation metric doesn’t improve beyond a defined tolerance within a set patience period (i.e. a number of batches, or epochs). Upon termination the model will roll-back to the last stored weight parametrization, the best version so far (Goodfellow, et al., 2016, p. 243).

### 2.3.2 Recurrent Neural Nets

Human comprehension builds on connecting current input with context. People find it typically easy to complement missing terms within a known sequence (i.e. completing an alphabetical sequence). For good language comprehension processing the sequential order of words is imperative. Regular feed-forward neural networks can’t do that.

Recurrent Neural Nets (RNNs) can share their parameters across different parts of a model. This is a powerful property allowing for processing texts of arbitrary length and generalizing across them. *“A traditional fully connected feedforward network would have separate parameters for each input feature, so it would need to learn all the rules of the language separately at each position in the sentence. By comparison, a recurrent neural network shares the same weights across several time steps”* (Goodfellow, et al., 2016, p. 368).

RNNs have recurrent loops that channel information from earlier time steps back into the network. They process a sequence of vectors  $x^{(t)}$ , with  $t$  indexing the sequential position of a time step.<sup>10</sup> The RNN processes the data  $x$  with a function  $f$  and parameters contained in  $\theta$  transforming it into a hidden state  $h$  and propagates  $h$  forward through time, where it’s combined with further sequential information of the next time step. Figure 2-4 illustrates this process with a circuit diagram (on the left side) and a black rectangle indicating the passage of multiple time steps. Unfolded (right side) the graph shows each time step explicitly connected to its previous and following time step.

---

<sup>10</sup> A time step in this context expresses foremostly the sequential position of an information, not necessarily the passage of a time period.

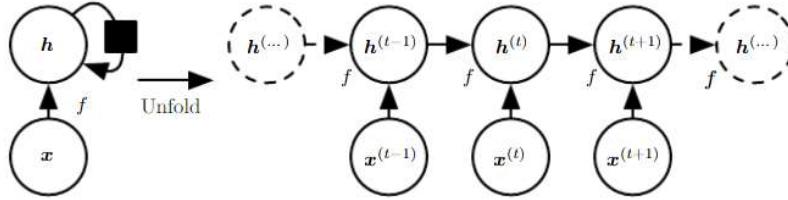


Figure 2-4: Unfolding of an RNN with no Outputs  
(Goodfellow, et al., 2016, p. 370)

It's easy to see the processing power of RNNs for text by mapping each time step  $x^{(t)}$  to a word of a sequential input like a sentence, an abstract or a document.

The RNN illustrated above is missing an important feature as it shows no outputs other than the hidden states. This can be changed: RNNs can be designed very flexible according to the assignment they are purposed to. The architectural core elements of inputs, hidden states and outputs can be combined in many ways to cater for different learning tasks:

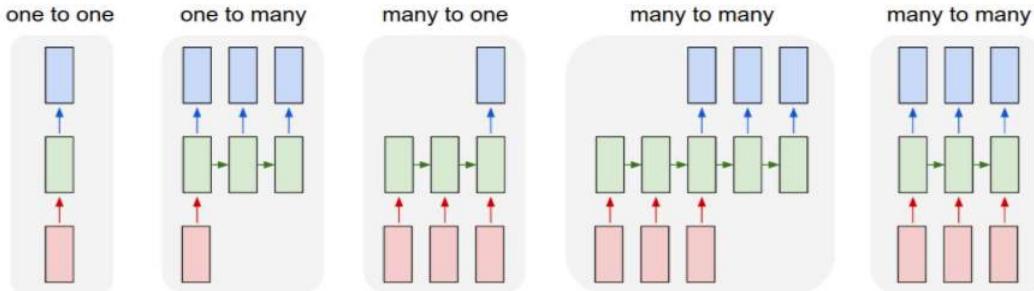


Figure 2-5: Archetypes of different RNN Architectures (Karpathy, 2015, p. 2)

For the given document classification task an RNN is supposed to have multiple inputs (the text), share connections between its hidden states and produce an output for every time step. Hence the “Many to Many” architecture shown in Figure 2-5 on the right hand side abstracts the architecture best suited for this specific job. This RNN produces an output vector  $o$  (the blue boxes) that is fed to a multi-layered dense network with a final layer applying a Softmax activation. This Softmax Layer finally generates the prediction format: A distribution of probabilities over all classes for every record.

Three weight matrices keep track of all parameters used in this RNN architecture:

- Matrix  $U$  for connections between inputs  $x$  and hidden states  $h$
- Matrix  $W$  for connections between hidden states and neighboring hidden states
- Matrix  $V$  for connections between hidden states and outputs  $o$

With two additional bias vectors  $b$  and  $c$  introduced the propagation of input and hidden states through the network can formally be expressed by three equations:

$$\begin{aligned} a^{(t)} &= b + Wh^{(t-1)} + Ux^{(t)} \\ h^{(t)} &= \tanh(a^{(t)}) \\ o^{(t)} &= c + Vh^{(t)} \end{aligned}$$

Equation 2-14: Forward Propagation in a multi-output RNN  
with a tanh-Activation (Goodfellow, et al., 2016, p. 374)

The strength of RNNs stems from two properties: A collection of hidden states, capable of preserving information through time and non-linear functions (like the hyperbolic tangent function ( $\tanh$ ) in Equation 2-14) that allow to model highly complex patterns of the input data (Hinton, et al., 2012).

Training an RNN means learning the best parametrization of the weight matrices  $U$ ,  $W$  and  $V$  to minimize the loss function. Like with any other neural net this is done by backpropagation. “*Computing the gradient through a recurrent neural network is straightforward. [...] No specialized algorithms are necessary*” (Goodfellow, et al., 2016, p. 379). One important difference in an RNN however is that the flow of computed derivatives needs to mirror the sequential chaining of the hidden states and outputs. For this reason that process is called “Backpropagation through Time”.

But despite their strengths described RNNs suffer of two major limitations that ask for additional mitigating strategies, particularly when trying to process long sequences of text (such as entire documents):

- The complex chaining of hidden states over many time steps makes RNNs especially deep. Very deep networks are even more susceptive for the vanishing or exploding gradients problem, resulting in an unfavorably unstable training process.
- Another limitation is the short-term memory problem. With every hidden state  $h^{(t)}$  constantly being rewritten while the information is propagated through the network, it’s difficult to keep information present over many time steps. “*After a while, the RNNs state contains virtually no trace of its first inputs*” (Géron, 2019, p. 514). RNNs suffer from a short term memory limitation.

### 2.3.3 Long Short-Term Memory

A 1997 paper introduced the idea of a Long Short-Term Memory (LSTM) architecture (Hochreiter & Schmidhuber, 1997). The authors show that LSTMs behave much more robust and can resolve the limitations of RNNs on long sequences.

The key is to incorporate memory cells that can learn what part of a previously stored context is not needed any longer and can be removed from memory. Likewise they determine what part of incoming information is added to this long short-term memory, preserving this context for later time steps.

Figure 2-6 illustrates (left panel) a schema with a hidden state in a RNN with a hyperbolic tangent ( $\tanh$ ) activation function. The right panel visualizes the concept of a memory cell in a LSTM network augmenting what is a hidden state in a regular RNN.

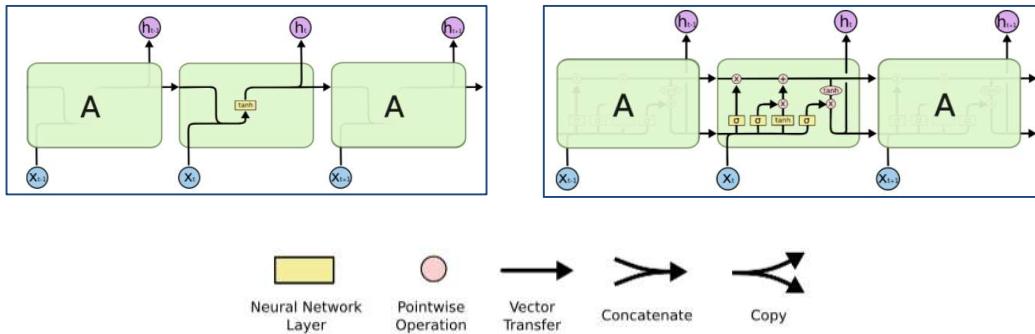


Figure 2-6: Schema of a hidden State in a regular RNN (left panel) and (in the right panel) a Memory Cell within an LSTM. (Olah, 2015)

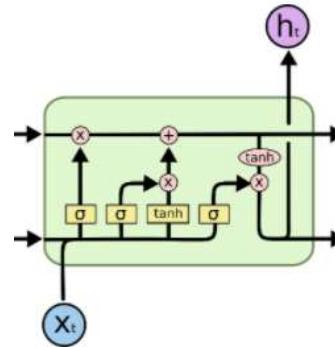
Memory cells modify the long-term information based on the input. Thus the newly arriving input is not only introducing new data but also setting three corresponding gating units (Sigmoid activation functions). These specific gates are called the forget gate  $f_t$ , the input gate  $i_t$  and the output gate  $o_t$ . Based on the combined input of  $x_t$  and the previously computed hidden state  $h_{t-1}$  (see below) these gates are set to be ...

- *open*: The Sigmoid outputs 1, indicating to keep everything (no filtering)
- *closed*: The Sigmoid outputs 0, signaling to remove everything (full filtering)
- *in-between*: If the Sigmoid outputs a value between 0 and 1 it will take over some parts of the vector it is applied to (partial filtering).

With that the inner workings in the memory cell can be formally expressed (Jurafsky & Martin, 2019, pp. 184-185) and illustrated (Olah, 2015) stepwise as follows:

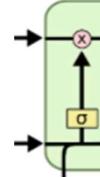
- (1) **Setting the gates**<sup>11</sup>: At time step  $t$  each gate  $f_t, i_t, o_t$  is computed by a respective Sigmoid  $\sigma$  that is applied to the sum of two matrix multiplications: The weight matrices  $U_f, U_i$  and  $U_o$  are multiplied with  $h_{t-1}$  (the short term state from the previous time step) and the respective weight matrices  $W_f, W_i$  or  $W_o$  are multiplied with the current input  $x_t$ . In the following steps the respective output vectors of the gates (all valued between 0 and 1) will be pointwise multiplied with the vectors they need to take control of. Thus masking the information of these vectors.

$$\begin{aligned} f_t &= \sigma(U_f h_{t-1} + W_f x_t) \\ i_t &= \sigma(U_i h_{t-1} + W_i x_t) \\ o_t &= \sigma(U_o h_{t-1} + W_o x_t) \end{aligned}$$



- (2) **Removing context from the long term context vector**: The forget gate  $f_t$  is elementwise multiplied with the context vector  $c_{t-1}$  to remove context that is not needed any longer, resulting in the modified context vector  $k_t$ :

$$k_t = c_{t-1} \odot f_t$$



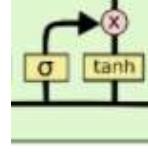
- (3) **Computing the new current cell context**: the context to be added currently to the memory cell  $g_t$  is the output of a hyperbolic tangent activation function applied to the sum of the multiplication of the input  $x_t$  and the hidden state  $h_{t-1}$  with their respective weight matrices  $U_g$  and  $W_g$ . By elementwise multiplication of  $g_t$  and the input gate  $i_t$  the filter is applied selecting what parts of the new context are taken over into the final current context  $j_t$

---

<sup>11</sup> The three gates are indicated with  $\sigma$  in the detailed illustrations below.

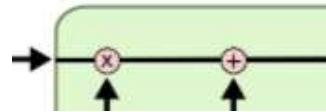
$$g_t = \tanh(U_g h_{t-1} + W_g x_t)$$

$$j_t = g_t \odot i_t$$



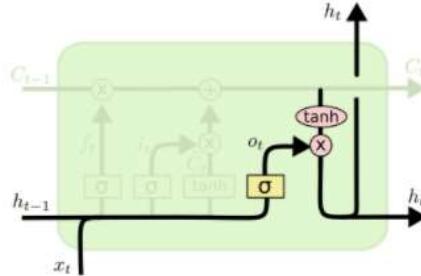
- (4) **Updating the long term context vector  $c_t$ :** The current cell context is added to the context vector  $k_t$ :

$$c_t = j_t + k_t$$



- (5) **Writing the hidden state  $h_t$ :** The updated context vector  $c_t$  is transformed by another hyperbolic tangent activation function and elementwise multiplied with the output gate  $o_t$  to determine the current hidden state  $h_t$  that will be passed on to the next cells.

$$h_t = o_t \odot \tanh(c_t)$$



With all appropriate weights set, an LSTM takes the hidden state and the long term context vector from the last time step and combines it with the current input vector. The previous hidden state and the current input determine the state of the gates in conjunction with their respective weights. Then based on these gates the long term context is modified and used to compute the hidden state of this memory cell. The (modified) long term context and the hidden state are passed on to the next layer.

*“An LSTM cell can learn to recognize an important input [...], store it in the long term state, preserve it for as long as it is needed [...], and extract it whenever it is needed. This explains why these cells have been amazingly successful at capturing long-term patterns in time series, long texts, audio recordings, and more”* (Géron, 2019, p. 517).

### 2.3.4 Bidirectional Recurrent Neural Networks

In a conventional RNN the hidden state  $h_t^f$  incorporates only the information that the RNN has seen and processed from the front (denoted with index  $f$ ) of the sequence up to the current time step  $t$ .

$$h_t^f = \text{RNN}_{\text{forward}}(x_1^t)$$

Equation 2-15: Hidden state computed from front context  
in a conventional RNN (Jurafsky & Martin, 2019, p. 182)

But good comprehension of a sequence of words can be difficult if only context from previous time steps and no context from later time steps is available. Very often the meaning of a text is better comprehended with all context considered. In a document classification task the full context typically is available to support at training and also at inference time. Using the full available context and not only the front context might yield better results.

This can be implemented into the RNN architecture if another layer  $h_t^b$  of the RNN gets to learn from the inverse sequence (backwards). With that the two vectors  $h_t^f$  and  $h_t^b$  can be concatenated to  $h_t$ , representing the full context at time step  $t$ :

$$\begin{aligned} h_t^b &= \text{RNN}_{\text{backward}}(x_1^n) \\ h_t &= h_t^f \oplus h_t^b \end{aligned}$$

Equation 2-16: Hidden state computed from front and back context  
in a conventional RNN (Jurafsky & Martin, 2019, p. 182)

### 2.3.5 Convolutional Neural Nets

Convolutional Neural Nets (CNNs) contributed significantly to the domain of image classification. The key principle of CNNs is the training of numerous filters that can identify immanent structures in the data and extract them to features for subsequent layers of a network. *“A convolutional neural network is designed to identify indicative local predictors in a large structure, and combine them to produce a fixed size vector representation of the structure, capturing these local aspects that are most informative for the prediction task at hand”* (Goldberg, 2015, p. 42).

In an NLP task it's unlikely for all words and sentences to carry the same descriptive power. Some word co-occurrences, some collocations or some sentences might be very distinguishing for a specific category. There should be emphasis to find exactly those patterns. This is the driving insight for applying CNNs to a text classification task.

In the NLP domain convolutions are computed over a 1-dimensional sequential input, typically a sentence, an abstract or an entire document. The basic convolution can be formally described as follows (Goldberg, 2015, p. 43):

In a sequence of  $n$  words  $x = x_1, \dots, x_n$ , every  $x_i$  is mapped to a corresponding  $d_{emb}$ -dimensional word embedding  $v(x_i)$ . A  $k$ -sized window is moving  $m = n-k+1$  times<sup>12</sup> across the sequence of words and concatenates the mapped embeddings. The series  $(v(x_i) \oplus v(x_{i+1}) \oplus \dots \oplus v(x_{i+k-1}))$  represents one window  $w_i$  with  $w_i \in \mathbb{R}^{kd_{emb}}$ . This window vector  $w_i$  is fed to a linear transformation and an element-wise applied non-linear activation function  $g()$ . The convolution layer produces  $m$  times a resulting vector  $p_i$  for every window  $w_i$  with a dimension  $d_{con}$ .

$$p_i = g(w_i W + b)$$

Equation 2-17: Basic Convolution (Goldberg, 2015, p. 43)

A CNN layer is typically fed to a subsequent network layer, expecting every input to be of same dimension. Of course text can vary in length of input, thus violating this requirement. As a remedy a Max Pooling<sup>13</sup> layer is applied producing a single vector  $c_j$ , representing every dimension  $j$  of the  $m$  vectors with its respective max value:

$$c_j = \max_{1 \leq i \leq m} p_i[j]$$

Equation 2-18: Max Pooling (Goldberg, 2015, p. 43)

A convolutional layer isn't limited to applying just one filter. It can apply many of them. Thus the resulting vector  $c$ , representing the input sequence is featuring as many dimensions  $j$ , as number of filters are applied. With every filter operating upon an independent weight matrix, bias and activation function the idea is to train particular filters for different immanent features within an input sequence. *"Ideally, each dimension will 'specialize' in a particular sort of predictors, and max operation will pick on the most important predictor of each type"* (Goldberg, 2015, p. 43).

<sup>12</sup> This formulation represents a “narrow convolution”, which applies a total number of  $m = n-k+1$  window movements over the input, potentially skipping a remainder of input words that don't fit into a full k-sized window. Should it be favorable to include those remaining words, a padding of an adequate number of zero-valued vectors can be applied to the beginning and the end of the sequence. This formulation would result in  $m = n-k+1$  window movements and represent a “wide convolution” (Kalchbrenner, 2014).

<sup>13</sup> The intuition behind selecting the max value is to choose the most salient information from the convolutional representation. While max pooling is commonly chosen, other mathematical operations like using the average or the min value can be employed instead.

Figure 2-7 illustrates a 1d convolution and max pooling performed on an example sentence, using a window size of 3, an assumed 2-dimensional embedding vector for every word and the application of 3 filters, resulting in a final 3 dimensional pooling vector.

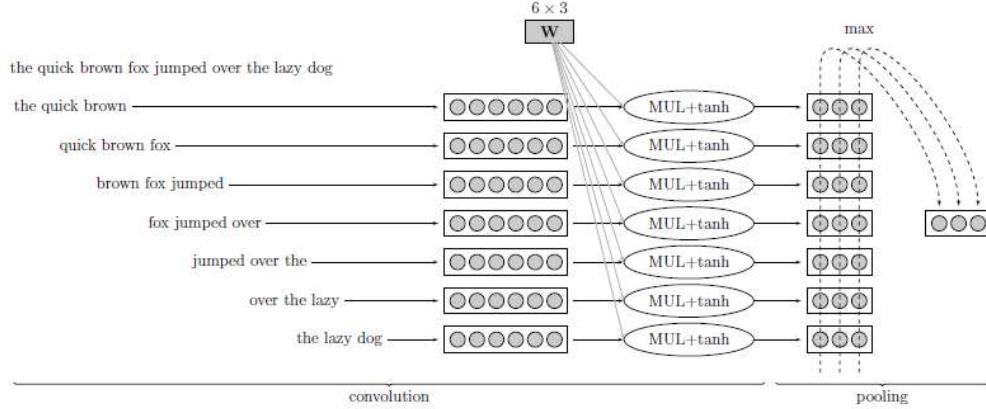


Figure 2-7: Example of 1d-Convolution and Max-Pooling with 3 Filters (Goldberg, 2015, p. 44).

During training backpropagation will continuously upgrade the gradients through the network and tune the corresponding parameters to minimize the loss function. With sufficient training the final CNN layer holds a number of expert filters for different feature patterns trained on the respective classification task.

Compared to conventional RNNs or LSTMs, CNNs can take much more advantage of parallel computing enabled by modern Graphical Processing Units (GPUs) because the learning of multiple filters doesn't depend on a sequential order of the processing steps. This allows for longer and more efficient training.

Furthermore “... results indicate that a simple convolutional architecture outperforms canonical recurrent networks such as LSTMs across a diverse range of tasks and datasets, while demonstrating longer effective memory. We conclude that the common association between sequence modeling and recurrent networks should be reconsidered, and convolutional networks should be regarded as a natural starting point for sequence modeling tasks” (Bai, et al., 2018).

## 2.4 Self-Attention & Transformer Networks

For comprehension of text it's beneficial to account for the order of the sequential input and the context of words when processing the text. While proximity of words is often helpful to capture a semantical concept, it doesn't work always. In cases where context is spread across an entire document with multiple sentences co-referring to each other

this is still a challenging task for a sequential network architecture: Training LSTMs on long sequences (like for document classification) is cumbersome and error-prone, because the gradients have to travel long distances and context can be distributed over different parts of a sequence.

A widely recognized paper on the task of neural machine translation (Bahdanau & Yoshua, 2016) introduced the concept of Attention: A global rather than a local or sequential approach to capture semantically relevant structures in a given text.

The key idea of Attention is to encode textual input with contextual focus information. Each token carries additional information to determine what other tokens it might depend upon or is referring to. This gave way for a team of Google© researchers to implement a variation of Attention into a network architecture called the Transformer network. Because their design of an encoder-decoder model broke away from the typical usage of RNNs, CNNs and LSTMs they announced: “Attention Is All You Need” (Vaswani, et al., 2017).

The Transformer model is designed for a sequence-to-sequence task (language translation) with an encoder and a decoder unit. As this project focuses on document classification the decoder (the right part of the model architecture shown in Figure 2-8) is omitted for the further discussion and the focus is put on the encoder (the left part).

For a document classification task the output of the encoder is fed to a network with further (dense) layers to classify the documents (not shown here).

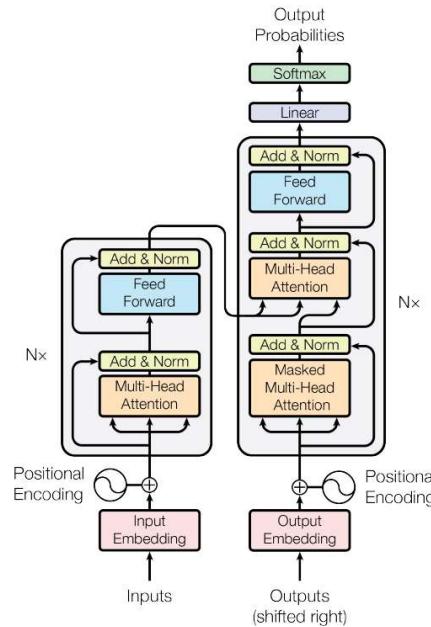


Figure 2-8: The Transformer Model Architecture (Vaswani, et al., 2017, p. 3)

The encoder is made up of  $N$  stacked identical layers<sup>14</sup>. Within each layer the data traverses two consecutive sub-layers: A Multi-Head Attention block and a Feed Forward block with residual connections<sup>15</sup> skipping over these two blocks and a Normalization layer after each of the two blocks.

#### 2.4.1 Positional Encodings

Because RNNs and LSTMs process input sequentially one embedding at a time step, the order of the tokens is maintained and there is no need for a dedicated positional encoding. The encoder module of a Transformer model expects the inputs preprocessed with a positional encoding:

Like in other Deep Learning models the input text is presented in the form of embedding vectors.<sup>16</sup> But unlike other sequentially working models the Transformer model is taking in the entire input sequence at once, entailing a big advantage: By usage of GPU technology and its ability to parallelize processing the training time is significantly reduced. But this advantage comes with a price: With no sequential information encoded any more, the order of the tokens is lost and cannot support the learning. Thus the authors introduced a concept to preserve the order of the inputs: Every embedding vector is combined with a positional embedding vector of same size encoding the unique position of the respective word into a unique numerical representation.

To compute the positional encoding wave frequencies are used: Embedding values with an even positional index are transformed using the sine function and values with an odd positional index are likewise transformed by the cosine function. Both functions take the positional index of the token ( $pos$ ), the embedding dimension  $i$  and the size of the embedding vector  $d_{model}$  for arguments to calculate a positional embedding vector  $PE$ :

$$PE_{pos,2i} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{pos,2i+1} = \cos(pos/10000^{2i/d_{model}})$$

Equation 2-19: Positional Encodings in the  
Transformers Model (Vaswani, et al., 2017, p. 6)

For every embedding vector there is a respective positional embedding vector  $PE$ . Being of same dimension the two vectors are added to form one combined vector. For

<sup>14</sup> The authors use  $N = 6$ .

<sup>15</sup> The residual connections are added to prevent the gradient signal from thinning out during backpropagation.

<sup>16</sup> The original Transformer model described by Vasvani et al uses a 512-dimensional embedding vector.

the next layers it suffices to use only this one vector. But this vector now is carrying the initial embeddings plus the positional information additionally encoded.

#### 2.4.2 Scaled Dot-Product Attention

The Positional Encodings are fed into a Multi-Head Attention layer (see Figure 2-8). This layer can be decomposed into single attention units called heads. Each head computes a scaled dot product attention: The incoming embedding vectors are parallelly channeled into three matrices: the query matrix  $Q$ , the key matrix  $K$  and the value matrix  $V$ . The matrices are of dimensionality  $d_k$  and  $d_v$ .

Self-Attention makes use of cosine similarity: The cosine of the angle between two vectors is approaching 1 the more similar these vectors are and -1 with decreasing similarity. Hence the dot product between the query matrix  $Q$  and the keys matrix  $K$  is computed to identify the keys most similar to the queries. The resulting matrix is scaled down<sup>17</sup> to keep numerical computation stable and then normalized with a softmax function to produce weights that add up to 1.

Scaled Dot-Product Attention

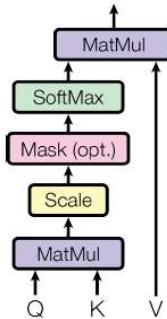


Figure 2-9: Scaled Dot-Product Attention  
in the Transformer Model Architecture (Vaswani, et al., 2017, p. 3)

For each token the resulting weight matrix represents the attention that should be given to every other token in the sequence. These weights are fed into another matrix multiplication together with the raw values matrix  $V$ . Thus transforming  $V$  into the Scaled Dot-Product Attention a new representation of the input with the attention scores encoded. Formally:

---

<sup>17</sup> The authors used  $\sqrt{d_k}$  to scale down the matrix multiplication  $QK^T$

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

Equation 2-20: Scaled Dot-Product Attention (Vaswani, et al., 2017, p. 4)

### 2.4.3 Multi-Head Attention

Traditional word vectors represent the meaning of words relatively stable across every usage. But the relationship expressed by an attention mechanism can be manifold. For a specific input sequence, attention can be given to different aspects in parallel, allowing for different perspectives and usages of the respective expression. “*Any given word can have multiple meanings and relate to other words in different ways, you can have more than one query-key-value complex attached to it. That’s “multi-headed attention.”*” (Nicholson, 2020).

The Transformers architecture implements this with the Multi-Head Attention Layer. The Scaled Dot-Product Attention described above using the matrices  $Q$ ,  $K$  and  $V$  is calculated  $h$  times in parallel. Every transformation  $i$  with  $i = 1, \dots, h$  makes use of a separate weight matrix  $W^Q$ ,  $W^K$  and  $W^V$ : “*On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding  $d_v$ -dimensional output values*“ (Vaswani, et al., 2017, p. 4).

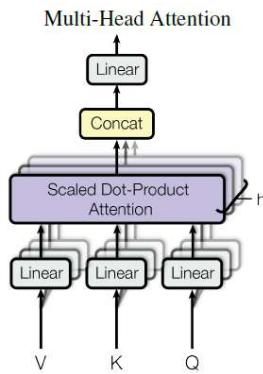


Figure 2-10: Multi-Head Attention  
in the Transformer Model Architecture (Vaswani, et al., 2017, p. 3)

By using  $h$  different starting points for  $Q$ ,  $K$  and  $V$  the algorithm can train jointly on different aspects of attention and isn’t limited to only one representation. The  $h$  output vectors of the Scaled Dot-Product Attention blocks are concatenated and one more time linearly transformed with a trainable weight matrix  $W^O$  to produce the Multi-Head Attention output:

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^o \\ \text{where } \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned}$$

Equation 2-21: Multi-Head Attention (Vaswani, et al., 2017, p. 5)

The concept of Attention and the Transformer model have fueled intensive research projects and further development since their first publication: “*Attention matters because it has been shown to produce state-of-the-art results in machine translation and other natural language processing tasks, when combined with neural word embeddings, and is one component of breakthrough algorithms such as BERT, GPT-2 and others, which are setting new records in accuracy in NLP. So attention is part of our best effort to date to create real natural-language understanding in machines. If that succeeds, it will have an enormous impact on society and almost every form of business.*” (Nicholson, 2020).

## 3 Data & Preprocessing

This chapter will detail the procurement of training and test data, explain the necessary preprocessing, especially the anonymization part and provide a brief exploration of the training data, its structure, properties and the challenges associated with it.

### 3.1 Procurement of Training Data

During 2020 a cross-functional team of BMW Bank experts implemented a new DMS. A complex rule-based logic was developed, configured in the software solution, tested and finally deployed to a final production environment.

The classifier to be developed in this project is to solve for all documents that this DMS cannot handle fully automatically. Documents falling short of automatic classification by the DMS in the first place are added to a backlog for manual intervention. Service agents are scheduled on a daily basis to clear the backlog manually, thus correctly labeling the document type for every document.

After Go-Live of the DMS a weekly set of training data has been exported from the production system. These exports cover documents manually classified by the service team and presented themselves as a hierarchical file structure in this form:

BatchClass => BatchID => DocumentID => Data Files

Each folder representing one DocumentID accommodates two text files (both in string format): A content file containing the text of the specific document and an index file with the meta-data for the content file. A typical example of the two files relating to one document is shown in the two panels below:

Content file <sup>18</sup> /Schaden/1182502/2949971/002D0353.txt:	Index file with meta data ../002D0353_index.txt:
'Page 1\nFrom:\t"Kanzlei Meyer" <info@muster.de>\nDate:\tThu, 17 Dec 2020 11:18:36+0100\nTo:\t"BMW Leasing GmbH"\t schaden@bmw.de>\nSubject:\tKunde: Muster Erik, Muster Weg 5, 12454 Musterstadt Verkehrsunfall vom:\n11.12.2020 Pkw, amtl.Kennz.: XX-JK1234 - Müller ./. Huber - Unser Zeichen: 1234/20-GG\nSehr geehrte Damen und Herren,\nIhr o.g. Kunde hat uns mit der zivilrechtlichen Schadensregulierung betreffend oben genannten Verkehrsunfall\nbeauftragt.\nDas Fahrzeug wurde bei dem o.g. Verkehrsunfall sehr stark beschädigt ...	"Schaden","Schaden PDF Image + Text","{Batch ID}", "1182502", "{DocumentID}", "2949971", "DocumentType", "SCHADENSCHREIBEN", "ClassificationResultWithConfidence", "SCHADENGUTACHTEN;60;P SCHADENANZEIGE;40;P SCHADENSCHREIBEN;20;P", "PageCount", "1", "{DocumentCount}", "1", "{\$InputChannel}", "EMAIL", "{\$sourceSystem}", "EMAIL", "AutoClassificationConfidence", "0", "\n

<sup>18</sup> All document content presented for illustrative purpose within this report has been cleared of any real personal data like names, addresses, registrations, account numbers etc..

A weekly batch of training data can contain up to 20.000 documents. The file structure is uploaded onto a High Security Data Analytics Platform (HSDAP) serving a Python 3.8 environment for data preprocessing and development. The file structure is parsed (Python) and the two files for each document are fed into a dataframe. The content and the index information are moved to separate columns of this dataframe. Then the index string is extracted into further columns capturing different descriptors of the data including the target label (“DOCTYPE”). Table 3-1 lists the different columns of the dataframe, their data type and a brief description of its meaning in the business context:

Data Structure of Export Data provided from Production DMS		
Column Name	Data type	Meaning
INDEXSTRING	object	Meta data regarding the document
RAWBODY	object	Content of the document (textual)
BATCHKLASSE	categorical	Source of origin (department)
BATCHCONTENT	categorical	Additional source information
BATCHID	int64	Unique identifier for a batch
DOCID	int64	Unique identifier for a document
DOCTYPE	categorical	The target class for the classifier
CONFIDENCE	object	Confidence levels for doc'types
AUTOCLASS	bool	Flag for manually labeled documents
PAGECOUNT	int64	# of pages of the document
DOCCOUNT	int64	# of documents belonging to a document
INPUTCHANNEL	categorical	Source of origin (Email, Fax, ...)
SOURCESYSTEM	categorical	Source of origin (System A, B, ...)
NBR_DOCTYPES	int64	# of d'types suggested by the DMS

Table 3-1: Structure of the raw data procured from the production system

This dataframe illustrates the format of all raw data retrieved weekly. It serves as the basis for further preprocessing steps before the data has clearance to be used for model training.

### 3.2 Anonymization of sensitive Data

Given the sensitive nature of the documents a thorough data risk assessment has to be conducted and an anonymization concept to be developed to get clearance for model training from the data protection officer. The below outlined routine of data protection and cleaning preprocessing is applied to every new batch of data retrieved:

1. **Blacklist check:** To remove particularly sensitive groups of senders (i.e. Rolls Royce© clients) a set of 109k email addresses is matched against the content of each document: First the document is tokenized using an existing open sourced

German language model<sup>19</sup> and then scanned for positive matches. Documents with positive matches have to be excluded from the training. This routine removes typically between 6-9% of documents in a procured data batch.

2. **Rough cleaning:** The remaining documents in the batch are cleaned and shortened with a number of string operations: Non-printable characters (i.e. “\t” or “\n”) are removed and typical conversational phrases and boiler plate copy is replaced with a short tag (i.e. “Sehr geehrte Damen und Herren,” is tagged „<Anrede>“ or “BMW Financial Services BMW Bank GmbH” is tagged „<BMWB>“).
3. **Regex rules tagging critical data:** Sensitive personal data contained in the document is identified by application of a battery of regular expression rules checking on the text. Positive matches are replaced with a specific tag like “<KFZK>” for a car registration plate. Appendix xy provides a listing with all rules for identifying and replacing sensitive information.
4. **Maximum length of documents:** Observing computational cost every document is trimmed to a maximum limit of 10k characters. Prior analysis confirmed that ~10% of the documents are shortened this way, leaving ~90% unchanged.
5. **Named Entity Recognition:** Documents are to be screened for contained personal data, specifically peoples’ names and addresses. Using the SpaCy software and its pre-trained German Language Model a Named Entity Recognition (NER) is performed on every tokenized document. Tokens recognized as peoples’ names are replaced with the tag “<PER>”, Locations such as city names are replaced with “<LOC>” and organizational names (i.e. “... GmbH”) are tagged with “<ORG>”.

Given the example above applying this routine produces this cleaned version of that document:

```
'Page 1 From: "<ORG>" <EMAIL> Date: Thu, 17 Dec 2020 11:18:36+0100 To: \<ORG>\\"<EMAIL> Subject:  
Kunde: <PER>, Muster Weg 5, <PLZ> <LOC> Verkehrsunfall vom: 11.12.2020 Pkw, aml. Kennz.: <KFZK> -  
<PER> ./ <PER> - Unser Zeichen: 1234/20-GG Sehr geehrte Damen und Herren, Ihr o.g. Kunde hat uns mit der  
zivilrechtlichen Schadensregulierung betreffend oben genannten Verkehrsunfall beauftragt ...'
```

The preprocessed and anonymized dataframes are persisted on the HSDAP. Training data cleaned and cleared (data protection and security regulations) is then uploaded onto a Deep Learning Platform (DLP) of the BMW AG, equipped with GPU resources and dedicated to model development and training.

---

<sup>19</sup> All preprocessing steps are conducted using the SpaCy German Language Model “de\_core\_news\_md”. Full documentation and detailed description can be found at <https://spacy.io/models/de>.

### 3.3 Explorative Data Analysis (EDA)

Within several weeks the routine described above yielded a total consolidated sample of labeled training data with  $N = 62734$  records, accounting only for the manually classified documents that the DMS failed to classify automatically. As shown in Table 3-1 the data retrieved provided for the texts but also for meta data regarding the input channels, the number of pages, etc. and necessarily for training classifiers the target information:

#### 3.3.1 Document Types (Labels)

A document type maps a document to a specific business related intention. Based on its document type a document is allocated to the relevant inbox. Document types can describe very department specific intents but also they can be of a generic nature, occurring in the theme of business for different departments. Thus they can be allocated across inbound traffic of multiple departments.

The data sample provides for 154 different document types. The left panel in Figure 3-1 depicts the distribution of instances of the 20 most frequent classes.

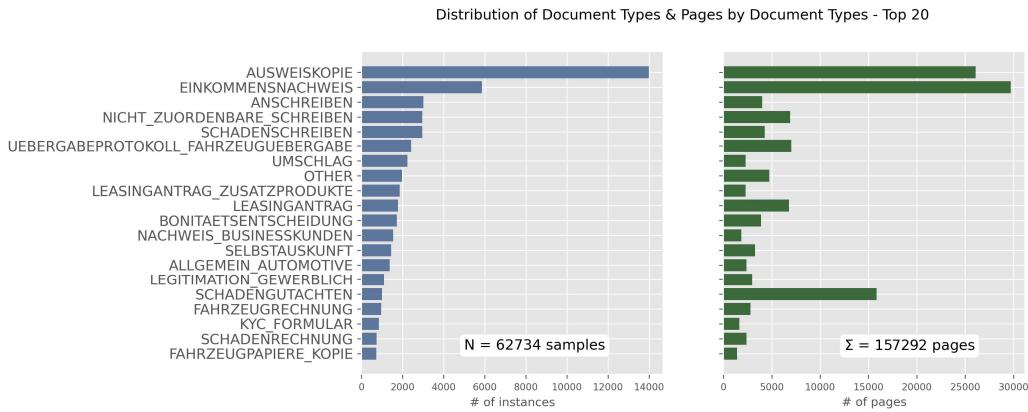


Figure 3-1: Distribution of the Top 20 Document Types (Instances & Page volume)

The distribution of the 154 different categories skews towards a small number of classes: The top 5 labels account for 46%, the top 10 for 62% and the top 20 for 80% of all  $N$  instances. With emphasis on frequent tasks in daily operations document types with an occurrence less than 100 instances in the sample (~2% of  $N$ ) are consolidated to a generic label ("OTHER"). Thus the consolidated sample provides for 59 classes, representing the most frequent 58 document types and the generic "OTHER".

Because the cost for manual inspection is driven by page volume the business objective is to minimize the number of pages (not documents) that require manual classification.

The right panel in Figure 3-1 illustrates the volume of pages related to each document type. That distribution corresponds with the frequency of documents. Some exceptions though exist where a category with low document count yet accounts for a high page volume (i.e. “SCHADENGUTACHTEN” representing ~10% of the total page volume). Hence the importance of the page count feature as a cost for misclassifications has to be considered. It will be addressed adequately with the setting of the metrics and the training parameters of the models (see chapter 4).

### 3.3.2 Other Meta Data

Table 3-1 displays a full list of all information retrieved in conjunction with the text. This meta data is provided by the DMS and can be used for additional features. There are 4 categorical features describing different aspects of the incoming route and technical format and 3 numerical features. Table 3-2: Distributions of categorical features shows the distribution of the categorical predictors in the sample of  $N$  documents:

Categorical Input Features						
BATCHKLASSE	BATCHCONTENT	INPUTCHANNEL	SOURCESYSTEM			
NeugeschäftUpload	.572	DS PDF Image + Text	.383	UPLOAD	.391	DOC_STORE .391
NeugeschäftSeitenBasiert	.132	DS PDF Image Only	.189	POST	.240	EMAIL .340
Bestand	.105	Schaden PDF Image + Text	.105	EMAIL	.230	GAA .240
Schaden	.105	BestandDoc	.104	FASTLANE	.122	FAX .021
Generic	.065	PDF Image + Text	.092	FAX	.015	OTHER .008
Banking	.021	GenericDoc	.065	OTHER	.003	
		PDF Image Only	.040			
		BankingDoc	.021			
		OTHER	.002			

Table 3-2: Distributions of categorical features

“BATCHKLASSE” describes different departmental units with responsibility for certain inbound channels (i.e. specific email addresses like “schaden@bmw.de”). “Neugeschäft” is the predominant subject area (department) accounting for more than 70% of all documents. This department provides an electronic upload channel that BMW retailers use to transfer contractual documents related to new leases and loans. This type of communication accounts for 57% of all inbound traffic. Given the thematic relationship with different business units the batch class feature might serve as a strong descriptor for the document classification task.

The “BATCHCONTENT” feature describes technical aspects of the documents received. More than 80% of all documents are derived from a source that applies an Optical Character Recognition (OCR) scan of an image. In 23% of them the original source is an image without any further text. Those documents are occurring in batch classes “Neugeschäft” (~70%) and “Schaden” (~10%), two departments that handle many documents with pictures (e.g. ID cards, certificates, scans, etc.). The substantial volume

of OCR-processed images and scans is contributing to the poor quality in text and the very noisy nature of the documents (see below).

“INPUTCHANNEL” and “SOURCESYSTEM” are very similar in that they describe the origin of the document with some variation in perspective between the two. Again the dominance of uploaded content shows in the categories “UPLOAD”, “FASTLANE” and “DOCSTORE”.

“PAGECOUNT”, “DOCCOUNT” and “NBR\_DOCTYPES” are numerical features. The first describes the number of pages for each document. DOCCOUNT describes how many other documents a single document is related to<sup>20</sup> and the number of document types (“NBR\_DOCTYPES”) is calculated from a list of document types that the DMS has already inferred on during the classification attempt. Figure 3-2 shows the distribution of the 3 numerical predictors split by different batch classes:

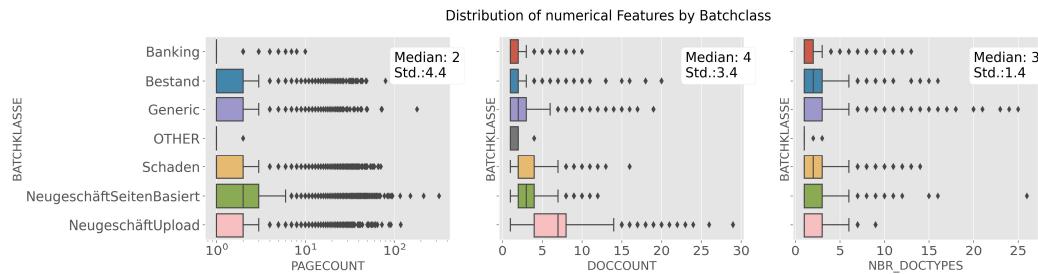


Figure 3-2: Distributions Numerical Features by Batch Class

Regarding the volume of pages with a median page count of 2 pages, the distribution illustrates many outliers, some of them even exceeding 100 pages. Looking at “DOCCOUNT” the distribution in the category “NeugeschäftUpload” shows a higher variance underlining that the documents in this batch class frequently have multiple interrelations to other documents. As to the number of document types the DMS typically estimates around 3 (median) different document types to be valid candidates, but looking at the outliers there is uncertainty listing up to 25 different document types as potential classification candidates for a given document.

For model training the categorical features are encoded using the OneHotEncoder class of the Scikit-learn package. The numerical features are normalized using the StandardScaler class in Scikit-learn to mitigate unfavorable bias effects of the outliers.

---

<sup>20</sup> Several single documents can be interrelated. For example a new customer might send an application for a lease together with more documentation of income, id-cards, proof of address, etc..

### 3.3.3 Text Data

Expectably the content of the  $N = 62734$  documents is the most important predictor for the intent classification task. The overall distribution of character and word count of the documents is depicted in Figure 3-3:

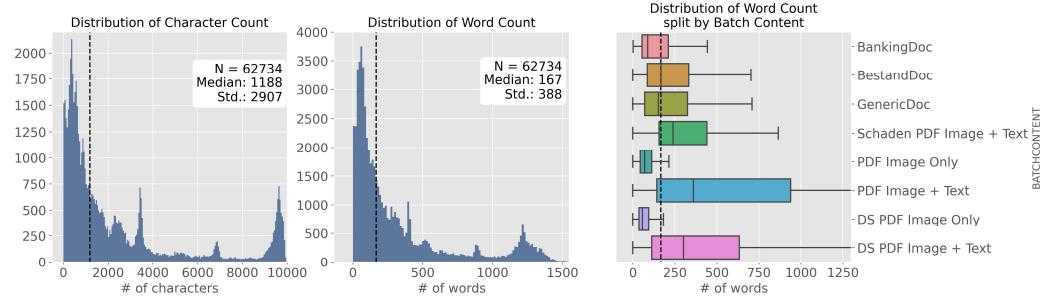


Figure 3-3: Distribution of word and character count in the  $N$  documents sample

The median length of a document is 1188 characters or 167 words with a standard deviation (Std.) of 2907 characters or 388 words. 80% of the documents contain 536 words or less with an average word length of 7 characters.<sup>21</sup>

A split analysis of word count by technical property “BATCHCONTENT” (right panel in Figure 3-3) illustrates that documents deriving from an image scan (OCR) with accompanying text (like a documentation of income etc.) contain significantly more words than average documents and documents with the property “Image Only” (examples would be a driving license or an ID-card) are significantly below the average documents regarding their word count.

This effect translates directly into the distribution of word count split by document types. Figure 3-4 shows the 15 most frequent document types in the sample and their respective word count distribution.

The most frequent document type is “AUSWEISKOPIE”. Naturally these documents do not contain a lot of text. “SCHADENGUTACHTEN” (damage assessment) on the other hand, an important document type contributing significantly to total page volume (see above) shows a much higher variance and median of word counts.

---

<sup>21</sup> Floating points have been converted to full integers for this purpose.

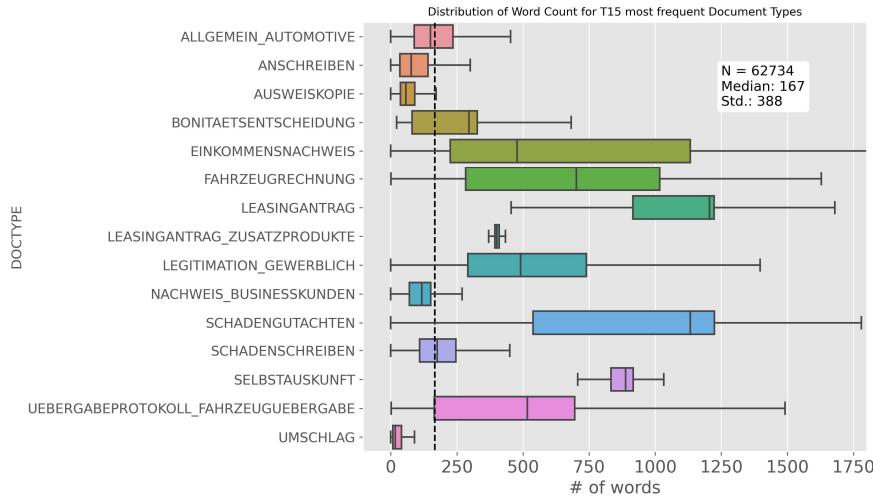


Figure 3-4: Distribution of word count split by Document type

Given the heterogeneous distributions of word count by different document types the length of the document might serve as a strongly contributing predictor in the classification task. Though potentially more words in a document might not necessarily help but rather introduce more noise and clutter. Given the importance of the OCR technology involved and the observation that the quality of the scans is often questionable, it is critical to understand how much “noise” really is contained in the data extracted.

There is no standard metric to operationalize “amount of noise” in a corpus. Therefore an approximation is applied: The vocabulary built of the given sample is compared to large vocabularies extracted from publicly available large German text corpora. For that a shared vocabulary of the following three publicly available sources<sup>22</sup> is build and used as a proxy for a gold standard:

- SpaCy German Language Model Large with 500k unique words
- Deepset.AI German Glove vectors with 1.3 million unique words
- 10k German News Dataset with 178k unique words.

The three dictionaries are consolidated using the set operations in python. The resulting gold standard vocabulary contains 1.493 million words that can be matched against the given vocabulary of the sample corpus with  $N = 62734$  documents.

---

<sup>22</sup> © ExplosionAI GmbH. For full documentation please refer to [https://spacy.io/models/de#de\\_core\\_news\\_lg](https://spacy.io/models/de#de_core_news_lg)  
 © deepset GmbH. For full documentation please refer to <https://deepset.ai/german-word-embeddings>  
 © Timo Block. For full documentation please refer to <https://tblock.github.io/10kGNAD/>

The vocabulary for the entire corpus but also for every single document type is build up with the usage of the Scikit-learn CountVectorizer class:

The total vocabulary size of the sample corpus is 979k expressions. Matching the 10k most frequent expressions of this sample corpus with the gold standard shows that 38% of these tokens are not represented in the gold standard dictionary.

A split analysis of “noise by document types” yields the results shown in Table 3-3. The ten “most noisy” document types are listed in the left and the “least noisy” document types in the right panel.

Comparison of Sample Vocabulary versus Gold Standard Vocabulary					
Top 10 most "noisy" categories	Vocabulary Length	Share of Non-Matches in 10k most freq. Words	Lowest 10 most "noisy" categories	Vocabulary Length	Share of Non-Matches in 10k most freq. Words
DARLEHENSANTRAG	51521	0.620	WIDERRUF_VERTRAG	3383	0.160
LEASINGANTRAG_ZUSATZPRODUKTE	31205	0.602	ANFORDERUNG_UNTERLAGEN	4450	0.181
SELBSTAUSKUNFT	85260	0.601	ALLGEMEIN_AUTOMOTIVE	24678	0.206
KYC_FORMULAR	20133	0.599	ANFRAGE_VERTRAGSAENDERUNG	10673	0.226
LEASINGANTRAG	199791	0.588	ABLOESE_INTERN	3267	0.244
DARLEHENSANTRAG_ZUSATZPRODUKT	18768	0.587	EINKOMMENSNACHWEIS	170012	0.248
VERTRAGSBESTAETIGUNG	10001	0.587	ADRESSEAENDERUNG	6273	0.249
FAHRZEUGPAPIERE_KOPIE	45835	0.580	SONDERGENEHMIGUNG	4073	0.253
RUECKNAHMEPROTOKOLL	52657	0.567	ALLGEMEIN_BANKING	7473	0.257
AUSWEISKOPIE	165790	0.563	SCHADENSCHREIBEN	31369	0.26

Table 3-3: Analysis of the most and least noisy Categories

The 59 document types yield different total vocabulary lengths ranging from 3383 words to 199791 words with a median of 11449 words. Regarding the “share of noise” the results spread from the noisiest category “DARLEHENSANTRAG” with 62% to the least noisy category “WIDERRUF\_VERTRAG” with only 16% not found in the gold standard. Measuring the correlation of the length of the vocabulary and the share of noise yields a Pearson Correlation Coefficient,  $R^2$  of 0.32, indicating a weak positive co-occurrence of longer vocabularies and more noisy text.

Measured across all document types a median noise level of 36% summarizes the assessment that the data in the given sample corpus is very noisy with more than every third token not matching against the gold standard benchmark. The data is largely received from preprocessing systems (OCR) that in addition to an already poor quality of the raw document (photocopies etc.) introduces additional bias during processing.

With this insight of the training source data being as noisy as shown, applying different classification approaches like the ones discussed in chapter 2 might yield very different results. State of the art methods pulling their strength from their semantic capabilities might not perform as superior in this real life environment as they do in many of the public or academic benchmark examples.

## 4 Experiments

This chapter will introduce different general approaches and varying classification methods applied within these routes. The experiments are run in three separate flights:

- (a) **Traditional approaches** using classic algorithms long established in machine learning theory together with a BOW/TF-IDF vectorization of the text.
- (b) **Deep Learning approaches**, varying model architectures of the Deep Learning field leveraging embedding vectors as textual input.
- (c) **Transfer learning** leveraging a widely recognized pre-trained transformer model implementation for the encoder part of the network architecture.

The focus is on a brief illustration of the models and architectures used and their key parameters set. Python code examples for the experiments are documented in the appendix. Chapter 5 will follow-up with a detailed discussion of the results from the three different routes of the experiments.

### 4.1 Evaluation & Metrics

The sample of  $N = 62734$  documents representing 59 different classes is split into a stratified train and test set with a ratio of .75 and .25, preserving the class distribution for the document type label of  $N$  in both partitions.

All experiments are validated on the  $N_{test} = 15684$  documents. With the business objective set on minimizing the volume of manually inspected pages every class is reported with its page volume weighted F1 score, the harmonic mean of its page volume weighted precision and recall score. The mean of the 59 specific F1 scores is used to express overall model performance (weighted F1).

The experiments are conducted on the DLP with a Python 3.8 runtime environment. The traditional machine learning approaches portrayed in chapter 4.2 are implemented using the Scikit-learn API (version 23.0.2). The Deep Learning experiments described in chapter 4.3 are implemented with the TensorFlow<sup>©</sup> framework (version 2.3) and the Keras<sup>©</sup> API. The Transformer Model experiment explained in chapter 4.4 makes use of the Transformers software library (version 3.5.1).

### 4.2 Machine Learning Approaches

A preprocessing pipeline is likewise applied to the train and test data: A Column Transformer object is instantiated with a One-Hot-Encoder for the categorical

variables, a standard-scaler for the numerical predictors and a TF-IDF vectorizer for the text. The latter applies lowercase conversion and builds a vocabulary of size 10000.

This pipeline is used for all experiments described in this chapter. The classifiers applied during the experiments are briefly introduced below with a focus on their key parameter settings. The Scikit-learn API provides extensive documentation<sup>23</sup> of all details to the respective classes used and their default parametrization.

#### 4.2.1 Baseline Classifiers

For a baseline starting point two models are applied that require no or limited parametrization and offer a fast training in return:

- Multinomial Naïve Bayes
- $k$ -NN Classifier

**Multinomial Naïve Bayes** classifier is a multi-class adaption of the classic Naïve Bayes classification algorithm. A probabilistic classifier founded on Bayes' theorem<sup>24</sup>. Its key principle is the calculation of a posterior probability for an instance belonging to a certain class, based on the prior and conditional probabilities of its features. This simple model requires no parameter tuning.

The performance of the  **$k$ -NN** classifier depends on the number of neighbors  $k$ , the weighting scheme amongst them and the distance function to compute neighborhood amongst all instances in the training set. A variety of different parameter combinations is applied to find the best performing combination using different values for  $k$ . Table 4-1 shows the key parameter settings applied during the  $k$ -NN experiments.

# Classifier (class)	Parameters	Parameter Values #
$k$ -NN Classifier	Number of neighbors = #	1, 2, ..., 10, 15, 20, 25, 30, 40, 50, 75, 100, 150, 250, 500, 1000
<code>skLearn.neighbors.KNeighborsClassifier()</code>	Weights = #	> distance > uniform
	Distance Function = #	> manhattan distance > euclidian distance

Table 4-1:  $k$ -NN Classifier with Key Parameter Settings

---

<sup>23</sup> Full documentation on: <https://scikit-learn.org/stable/>

<sup>24</sup> Bayes theorem is founded on the works of Reverend Thomas Bayes in the 18<sup>th</sup> century.

## 4.2.2 Linear Models

Building on the base lines above different linear models are employed:

- Logistic Regression Classifier
- Linear Support Vector Machine
- Stochastic Gradient Descent

Designed for binary classification, the **Logistic Regression** classifier is set to an OVR scheme to serve the multi-class context given here. The algorithm is fed a weighting scheme by a category-weight dictionary, thus allowing for the page volume to be factored into the training. The choice of solvers depends on the setting of penalty and other parameters. Regularization is controlled with the L1 and L2 regularization norm.

# Classifier (class)	Parameters	Parameter Values #
<b>Logistic Regression</b> <i>sklearn.linear_model.LogisticRegression()</i>	penalty := Regularization norm	> l1 > l2
	class_weight	> None > Dict {"DOCTYPE": PAGECOUNT}
	multi_class	ovr (one vs. rest scheme)
	solver	lbfgs
	max_iter	1000

Table 4-2: Logistic Regression Classifier with Key Parameter Settings

The **Linear Support Vector Machine** classifier is set to OVR. It provides for L1 and L2 regularization and accepts a weighting scheme like Logistic Regression above. Scikit-learn offers a specific Linear SVM class, optimized for linear problems: Omitting the usage of kernel functions, it trains significantly faster than the standard SVM classifier class with kernel functions (see below). An optimal setting for the regularization parameter  $C$  is imperative for good results.  $C$  scales the impact of the slack variables (see Equation 2-7).

# Classifier (class)	Parameters	Parameter Values #
<b>Linear Support Vector Machine</b> <i>sklearn.svm.LinearSVC</i>	penalty := Regularization norm	> l1 > l2
	loss	> hinge > squared_hinge
	class_weight	> None > Dict {"DOCTYPE": PAGECOUNT}
	multi_class	ovr (one vs. rest scheme)
	C := Regularization Strength	0.55, 0.66, 1.0, 1.5
	dual := Dual or primal problem	False

Table 4-3: Linear Support Vector Machine with Key Parameter Settings

The **Stochastic Gradient Descent Classifier** (SGD) class is not an algorithm in its own right but a technically optimized framework that provides for efficient training with Stochastic Gradient Descent learning. It's tailored to high-dimensional machine learning problems like text classification. With a choice of different convex loss functions it emulates different algorithms: While the usage of the default "hinge loss" function results in a linear SVM, the usage of a "log loss" function delivers a logistic regression (Anon., 2021).

# Classifier (class)	Parameters	Parameter Values #
Stochastic Gradient Descent Classifier	penalty := Regularization norm	> l1 > l2
	loss	> hinge, squared_hinge > modified_huber > log_loss
sklearn.linear_model. SGDClassifier	class_weight	> None > Dict {"DOCTYPE": PAGECOUNT})

Table 4-4: Stochastic Gradient Descent Classifier with Key Parameter Settings

#### 4.2.3 Non-linear Classification Methods

Beyond linear methods further non-linear techniques are applied:

- Support Vector Machine using kernel functions
- Gradient Boosting

Kernel functions enable **SVMs** to fit a hyperplane in high-dimensional space even when data is not linearly separable. The Support Vector Classifier class provides for kernel functions like a Polynomial, a Radial-Basis Function or a Sigmoid. As described for the Linear SVM the optimal setting of the regularization parameter  $C$  is important.

# Classifier (class)	Parameters	Parameter Values #
	C := Regularization Strength	0.55, 0.66, 1.0, 1.5
Support Vector Machine	kernel := Kernel Function	> linear > poly > rbf > sigmoid
sklearn.svm.SVC	class_weight	> None > Dict {"DOCTYPE": PAGECOUNT})
	decision_function_shape	OVR

Table 4-5: Support Vector Machine with Key Parameter Settings

**Gradient Boosting** is another non-linear machine learning technique. The estimator is made up of an ensemble of several weak learners. In a forward stage-wise fashion the algorithm sequentially adds new models to the ensemble at every learning step. Every new predictor builds on the mistakes of the previous one. Typically tree-based models

are used and fitted to the residual errors of the last iteration. (Géron, 2019, p. 203). A key parameter is the number of  $m$  estimators to produce (equivalent to the number of learning steps). In a multi-class task the Scikit-learn implementation fits a number of regression trees equivalent to the number of classes at every learning step. Thus the algorithm produces a total of  $n$  classes \*  $m$  estimators. Equipped with an early-stopping logic the number of estimators can be set to a high number. The model stops producing new trees if no further improvement measured on a validation set is observed over a defined number of training steps. Since the weak learners in the ensemble are built of decision tree models, there comes a wide range of parameters to be customized for controlling the tree production (growing and pruning). For the experiments applied herein the default parametrization is used.

# Classifier (class)	Parameters	Parameter Values #
<b>Gradient Boosting</b>  <i>skLearn.ensemble.GradientBoostingClassifier</i>	loss	deviance (logistic regression)
	n_estimators	100
	max_features	auto := sqrt(n_features)
	validation_fraction	0.15
	n_iter_no_change	5
	learning_rate	0.1
	subsample	0.5, 0.66, 0.8, 1.0

Table 4-6: Gradient Boosting Classifier with Key Parameter Settings

### 4.3 General Training Architecture

As described in chapter 3 the training data provides textual input and for additional features with interesting properties potentially supporting the classification. The Tensorflow/Keras Functional API allows to design model architectures accepting different inputs (text, categorical and numerical) with separate (pre)processing strands to them. With this flexibility a General Training Architecture (GTA) for the Deep Learning experiments is built:

- The numerical and categorical input is preprocessed with scaling and One-Hot-encoding (OH) as described in chapter 4.2 before being fed to the GTA.
- For the Deep Learning experiments<sup>25</sup> the textual input is truncated or padded to

---

<sup>25</sup> The Transformer Model chosen requires a specific preprocessing and tokenization for the input texts. The herein described vectorization and embedding steps are therefore only applicable to the Deep Learning experiments described in chapter 4.4.

a sequence length of 512 tokens. The vocabulary size is capped to the most frequent 10000 words. Vectorization is performed using the TextVectorization class<sup>26</sup>. It performs lowercasing and stripping of punctuation on the input sequences, creates an integer mapping for each word and transforms the input sequence into a vector of respective integers.

- The vectorized (text) sequences of 512 integers (representing one document) are fed to a plain Embedding Layer<sup>27</sup>. The Embedding Layer is trained alongside the training process as the gradient signal propagates backwards through the network. Some experiments leverage pre-trained embedding vectors for initialization of the Embedding Layer (see below).
- The 300-dimensional output from the Embedding Layer is fed to a Text Model Layer. This Text Model layer is implemented with different Deep Learning architectures within different experiments described in chapter 4.4.

After preprocessing and application of the Embedding Layer and the Text Model all inputs are concatenated and fed to a dense Feed Forward Network:

- Two Dense Layers with 128 neurons and 64 neurons respectively, initialized with a “he-initialization” scheme and activated with a ReLU function.
- A final Dense Layer with  $n\_classes = 59$  neurons and Softmax activation producing the networks’ output vector of probabilities for each class.

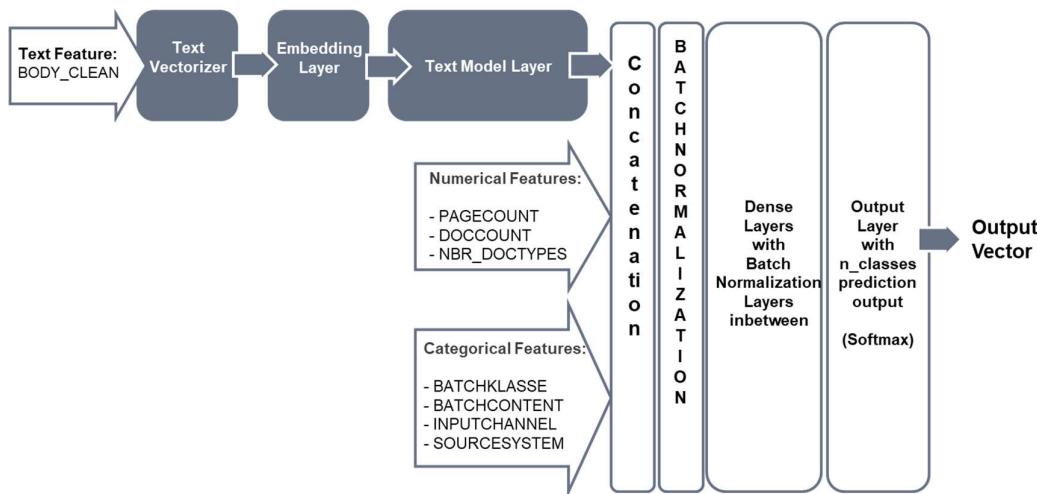


Figure 4-1: General Training Architecture used for the Deep Learning Experiments

---

<sup>26</sup> [https://www.tensorflow.org/api\\_docs/python/tf/keras/layers/experimental/preprocessing/TextVectorization](https://www.tensorflow.org/api_docs/python/tf/keras/layers/experimental/preprocessing/TextVectorization)

<sup>27</sup> [https://www.tensorflow.org/api\\_docs/python/tf/keras/layers/Embedding](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding)

During the learning process the networks parameters are incrementally updated at each learning step to minimize a Cross Entropy loss function (described in chapter 2.3.1.1). Using the Sparse Categorical Crossentropy implementation<sup>28</sup> enables training with labels encoded as integers (and not as OH-encoded arrays).

The RMSprop optimizer is chosen with its default settings kept. The learning rate is set to  $1.0 \times 10^{-4}$  or  $1.0 \times 10^{-5}$  and training data is fed to the network in batches of size 64 or 32. To leverage the cost information (page count) the instances are additionally weighted during the training process in some experiments using a pre-computed class-weights dictionary with normalized page count volume for each class.

Different regularization techniques are experimented with: Batch Normalization (BN) layers are finally chosen over Drop out layers. They provide for stability as well as regularization during training. One BN layer is implemented after the Concatenation Layer combining the three input strands and again one BN layer is used between each Dense Layer. All training runs are monitored with Early-Stopping-Callbacks for additional regularization: Training stops with no further improvement on the loss of a validation set (a .2 fraction of the training data). In case of Early Stopping the most performant (last stored) version of the model can be reloaded from disk and applied for the final evaluation to the held-out test set.

## 4.4 Deep Learning approaches

This chapter briefly describes several experiments run with different Deep Learning ideas built into the GTA (Embedding and Text Model Layer in Figure 4-1).

### 4.4.1 Bidirectional Long Short Term Memory

At first the Text Model Layer is implemented with a Bidirectional Long Short Term Memory model (BiLSTM). It's built of two LSTM layers combined in a bidirectional wrapper<sup>29</sup>. This allows for processing textual input in a forward and backward fashion. All hyper parameters are kept within their default settings.

The 600-dimensional output (two 300-dimensional vectors concatenated) of the BiLSTM is fed into the concatenation layer together with the numerical and categorical input and processed in the described GTA (Figure 4-1).

---

<sup>28</sup> [https://www.tensorflow.org/api\\_docs/python/tf/keras/losses/SparseCategoricalCrossentropy](https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy)

<sup>29</sup> [https://www.tensorflow.org/api\\_docs/python/tf/keras/layers/Bidirectional](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Bidirectional)  
[https://www.tensorflow.org/api\\_docs/python/tf/keras/layers/LSTM](https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM)

#### 4.4.2 Convolutional Neural Nets

For different CNN configurations the Text Model Layer is implemented with one or more CNN Layers followed by a Max Pooling Layer<sup>30</sup> before the output is fed to the subsequent layers of the GTA. As part of the experiments varying combinations of number of filters and kernel sizes (the length of the 1D convolutional window) are applied:

- 128 filters and 5 kernel
- 300 filters and 5 kernel
- 300 filters and 7 kernel
- 512 filters and 7 kernel

To measure the impact of applying different window sizes (kernel sizes) parallelly in one model some experiments use two parallel CNN layers in the architecture with a concatenation layer. Their setting is a combination of 300 filters/5 kernel and 300 filters/7 kernel.

All experiments are run with ReLU activation functions on the CNN layers. By applying a “narrow convolution” there is no padding on the inputs and the number of strides and all other hyper parameter are set to default value.

#### 4.4.3 CNN with pre-trained Embeddings

The experiments described above make use of a plain Embedding Layer that is initialized (by default) randomly from a uniform distribution. With training alongside the network these weights are automatically fine-tuned. But the limited amount of training data with 62k documents only is constraining the learning of sufficiently good embeddings. Pre-trained open-sourced German embedding vectors can be used alternatively to mitigate this shortcoming. Having been trained on large text corpora these vectors can provide a starting point for faster training and conversion on better results.

Three German embeddings are downloaded and experimentally applied in combination with the CNN models described above. An embedding matrix is constructed for every source by mapping the training vocabulary to the respective pre-trained vectors. But noisy data (see chapter 3.3.3) is challenging: The pre-trained embedding vectors match only partially with the training vocabulary. Every mapping creates a considerable share

---

<sup>30</sup> The Tensorflow/Keras Conv1D class and GlobalMaxPool1D class are used as building blocks for this step:  
[https://www.tensorflow.org/api\\_docs/python/tf/keras/layers/Conv1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv1D)  
[https://www.tensorflow.org/api\\_docs/python/tf/keras/layers/GlobalMaxPool1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalMaxPool1D)

of zero vectors in the embedding matrix (see Table 4-7 below). Thus limiting the potentially beneficial impact of these pre-trained vectors. Even in the best case (SpaCy German Language Model) every third expression of the training vocabulary is not met by a respective pre-trained embedding vector.

Pre-Trained Embeddings mapped to the Project's Vocabulary (10k)			
Source	Vocabulary Length	Dimensions	Share of Zero Vectors
SpaCy German Language Model	500000	300	.332
Deepset AI German GloVe vectors	1309281	300	.361
Deepset AI German Word-2-Vec vectors	854776	300	.452

Table 4-7: Zero Vectors from Pre-Trained Embeddings

For each experiment the respective embedding matrix is used for weight initialization of the Embedding Layer within the network and all experiments are run with the embedding matrix set to “trainable” throughout the learning process.

## 4.5 Bidirectional Encoder Representations from Transformers

In 2018 a team of Google researchers introduced BERT: “Bidirectional Encoder Representations from Transformers” (Devlin, et al., 2018). A language representation model developed on the basis of the Transformer Model (Vaswani, et al., 2017) and other ideas of the NLP community.

BERT is a pre-trained Transformer Encoder stack. It comes in a BERT<sub>BASE</sub> version and a BERT<sub>LARGE</sub> version. BERT<sub>BASE</sub> incorporates 12 encoder layers (called “Transformer Blocks”), 12 attention blocks and 768 hidden units in the feed-forward network. BERT can learn bidirectional representations from left and right contexts using the self-attention logic. *“As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks ... ”* (Devlin, et al., 2018, p. 1).

Since then numerous pre-trained BERT models and variations thereof have been published and open sourced. Google© offers a multi-lingual version<sup>31</sup> trained on 104 languages and their largest Wikipedia corpuses. This BERT multilingual base model is available in a case-sensitive (cased) or an uncased version. The Transformers framework from HuggingFace Inc.© offers a general API to a variety of language

---

<sup>31</sup> See documentation of the model: <https://huggingface.co/bert-base-multilingual-uncased?>

models including different BERT models. Using this API the Google BERT multilingual model is downloaded and integrated into the GTA as shown in Figure 4-2.

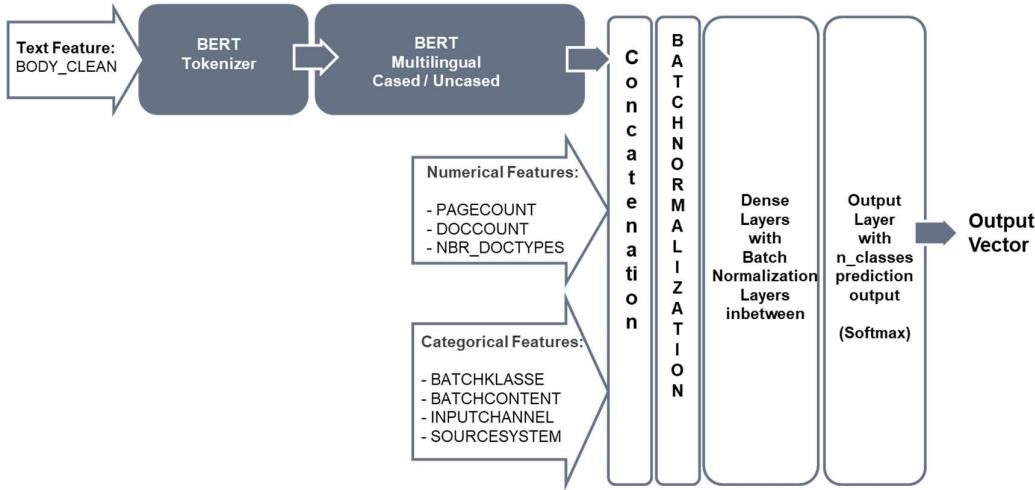


Figure 4-2: BERT Multilingual & General Training Architecture

The BERT implementation from Google used in the experiments trains more than 177 million parameters. Thus the training of one epoch range takes roughly 20 minutes on the DLP. Given a computational budget to be observed the BERT experiments are capped to a maximum of 20 epochs for training. Due to memory limitations on the DLP the experiments are run with a maximum batch size of 32 and a maximum sequence length of 384 tokens. Thus limiting the potential capacity of the original BERT model which is designed for a maximum sequence length of 512 tokens.

Experiments are run with the cased and uncased version likewise using a learning rate of  $1.0 \times 10^{-4}$  and class weights (page volume) applied during training.

The textual input is transformed using a dedicated tokenization scheme required for the BERT model<sup>32</sup>. This specific vectorization is performed with a pre-defined tokenizer class available through the HuggingFace API<sup>33</sup>. The BERT model layer is configured for not outputting its hidden states.

---

<sup>32</sup> This BERT implementation uses a WordPiece tokenizer with a vocabulary of 106k expressions (multilingual). Special tokens are introduced to mark the beginning of a sequence ([CLS]), separate different sentences ([SEP]) into segments and a positional index to represent the sequential order of the input (Devlin, et al., 2018, pp. 4-5).

<sup>33</sup> Documentation: [https://huggingface.co/transformers/model\\_doc/bert.html#berttokenizerfast](https://huggingface.co/transformers/model_doc/bert.html#berttokenizerfast)

## 5 Results

The results of the experiments are grouped in three flights as described in chapter 4. All Results are reported on the test data set with  $N_{test} = 15684$  instances (25% of  $N$ ).

After discussing each group of experiments the two best performing models are compared to one another to identify individual strengths and weaknesses. A final experiment synthesizes these individual strengths into a combined ensemble estimator.

Finally this chapter will conclude with a discussion of limitations on the experiments and possible improvements and ideas for further exploration and projects.

### 5.1 Machine Learning Models with BOW/TF-IDF vectorization

The Scikit-Learn API allows for the application of numerous classifiers using a standardized input scheme. Chapter 5.1.1 gives an overview of the top line results for the different algorithms employed, before chapter 5.1.2 focusses on the best performing model with an in-depth exploration of its results.

#### 5.1.1 Machine Learning Classifiers – Top Line Results

Two first baseline classifiers requiring only little hyper parameter tuning are employed.

- The Multinomial Naïve Bayes classifier delivers an instant .842 weighted F1 baseline when trained with a weighting scheme to inform the prior-probabilities of the classes.
- The  $k$ -NN classifier falls short compared to this, despite numerous test runs with different distance functions and values of  $k$ . Euclidian distance function performs better than Manhattan distance function, but the baseline set by the Multinomial Naïve Bayes classifier is not met by  $k$ -NN.

Top Line Results - Machine Learning Models with BOW/TF-IDF: Baseline		
Model	Parameter	Weighted F1
Multinomial Naive Bayes	fit_prior = w. uniform distribution	.835
Multinomial Naive Bayes	class_priors = array w. page weights	.842
$k$ -NN Classifier	euclidian dist., uniform weights, k=1	.818
$k$ -NN Classifier	euclidian dist., distance weights, k=3	.824
$k$ -NN Classifier	manhattan dist., uniform weights, k=1	.749
$k$ -NN Classifier	manhattan dist., distance weights, k=2	.75

Table 5-1: Baseline Classifiers Experiments: Weighted F1 Results

With the baseline set, different linear models are employed to the document classification task.

Top Line Results - Machine Learning Models with BOW/TF-IDF: Linear Models		
Model	Parameter	Weighted F1
Logistic Regression Clf.	penalty = L2, classweight=weighted	.737
Logistic Regression Clf.	penalty = L2, classweight=None	.889
Linear SVM	C=1.0, L2, squared_hinge, c'weight=None	.9
Linear SVM	C=1.0, L1, squared_hinge, c'weight=None	.894
Linear SVM	C=0.55, L2, squared_hinge, c'weight=None	.902
Linear SVM	C=0.55, L2, squared_hinge, c'wght=weighted	.872
Stochastic Gradient Descent	loss=log, L2	.863
Stochastic Gradient Descent	loss=modified_hubert, L2	.894
Stochastic Gradient Descent	loss=hinge, L2	.889

Table 5-2 shows the results of these linear model experiments.

Top Line Results - Machine Learning Models with BOW/TF-IDF: Linear Models		
Model	Parameter	Weighted F1
Logistic Regression Clf.	penalty = L2, classweight=weighted	.737
Logistic Regression Clf.	penalty = L2, classweight=None	.889
Linear SVM	C=1.0, L2, squared_hinge, c'weight=None	.9
Linear SVM	C=1.0, L1, squared_hinge, c'weight=None	.894
Linear SVM	C=0.55, L2, squared_hinge, c'weight=None	.902
Linear SVM	C=0.55, L2, squared_hinge, c'wght=weighted	.872
Stochastic Gradient Descent	loss=log, L2	.863
Stochastic Gradient Descent	loss=modified_hubert, L2	.894
Stochastic Gradient Descent	loss=hinge, L2	.889

Table 5-2: Linear Model Experiments: Weighted F1 Results

- With one exception (Logistic Regression) all linear models deliver F1 scores above baseline. F1 scores with weighted instances are below those runs without weighting scheme.
- Logistic Regression falls short when trained with class-weighting but scores considerably high achieving .889 weighted F1 when run without it.
- Linear SVM also yields much better results when applied without weighting scheme and performs even better when the margin around the decision boundary is softened with the  $C$  parameter reduced to .55.
- Training the models with Stochastic Gradient Descent delivers strong results against the baseline when emulating a SVM using the default hinge loss function or even better when applied using a Modified Huber loss function.

Given the complexity of the noisy data another flight of experiments is employing non-linear classification techniques to benchmark for the best approach. Table 5-3 summarizes the performance of SVM classifiers run with different kernel functions<sup>34</sup>.

---

<sup>34</sup> Applying a Support Vector Machine Class with a linear kernel function is not a non-linear method. The result of this experiment is added for comparison (\* in Table 5-3).

Top Line Results - Machine Learning Models with BOW/TF-IDF: Non-Linear Models		
Model	Parameter	Weighted F1
SVM w. Polynomial Kernel	C=0.55, classweight=None	.865
SVM w. RBF Kernel	C=0.55, classweight=None	.863
SVM w. Sigmoid Kernel	C=0.55, classweight=None	.828
SVM w. Linear Kernel*	C=0.55, classweight=None	.893*
Gradient Boosting		.811

Table 5-3: Non-Linear Model Experiments: Weighted F1 Results

- Using SVMs with varying kernel functions doesn't yield stronger results than the linear models reported above.
- The result of Gradient Boosting is notably poor with weighted F1 scores below the baseline. Overfitting the noisy data is a pitfall with the tree-based approach (weak learners are trees). Without extensive hyper parameter tuning to control the behavior of the tree production process the Gradient Boosting ensemble can't mitigate this.

### 5.1.2 Exploration on the Linear SVM performance

Summarizing the results reported above the Linear SVM with a low  $C$  regularization delivers the best weighted F1 (.902) in the first flight of experiments.

The business objective is to optimize those document types that cause a substantial amount of page load. Table 5-4 shows a ranking of F1 and the respective page volume weight (middle panel lists the Top 15 scoring classes, right panel lists the Low 15 scoring classes). The scatterplot in the left panel depicts the distribution of the page volume weights and the F1 scores for all 59 classes.

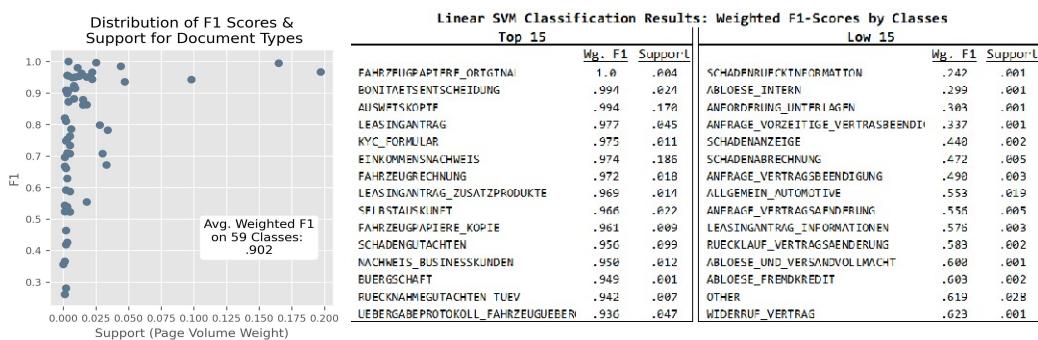


Table 5-4: Linear SVM – F1 & Support by Document Type

11 categories fall short of 60% F1, all of them with an insignificant contribution to the page load (support). 35 classes yield an F1 above 80% and 22 classes above 90%. Ranked by F1 the Top 15 classes shown in the middle panel of Table 5-4 account for 67% of the page volume in  $N_{test}$  while the Low 15 classes account for 7% (right panel).

Looking at a simplified confusion matrix (Figure 5-1 below) with the most error-prone classes two patterns emerge:

- (1) Categories describing a more generic semantic concept with less distinctive language are more likely to mislead the classifier. This can be shown with the document types "ALLGMEIN AUTOMOTIVE", "ANSCHREIBEN", "ALLGEMEIN BANKING", "NICHT\_ZUORDNABARE\_SCHREIBEN" and "OTHER". Not only do they overlap semantically with each other, they also provoke errors from other potentially more distinctive classes because they have partial overlap in language used. For example the classes related to documents referencing the change of terms and conditions (labels starting with "ANFRAGE\_") are frequently misclassified to the class "ALLGEMEIN AUTOMOTIVE".
- (2) Categories that are more distinct in meaning but are close to conceptually similar categories drive misclassifications. This can be seen with the categories containing documents that are related to the concept of "SCHADEN" (damage). Categories related to damage management are frequently misclassified to the category "SCHADENSCHREIBEN". This category is the more generic concept amongst the "SCHADEN"-related categories (with high incidence and page load).

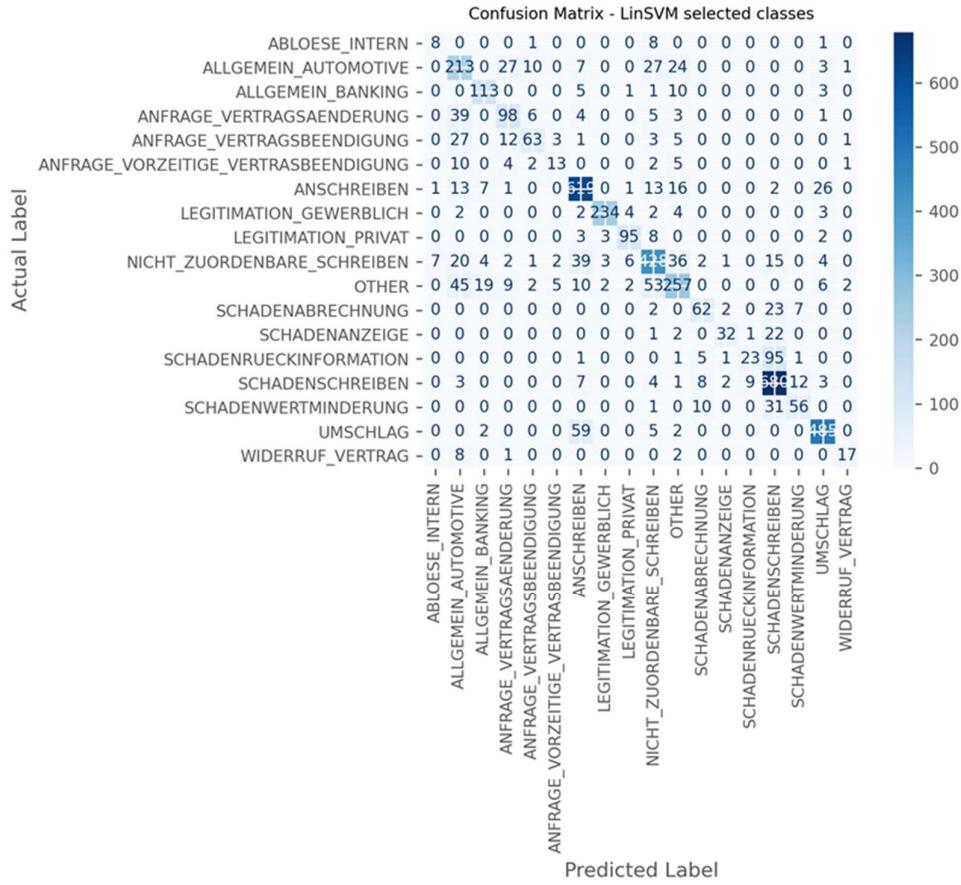


Figure 5-1: Linear SVM: Confusion Matrix for selected Classes

The coefficients of the Linear SVM classifier can be inspected for further insights to its prediction logic. The classifier object holds an array with shape 59 classes and 10030 coefficients that are used in the linear model. The coefficients can be ranked according to their absolute value as a proxy for measuring feature importance.

Linear SVM Classification - Coefficients (abs*) for selected Classes					
SCHADENSCHREIBEN [Wgth. F1=.810]		SCHADENRUECKINFORMATION [Wgth. F1=.242]		SCHADENGUTACHTEN [Wgth. F1=.956]	
Feature	Coeff.*	Feature	Coeff.*	Feature	Coeff.*
eur	2.824	zwischenzeitlich	1.243	PAGECOUNT	2.760
betrag	2.112	verzichtet	.981	rechnung	1.607
intro	1.899	abgemeldet	.975	bild	1.458
folgt	1.632	sonderkündigung	.938	kalkulation	1.358
telefax	1.565	haftpflichtversicherer	.922	mfg	1.016
rechnen	1.540	ihr	.847	anrede	1.003
PAGECOUNT	1.531	wie	.837	666666	.999
attachments	1.491	ihnen	.807	kfzkz	.968
anrede	1.435	bestandsnummer	.796	technische	.943
wertminderung	1.417	13	.790	auftrag	.931

Table 5-5 shows the feature importance of the 10 most distinctive features for three selected example categories, all related to the concept of damage ("SCHADEN") but with diverging F1.

Linear SVM Classification - Coefficients (abs*) for selected Classes					
SCHADENSCHREIBEN [Wgth. F1=.810]		SCHADENRUECKINFORMATION [Wgth. F1=.242]		SCHADENGUTACHTEN [Wgth. F1=.956]	
Feature	Coeff.*	Feature	Coeff.*	Feature	Coeff.*
eur	2.824	zwischenzeitlich	1.243	PAGECOUNT	2.760
betrag	2.112	verzichtetet	.981	rechnung	1.607
intro	1.899	abgemeldet	.975	bild	1.458
folgt	1.632	sonderkündigung	.938	kalkulation	1.358
telefax	1.565	haftpflichtversicherer	.922	mfg	1.016
rechnen	1.540	ihr	.847	anrede	1.003
PAGECOUNT	1.531	wie	.837	666666	.999
attachments	1.491	ihnen	.807	kfzkz	.968
anrede	1.435	bestandsnummer	.796	technische	.943
wertminderung	1.417	13	.790	auftrag	.931

Table 5-5: Linear SVM – Feature Importance for selected semantically related Classes  
(\*Coefficients are presented with their absolute value)

- Looking at “SCHADENSCHREIBEN” a category that is moderately well handled by the classifier (F1 .81) the ranking is led by rather generic words like “eur”, “betrag” or “intro”. The classifier is not leveraging on very distinctive language for this class.
- “SCHADENRUECKINFORMATION”, the class with the lowest F1 (.242) shows even less distinctive language used by the model: The most important word is “zwischenzeitlich” (meanwhile). This absence of more descriptive language used explains the poor performance of this class, caused by a big number of misclassifications landing on the label “SCHADENSCHREIBEN” (see also the analysis in chapter Linear SVM versus 2-layered CNN5.4).
- “SCHADENGUTACHTEN” on the other hand, a frequently occurring class with an even higher weight yields a strong 95.6% in weighted F1. The most important feature is the page count information. As already indicated by the distribution of word count by document classes (see Figure 3-4) the model can yield strong results for this particular category by leveraging the available document length information.

Focusing on the classes with particular strong prediction results

Linear SVM Classification - Coefficients (abs*) for selected Classes					
BONITAETSENCScheidung [Wg. F1=.994]		AUSWEISKOPIE [Wg. F1=.994]		LEASINGANTRAG [Wg. F1=.977]	
Feature	Coeff.*	Feature	Coeff.*	Feature	Coeff.*
deckblatt	1.193	passport	2.391	typ	2.118
bonitätsprüfung	1.099	idd	2.087	darlehensnehmer	1.852
anfrage	.967	cm	1.730	leasingantrag	1.817
welche	.931	BATCHCONTENT_PDF Image Only	1.680	leasingnehmer	1.523
BATCHKLASSE_BESTAND	.893	BATCHCONTENT_DS PDF Image Only	1.670	kaution	1.257
ihre	.803	2007	1.660	geschäftsbedingungen	1.170
lesbare	.767	5	1.586	gewerbliches	1.160
rahmen	.751	teil	1.457	vorhanden	1.144
vertragsabschluss	.742	fz	1.438	km	1.127
ausgestellt	.734	BATCHCONTENT_DS PDF Image + Te:	1.366	antragsteller	1.025

Table 5-6 provides insight on three top categories and their coefficients:

Linear SVM Classification - Coefficients (abs*) for selected Classes					
BONITAETSENCScheidung [Wg. F1=.994]		AUSWEISKOPIE [Wg. F1=.994]		LEASINGANTRAG [Wg. F1=.977]	
Feature	Coeff.*	Feature	Coeff.*	Feature	Coeff.*
deckblatt	1.193	passport	2.391	typ	2.118
bonitätsprüfung	1.099	idd	2.087	darlehensnehmer	1.852
anfrage	.967	cm	1.730	leasingantrag	1.817
welche	.931	BATCHCONTENT_PDF Image Only	1.680	leasingnehmer	1.523
BATCHKLASSE_BESTAND	.893	BATCHCONTENT_DS PDF Image Only	1.670	kaution	1.257
ihre	.803	2007	1.660	geschäftsbedingungen	1.170
lesbare	.767	5	1.586	gewerbliches	1.160
rahmen	.751	teil	1.457	vorhanden	1.144
vertragsabschluss	.742	fz	1.438	km	1.127
ausgestellt	.734	BATCHCONTENT_DS PDF Image + Te:	1.366	antragsteller	1.025

Table 5-6: Linear SVM – Feature Importance for selected Classes  
(\*Coefficients shown with their absolute value)

The top ranked features for these classes show distinctive language with regards to the concepts they refer to. “BONITAETSENCScheidung” and “LEASINGANTRAG” feature expressions like “bonitätsprüfung”, “darlehensnehmer” and “leasingantrag”. The expressions “passport”, “idd” and “cm” are strong drivers for classifying documents labeled “AUSWEISKOPIE”. German passports and ID-cards frequently carry the character sequence “idd” in the machine readable zone of the passport and the sequence “cm” is referring to a person’s height registered in the passport. Additionally the classifier makes use of meta data by leveraging the BATCHCONTENT feature describing the document to be of “Image only” type.

Summarizing the discussion of the linear models, they generally deliver strong results against the set baseline. The Linear SVM produces the best results in this flight of experiments, slightly ahead of the Logistic Regression Classifier. The results suggest that the noise contained in the data (see discussion in chapter 3.3.3) does not hinder the model to learn meaningful patterns for most of the classes and in particular for the classes carrying a higher weight measured by their page count volume.

## 5.2 Deep Learning Models leveraging Embeddings

The second group of experiments focusses on different Deep Learning architectures applied to the problem at hand. All experiments are run in the framework of the GTA (see Figure 4-1), with a specific Deep Learning technique (i.e. a BiLSTM) integrated as a Text Model layer. This chapter will summarize the top line results of the different Deep Learning methods validated on the test data set  $N_{test}$ .

The first idea put to test is a BiLSTM architecture (for details see chapter 4.4.1). The BiLSTM is fed with plain (not pre-trained) embedding vectors for input. During the learning process the Embedding Layers’ weights are continuously optimized. Several

BiLSTM experiments are run with variations of batch size and learning rate.

Top Line Results - Deep Learning Experiments w. (not pre-trained) Embedding Layer		
Model Architecture	Key Parameters	Wght. F1
<i>Abbreviations: SQL Sequence Length, BS Batchsize, LR Learnrate, CW Classweights, EP Epochs</i>		
BiLSTM	SQL=512, BS=64; LR=.0001, CW=True, EP=70	.838
BiLSTM	SQL=512, BS=30; LR=.0001, CW=True, EP=70	.848
BiLSTM	SQL=512, BS=64; LR=.00001, CW=True, EP=120	.857
BiLSTM	SQL=512, BS=30; LR=.00001, CW=True, EP=108	<b>.883</b>

Table 5-7 lists the weighted F1 scores for those experiments.

Top Line Results - Deep Learning Experiments w. (not pre-trained) Embedding Layer		
Model Architecture	Key Parameters	Wght. F1
<i>Abbreviations: SQL Sequence Length, BS Batchsize, LR Learnrate, CW Classweights, EP Epochs</i>		
BiLSTM	SQL=512, BS=64; LR=.0001, CW=True, EP=70	.838
BiLSTM	SQL=512, BS=30; LR=.0001, CW=True, EP=70	.848
BiLSTM	SQL=512, BS=64; LR=.00001, CW=True, EP=120	.857
BiLSTM	SQL=512, BS=30; LR=.00001, CW=True, EP=108	<b>.883</b>

Table 5-7: BiLSTMs – Top Line Classification Results

The F1 scores improve with a more granular learning rate of  $1.0 \times 10^{-5}$  and a smaller batch size of 30. The best performing BiLSTM experiment yields a .883 weighted F1. This result ranks similar to the linear models but settles behind the performance of the Linear SVM (see above).

The next run of experiments covers different CNN architectures with variation in number of filters and kernel size.

Top Line Results - Deep Learning Experiments w. (not pre-trained) Embedding Layer		
Model Architecture	Key Parameters	Wght. F1
<i>Abbreviations: SQL Sequence Length, BS Batchsize, LR Learnrate, CW Classweights, EP Epochs</i>		
<i>CNN &amp; Max Pooling</i>		
CNN with 128 Filter/Kernel Size 5	SQL=512, BS=32; LR=.0001, CW=False, EP=120	.914
CNN with 300/5	SQL=512, BS=64; LR=.0001, CW=False, EP=120	.904
CNN with 300/7	SQL=512, BS=32; LR=.00001, CW=False, EP=120	.874
CNN with 128/5	SQL=512, BS=32; LR=.00001, CW=True, EP=120	.927
2-Layer CNN w. 300/5 & 300/3	SQL=512, BS=64; LR=.0001, CW=True, EP=120	<b>.94</b>

Table 5-8: CNNs – Top Line Classification Results

With these models training much faster than BiLSTMs several experiments are applied with a number of 120 epochs each to get a maximum read on their performance. The first experiment running with a setting of 128 filters and a kernel size of 5 already outperforms the baseline, the linear models and the BiLSTM experiments by large margins. Increasing the number of filters and kernel size (300/7) doesn't improve on this but weighing in the page volume (class weights) during training increases the performance to .927. Building on this substantially stronger results are achieved with an architecture that provides for 2 parallel CNN layers learning with two different kernel sizes (5 and 3) and 300 filters each. This setting yields .94 in F1, outperforming all other experiments by far.

Applying open sourced available embedding vectors, pre-trained on large German text corpuses might even improve on the CNN methods. Based on three available pre-trained sources specific embedding matrices are built up and used to initialize the Embedding Layer. Given the noisy data (see chapter 3.3.3) this creates a challenge with a high share of zero-vectors in the embedding matrices used (see chapter 4.4.3).

Top Line Results - Deep Learning Experiments w. pre-trained Embedding Layer		Wght.	F1
Model Architecture	Key Parameters		
<i>Abbreviations: SQL Sequence Length, BS Batchsize, LR Learnrate, CW Classweights, EP Epochs</i>			
<b>CNN &amp; Max Pooling with pretrained Embeddings</b>			
CNN with 300/5 & GloVe	SQL=512, BS=64; LR=.0001, CW=True, EP=120		.935
CNN with 300/7 & GloVe	SQL=512, BS=32; LR=.0001, CW=False, EP=120		.907
2-Layer CNN w. (300/5; 300/3) & Word2Vec	SQL=512, BS=64; LR=.0001, CW=True, EP=120		.918
2-Layer CNN w. (300/5; 300/3) & Spacy LM	SQL=512, BS=64; LR=.0001, CW=True, EP=120		.919
2-Layer CNN w. (300/5; 300/3) & GloVe	SQL=512, BS=64; LR=.0001, CW=True, EP=120		.937

Table 5-9: CNNs with pre-trained Embeddings - Top Line Classification Results

The first two experiments are picking up on the CNN layer with 300 filters and kernel size 5 or 7 respectively. Applying the deepset.AI GloVe embeddings can improve the F1 scores compared to the same models with plain embeddings substantially. But likewise to the experiments reported in Table 5-8 simply increasing the number of filters doesn't buy more mileage in F1.

The pattern of the pre-trained embedding vectors being superior changes by comparing results of the 2-layered CNNs with and without pre-trained embeddings. While the deepset.AI GloVe vectors achieve the highest F1 in this group of experiments with pre-trained embeddings they still fall short of the performance reported on the 2-layered CNN experiment with plain (vanilla) embedding vectors (compare with Table 5-8).

Summarizing this flight of experiments, with enough tweaking of settings and computation time the application of BiLSTMs can be leveled to results slightly behind the linear models. CNNs however surpass the linear models (with one exception) in all experiments run. Applying pre-trained vectors doesn't guarantee better results. The promising idea of leveraging pre-trained vectors cannot proof its superiority in the noisy data environment of this document classification task. The best performing Deep Learning experiment is a 2-layered CNN trained with simple plain embedding vectors.

### 5.3 Transformer Model: BERT

The last flight of experiments covers the BERT Base Multilingual model with self-attention mechanics. The experiments employ the cased and uncased version.

Top Line Results - Deep Learning Experiments - BERT Base Multilingual		
Model Architecture	Key Parameters	Wght. F1
<i>Abbreviations: SQL Sequence Length, BS Batchsize, LR Learnrate, CW Classweights, EP Epochs</i>		
BERT Base Multilingual Cased	SQL=384, BS=32; LR=.0001, CW=True, EP=26	.859
BERT Base Multilingual Cased	SQL=384, BS=32; LR=.0001, CW=False, EP=25	.825
BERT Base Multilingual Uncased	SQL=384, BS=32; LR=.0001, CW=True, EP=28	.846
BERT Base Multilingual Uncased	SQL=384, BS=32; LR=.0001, CW=False, EP=20	.812

Table 5-10: BERT Multilingual Cased & Uncased - Top Line Classification Results

The cased version of the applied BERT model performs significantly stronger but still settles in a lower range of scores when compared to the linear models and the CNNs (see chapter 5.1.1 and 5.2). Table 5-11 shows the Top 15 and Low 15 scoring classes for the cased BERT model (with an overall F1 of .859).

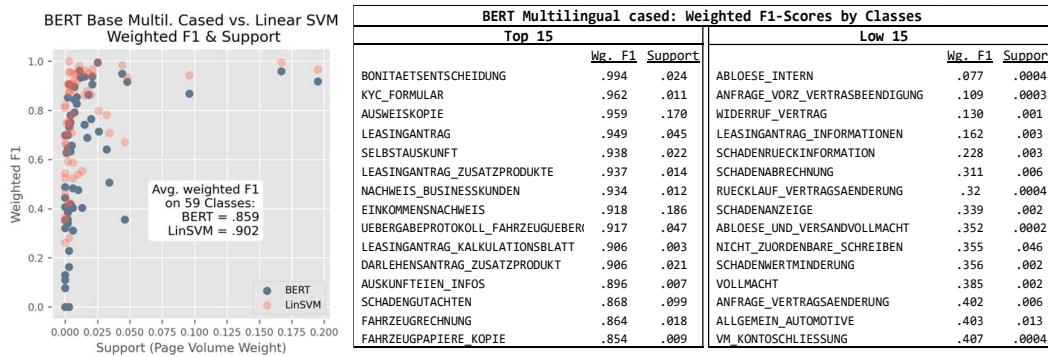


Table 5-11: BERT Multilingual Cased - F1 & Support by Document Type

The scatterplot in the left panel of Table 5-11 compares the F1 scores for each class between the better performing Linear SVM (red glyphs) and the BERT cased model (blue glyphs). The BERT model cannot outperform the Linear SVM in any of the 59 different categories. It lags behind on the more important (high support) categories as well as on the classes with less impact and cannot keep up with the Linear SVM particularly in the range of weighted F1 score above 50%.

The potential benefit of a large pre-trained model, the big amount of parameters (177 Mio.) and the self-attention transformer architecture described in chapter 2.4 don't show their often attributed superiority in this particular document classification problem: The BERT Multilingual cased model scores higher than the uncased version but remains only moderately above the set baseline performance. This might be due to the memory constraints mentioned above limiting the sequence lengths of the BERT experiments to  $\frac{3}{4}$  of the inputs used for the Deep Learning experiments. The influence of the noisy data might also prevent the BERT architecture to leverage its strength in comprehending semantics throughout a text sequence by applying self-attention logic.

## 5.4 Linear SVM versus 2-layered CNN

Chapter 5.1, 5.2 and 5.3 discussed the results of three groups of experiments. With the BERT models significantly lagging behind the models of the two other groups this chapter will focus on benchmarking the two superior performing models to identify individual strengths and weaknesses for further leverage:

The Linear SVM (LinSVM) is benchmarked against the 2-layered CNN (2L-CNN). The 2L-CNN yields an F1 of 94% with a 3.8 percentage points advantage over the LinSVM (90.2%).

Figure 5-2 illustrates a breakdown of this deviation by each one of the 59 classes: The x-axis depicts every class with its deviation in weighted F1 between the 2L-CNN and the LinSVM. Positive x-values describe classes with superiority of the 2L-CNN and negative x-values vice versa. The support for each class is indexed on the y-axis.

The LinSVM scores higher on 20 classes (red glyphs in Figure 5-2) but the 2L-CNN outnumbers this with 39 labels (blue glyphs) upon which it's scoring higher than the LinSVM. Not only does the 2L-CNN score higher on  $\frac{2}{3}$  of the classes, many of those classes also carry a higher page count weight thus scaling the weighted F1 higher than classes with a lower page count weight.

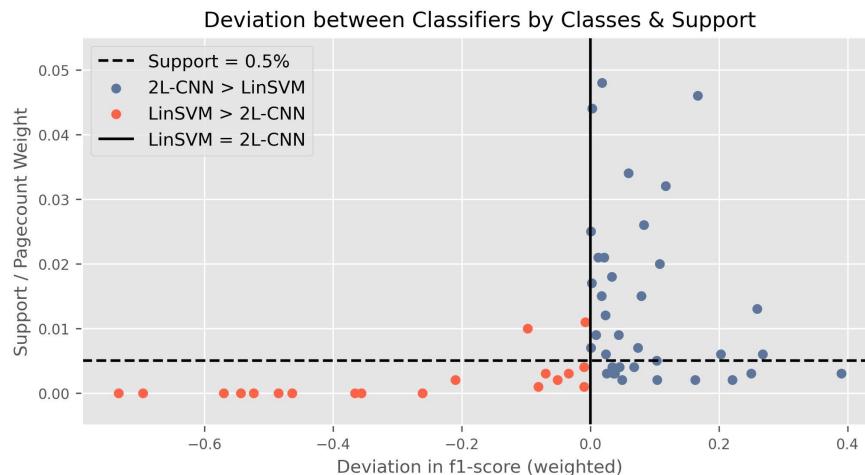


Figure 5-2: Comparison 2-layered CNN and Linear SVM:  
Deviation in Classification Performance - Breakdown by Classes & their Support

A dashed line in Figure 5-2 marks a page count weight of .005. Above this threshold only 3 classes<sup>35</sup> are found where the LinSVM is superior to the 2L-CNN. But 29 classes

---

<sup>35</sup> Note that 1 outlier for the LinSVM scoring higher than the 2L-CNN is excluded in the graph.

above the threshold account for labels with the 2L-CNN outperforming the LinSVM. This observation suggests a tendency for the 2L-CNN to score better than the LinSVM in classes where documents typically contain more pages (and thus more words). Table 5-12 shows the Pearson's Correlation Coefficient between the 59 different F1 scores (as illustrated above) with the lengths of the documents of the respective classes (measured in pages or word count) for the two models in focus:

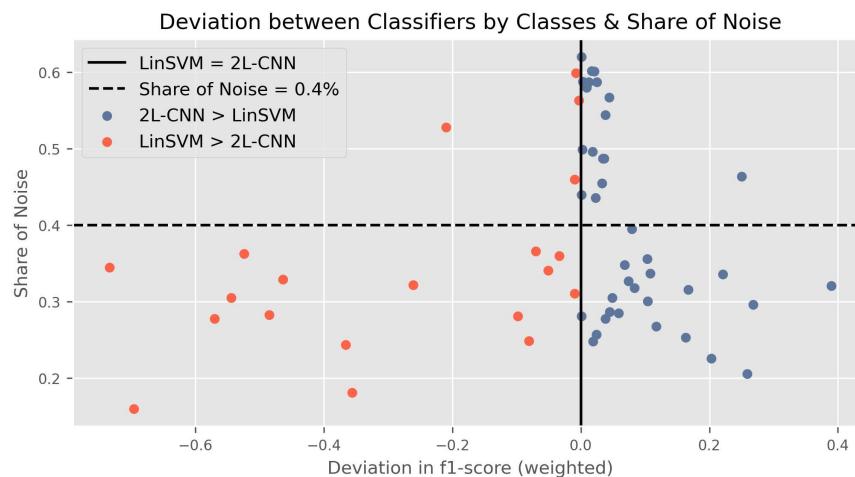
Correlation of Classifier Performance & Length of Documents		
R <sup>2</sup> (Pearson)	Wght. F1 & Page Count	Wght. F1 & Word Count
Linear SVM	.326	.377
2-Layered CNN	.309	.372

Table 5-12: Correlation of Classifier Performance by Class & Document Length

The R<sup>2</sup> scores indeed prove a positive correlation between the document length and the classifier performance in general. But the two classifiers don't show much different R<sup>2</sup>-scores in between themselves regardless if measured by pages or word counts. Thus the typical document lengths of different classes cannot explain the advantageous behavior of the 2L-CNN classifier in  $\frac{2}{3}$  of the 59 classes.

With document length not contributing to the explanation, the varying degree of noise amongst the different classes moves into focus:

Is there a relationship between the superiority of the 2L-CNN and the noisiness of respective categories? Chapter 3.3.3 established a proxy to operationalize the amount of noise contained in the documents for every class. Figure 5-4 is a slight variation of Figure 5-3 with the y-axis depicting this proxy "Share of Noise" for each class.

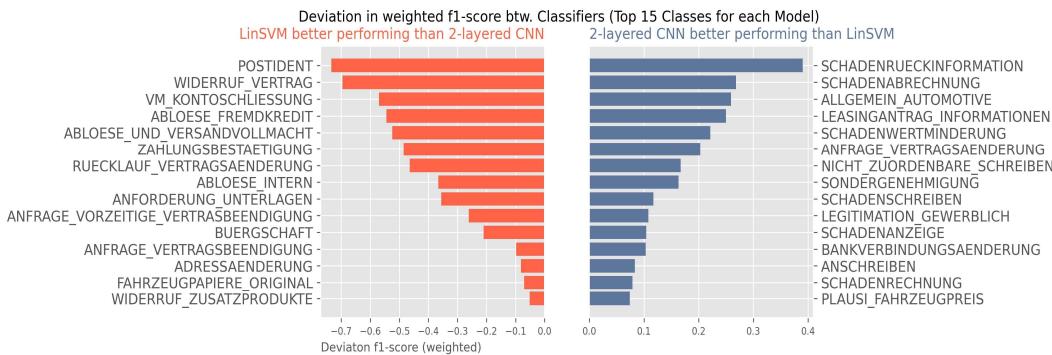


**Figure 5-3: Comparison 2-layered CNN and Linear SVM:  
Deviation in Classification Performance  
Breakdown by Classes & their Share of Noise**

The data suggests that the 2L-CNN model does better on noisier data than the LinSVM: Above a threshold of 40% share of noise there are 17 classes where the 2L-CNN scores better than the LinSVM (only 4 categories).

But most of the categories contributing to the overall superiority of the 2L-CNN are within a share of noise range between 20% and 40%. In this range there are equally as many categories with the LinSVM being superior over 2L-CNN as categories where the 2L-CNN performs better.

In summary the 2L-CNN shows more often better results versus the LinSVM with noisier data (above the threshold). But this advantage scales low. With less noisy data both classifiers can show superiority on different classes. In some cases this superiority scales by large margins. Figure 5-4 illustrates this effect for the respective Top 15 classes where one classifier outperforms the other.



**Figure 5-4: Comparison 2-layered CNN and Linear SVM:  
Top 15 Classes with Superiority vs. one another**

Looking at the 2L-CNNs strengths in the right panel two patterns emerge:

- The 2LCNN performs better on topics related to damage management (labels containing "SCHADEN").
- And it can classify more reliable on generic classes with heterogeneous language and documents like "ALLGEMEIN\_AUTOMOTIVE", "ANSCHREIBEN", "NICHT\_ZUORDENBARE\_SCHREIBEN".

Exemplarily Table 5-13 shows a breakdown on the errors that both models produce on two selected classes to illustrate those patterns mentioned above:

Misclassifications for selected Classes					
SCHADENRUECKINFORMATION n <sub>test</sub> = 129 (1.0) - Accuracy*		ALLGEMEIN_AUTOMOTIVE n <sub>test</sub> = 344 (1.0) - Accuracy*			
Class	LinSVM	2LCNN	Class	LinSVM	2LCNN
SCHADENRUECKINFORMATION	.209	.612	ALLGEMEIN_AUTOMOTIVE	0.616	0.770
SCHADENSCHREIBEN	.674	.341	NICHT_ZUORDENBARE_SCHREIBEN	0.081	0.049
SCHADENABRECHNUNG	.070	.023	ANFRAGE_VERTRAGSAENDERUNG	0.070	0.026
SCHADENWERTMINDERUNG	.008	.008	OTHER	0.064	0.044
ALLGEMEIN_AUTOMOTIVE	.008	--	ANFORDERUNG_FAHRZEUGPAPIERE	0.044	0.009
ANSCHREIBEN	.008	--	ANSCHREIBEN	0.032	0.006
SCHADENGUTACHTEN	.008	--	ANFRAGE_VERTRAGSBEENDIGUNG	0.029	0.084
BANKVERBINDUNGSÄNDERUNG	.008	--	WIDERRUF_ZUSATZPRODUKTE	0.017	--
NICHT_ZUORDENBARE_SCHREIBEN	.008	.008	UMSCHLAG	0.012	0.006
SCHADENRECHNUNG	--	.008	RUECKLAUF_VERTRAGSAENDERUNG	0.009	--
			ANFORDERUNG_UNTERLAGEN	0.009	--
			ADRESSENÄNDERUNG	0.006	--
			EINKOMMENSNACHWEIS	0.003	--
			VOLLMACHT	0.003	--
			ZAHLUNGSBESTÄTIGUNG	0.003	--
			RUECKNAHMEPROTOKOLL	0.003	--
			UEBERGABEPROTOKOLL_FAHRZEUGUEBI	--	0.003
			AUSKUNFTEN_INFOS	--	0.003

Table 5-13: True Positives & False Positives for selected Classes<sup>36</sup>

The left panel illustrates the mistakes made by the LinSVM and the 2LCNN on the category "SCHADENRUECKINFORMATION". Both models make mistakes with their prediction landing on another "SCHADEN"-related label. But the 2LCNN distinguishes much better between the false label "SCHADENSCHREIBEN" and the true label than the LinSVM.

The right panel emphasizes the advantage of the 2LCNN on the generic class "ALLGEMEIN\_AUTOMOTIVE". A category collecting all sorts of different documents related broadly to automotive intents that don't fit a dedicated specific category. While the LinSVM chooses from a range of 15 different misclassified labels for the prediction the 2L-CNN keeps the variance at bay choosing from only 9 different labels with a significantly higher overall accuracy than the LinSVM.

## 5.5 Synthesis – Combining individual Strengths

The Deep Learning approach and the BOW/TF-IDF approach show different strengths and weaknesses when applied to the documents in this use case. Both models can learn different aspects of the fundamental data. Consequently those individual strengths of each approach can be synthesized into an ensemble classifier, combining the best of both worlds.

---

<sup>36</sup> Please note that performance in this table is reported as accuracy score, calculated from the confusion matrix. There is no weighting scheme applied for this score. Because of this these numbers are slightly lower than the weighted F1 scores that are reported throughout this chapter to measure classification performance.

Instead of a simple voting scheme this estimator could use probability distributions that classifiers in the ensemble team produce. For each instance the 2LCNN outputs an array of probabilities over the 59 different classes. Because the LinSVM not generating a probability distribution like the 2LCNN, it is exchanged in this final experiment with the Logistic Regression Classifier (LogReg) trained before. It's a linear model, but can produce a probability distribution over the prediction of classes. The best experiment with Logistic Regression yields .889 F1 (see Table 4-2), close enough to stand in for the LinSVM (.902) as a proxy for a good linear model leveraging BoW/TF-IDF.

The Ensemble estimator compares the predicted probability distributions of classes for each document and chooses the prediction with the highest confidence. Applied to  $N_{test}$  the ensemble estimator yields the results reported in Table 5-14.

Across all 59 classes applying the Ensemble estimator yields a .948 F1, surpassing the so far best performing single classifier, the 2LCNN with an F1 of .94.

Classification Performance: Linear Models vs. 2LCNN vs. Ensemble				
	Weighted f1-score on all Classes & selected Classes			
Class	LinSVM	LogReg	2LCNN	Ensemble
<b>TOTAL (59 Classes)</b>	<b>.902</b>	<b>.889</b>	<b>.94</b>	<b>.948</b>
POSTIDENT	.821	.805	.087	.773
WIDERRUF_VERTRAG	.696	.377	.001	.342
VM_KONTOSCHLIESSUNG	.667	.583	.097	.541
SCHADENRUECKINFORMATION	.281	.276	.671	.662
SCHADENABRECHNUNG	.588	.487	.856	.843
ALLGMEIN_AUTOMOTIVE	.554	.493	.813	.808

Table 5-14: Overview Classifier Performance with Ensemble Classifier

The Ensemble builds on the strengths of the linear approach combined with those of the 2LCNN:

- The first group of classes reported underneath the total results in Table 5-14 is a selection of categories where the LinSVM and similarly the LogReg handles the classification better than the 2LCNN. The Ensemble classifier can leverage the strength of the LogReg classifier and improve F1 scores in comparison to the 2LCNN significantly.
- The second group of classes shows the opposite effect: The Ensemble estimator builds on the predictions produced by the 2LCNN and scores higher than the respective linear models.

## 5.6 Limitations & possible Improvements

During this project several limitations on the experiments are observed:

**Managing noise:** The heterogeneity of technical formats of the input data (scans, emails, forms, pictures, etc.) introduces a lot of noise to the transformed texts used for training the classifiers and in this project only limited effort is applied to de-noise the text. The quality of the input data thus could be improved with more thorough data cleaning to filter and transform noise. Further steps could be the development of more regex-based preprocessing routines or the application of language normalization techniques like applying stemming, lemmatization and stop-word-filtering.

**Anonymization:** The very sensitive nature of the data forces strict data protection and information security measures to be employed to the raw data (see chapter 3.2). A key element of the anonymization procedure is the application of a pre-trained SpaCy German Language model to identify and exchange names of people, organizations and locations with a generic label. The SpaCy authors report an F1 of .842 on the NER-task<sup>37</sup>. While chosen as a better method than just anonymizing by regex or lookup rules this method still is prone to a certain degree of error. Thus adding additional noise and introducing wrong labels at times. This effect can be mitigated with a subject-related, improved pre-training of the language model used for this particular NER-task.

**Feature selection:** All classifiers are exposed to the preprocessed and vectorized textual input plus the categorical and numerical features provided within the meta data of the documents. The classification performance could be potentially improved by the definition of additional features. That could be general linguistic features like Part-of-Speech-Tags, syntactical dependencies or specifically fine-tuned features (e.g. specific NER features) to improve particularly weak performing classes.

**Hyper parameters:** Within the first group of experiments numerous algorithms are applied using the Scikit-learn API. While different parametrizations are tested on the selected classifiers there still is room for improvement with a more thorough search for the best hyper parameter setting. Grid-search or random-grid search in conjunction with cross-validation could potentially yield the best parametrization for each model tested.

**Deep Learning architecture:** The GTA applied to all models (Figure 4-1) serves as a standard framework to apply different Deep Learning ideas like LSTMs and CNNs to compare their performance. But deep neural nets allow for very complex architectures with many strategic choices to be made: The format of the input (sequence length, batch size), the number of layers and neurons, additional skip-connections, the optimizers chosen, the regularization techniques applied, etc.. In this universe of parameters and architectural decisions particular ideas could be improved with a more fine-tuned design and choice of the key architectural elements. The 2-layered CNN for example, while best performing as is amongst the Deep Learning methods applied, invites for

---

<sup>37</sup> See documentation: [https://github.com/explosion/spacy-models/releases//tag/de\\_core\\_news\\_md-3.0.0](https://github.com/explosion/spacy-models/releases//tag/de_core_news_md-3.0.0)

experiments with even more parallel or subsequent filter/kernel combinations. Those experiments could leverage on the CNN's capability of learning different relevant patterns within the text in parallel.

**Pre-trained embeddings:** One flight of the Deep Learning experiments is using open sourced German pre-trained word embeddings from deepset.AI and Explosion (SpaCy). Both companies report that their embeddings are trained predominantly on German Wikipedia<sup>38</sup>. As shown in chapter 3.3.3 and manifested in the results (chapter 5.2) these pre-trained embedding vectors have only limited overlap with the vocabulary build from the training data. The resulting embedding matrices contain a high share of zero-vectors. Less noisy data (see above) and more subject-matter specific pre-trained embeddings could potentially yield much better embedding matrices and consequently F1 scores than reported.

**Transformer models:** Within the last three years “*The Transformer (Vaswani et al., 2017) has rapidly become the dominant architecture for natural language processing, surpassing alternative neural models such as convolutional and recurrent neural networks in performance [...] The Transformer architecture is particularly conducive to pretraining on large text corpora, leading to major gains in accuracy on downstream tasks including text classification ...*” (Wolf, et al., 2020, p. 38). The widely reported superiority of this architecture cannot be demonstrated in the experiments herein. A major handicap is the experienced memory limitation on the DLP, limiting the potential capacity of this model. In addition the applied BERT Multilingual transformer model open sourced by Google is trained on 104 different languages. Building on the success of the BERT idea, specific German BERT implementations and variations thereof are available meanwhile. With dedicated pre-training on German these model could potentially yield higher scores, provided they can overcome the noise level contained in the data and run without memory limitations as experienced.

**Ensemble:** Chapter 5.5 describes the synthesis of two models to an ensemble estimator based on the property that both of them generate probability distributions. This idea of training specialized estimators and combining them can be extended to integrate a range of even more diverse classifiers and fine-tune (or train) a voting scheme amongst them.

**Specialization:** Many classes are exclusive to certain business units. Hence there is a strong relationship between the feature “BATCHKLASSE” and the probability for certain classes. This insight could provide for more specialized classifiers trained on individual batch classes. The application of the individual model is triggered by the meta

---

<sup>38</sup> SpaCy reports to be additionally trained on the TIGER Corpus (see documentation: <https://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/tiger/>).

information of the data. The general principal of specialized models can also be implemented by hierarchical classifiers that determine higher levels of logic (beyond the document type) before they defer to a final range of models predicting the document type.

## 6 Conclusions

This project surveys different approaches and models to solve a document classification task in a sparse and noisy data environment. Document classification is a widely researched field within supervised learning with a high relevance in practical business applications. Hence many approaches and ideas for solving this task exist.

The presented text classification task is characterized by a sparse and noisy data environment: Documents arrive in a big variety of technical formats and lengths with external OCR and other preprocessing employed beforehand, that add significant levels of noise to the data. To operationalize the contained “Share of Noise” a proxy has been established. Common public sources of German language are benchmarked against the training corpus. With additional compliance requirements the documents undergo extensive anonymization adding further dilution to the semantics of the texts.

This project experimentally applied three different general routes to approach a text classification problem. Building on different text vectorization schemes different methods are employed to search for the best suited approach given the specific use case.

The experiments yield classification results in a range from .737 to .948 on weighted F1 across a total of 59 classes of document types.

The more traditional approach to transform textual input into a BoW representation with additional TF-IDF scaling can achieve substantial results if linear models like Logistic Regression Classifiers or Linear Support Vector Machines are applied. The latter proof to achieve the highest score (.902) in this group of experiments, standing the test and underlining that the traditional approach can deliver substantial results despite a high share of noise in the given textual data. For some classes with highly descriptive expressions the linear models achieve results above .99. More generic classes with big heterogeneity in language and style are more difficult for these models. In these cases their performance falls victim to the simplicity of the BOW/TF-IDF vectorization, which doesn't provide contextual information and doesn't allow for separate different patterns to be learned. The same holds for categories that are semantically specific but do share big thematic overlap with some other related categories.

Deep Learning methods have fueled many advances and break-through results within the NLP community in recent years. The second group of experiments puts different ideas of Deep Learning methods to test. They all share the principle to use word embeddings for vectorized input and leverage the theoretical advantage of capturing contextual information and word order within a sequence of text. With sufficient fine-tuning BiLSTMs can produce results close to the linear models but despite the

expensive computation with much longer training time they do not surpass the simpler approach of the linear models. The most promising contender in the second group of experiments is the concept of Convolutional Neural Nets. A 2-layered CNN yields the best result (.94) across all three groups of experiments. Their capability of applying different windows (kernel) and learning hundreds of filters for a set kernel size in parallel shows its strength especially in the area of the shortcomings described for the linear models above: The 2-layered CNN achieves better results especially in generic but heterogeneous classes and in classes with semantically related content that are hard to distinguish in between. With that it surpasses all other approaches tested and shows superiority in the given sparse and noisy data environment.

The third group of experiments covers the application of the Transformer architecture. BERT, a widely recognized implementation of this idea is put to the test. But the experiments cannot show their theoretical strength and superiority. BERT falls short in performance despite a range of more than 170 million parameters fitted and the advantage of a massively pre-trained embedding space. The advantage of identifying relevant parts of language distributed across several parts of a sequence by applying the self-attention mechanism does not play out in the given sparse and noisy data environment.

Additionally it can be shown that the traditional approach with a BoW/TF-IDF vectorization and the application of CNNs leveraging plain embeddings allow for different patterns in the data to be recognized by each approach respectively. Consequently the strengths of these two approaches can be forged into one ensemble estimator exploiting the strengths of the approaches. A combination of both approaches yields the best of both worlds and exercises maximum superiority versus a single estimator approach.

The BMW Bank expects a deployment recommendation and a minimum viable product prototype within the defined scope of this project:

Given the complexity and budget implications of training Deep Learning methods and maintaining a real life permanent production environment the recommendation goes towards the simpler approach of a BoW/TF-IDF vectorization combined with a Linear Support Vector Machine or a Logistic Regression Classifier<sup>39</sup> as a first step implementation. Building on this further ideas with batch class specific classifiers and ensemble estimators should be developed, tested and finally deployed to replace the plain linear models.

---

<sup>39</sup> Logistic Regression Classifiers can produce a probability distribution for each prediction. This could be an important property for the subsequent processing of the inference result within the DMS.

At time of writing the trained Linear SVM and Logistic Regression Classifier are actively deployed as a proof of concept into a secured BMW Bank cloud environment and are fully functionally serving as a REST-API web service.

## Bibliography

- Aggarwal, C. C. & Zhai, C., 2012. A Survey of Text Classification Algorithms. In: *Mining Text Data*. s.l.:s.n., pp. 163 - 222.
- Anon., 2021. *Scikit-Learn Documentation*. [Online] Available at: <https://scikit-learn.org/stable/modules/sgd.html#sgd> [Accessed 11 01 2021].
- Bahdanau, D. C. K. & Yoshua, B., 2016. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv*.
- Bai, S., Kolter, J. Z. & Koltun, V., 2018. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *CoRR*, Volume abs/1803.01271.
- Bekkerman, R., 2004. "Automatic Categorization of Email into Folders: Benchmark Experiments on Enron and SRI Corpora". *Computer Science Department Faculty Publication Series*, p. 218.
- Bishop, M. C., 2006. *Pattern Recognition and Machine Learning*. s.l.:Springer Science & Business Media, LLC..
- Breiman, L., 1997. Arcing the Edge. *Technical Report 486*.
- Chaubard, F. & Socher, R., 2019. CS224n: Lecture Notes: Part VIII: Convolutional Neural Networks. *CS224n: Natural Language Processing with Deep Learning*.
- Cortes, C. & Vapnik, V., 1995. Support-Vector Networks. *Machine Learning*, Issue 20, pp. 273-297.
- Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K., 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*, Volume abs/1810.04805.
- Eisenstein, J., 2018. *Natural Language Processing*. s.l.:s.n.
- Firth, J. R., 1957. "A synopsis of linguistic theory 1930-1955.". In *Special Volume of the Philological Society*.
- Géron, A., 2019. *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow*. 2nd ed. s.l.:O'Reilly Media, Inc..
- Glorot, X. & Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 249-256.
- Goldberg, Y., 2015. A Primer on Neural Network Models for Natural Language Processing. *arxiv*.
- Goodfellow, I., 2015. *Gradient Descent and the Structure of Neural Networks Cost Functions*. [Online]

Available at: [https://www.deeplearningbook.org/slides/sgd\\_and\\_cost\\_structure.pdf](https://www.deeplearningbook.org/slides/sgd_and_cost_structure.pdf)  
[Accessed 26th 12 2020].

Goodfellow, I., Bengio, Y. & Courville, A., 2016. *Deep Learning*. s.l.:MIT Press.

Hastie, T., Tibshirani, R. & Friedman, J., 2008. *The Elements of Statistical Learning : Data Mining, Inference, and Prediction* New York: Springer.. 2nd ed. New York: Springer.

He, K., Zhang, X., Ren, S. & Sun, J., 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv:1502.01852 [cs.CV]*.

Hinton, G. E. et al., 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*.

Hinton, G., Srivastava, N., Swersky & Kevin, 2012. Neural Networks for Machine Learning: Lecture 6.

Hochreiter, S. & Schmidhuber, J., 1997. Long Short-term Memory. *Neural computation*, 12, Volume 9, pp. 1735-1780.

Ioffe, S. & Szegedy, C., 2015. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. [Online]  
Available at: <https://arxiv.org/pdf/1502.03167.pdf>  
[Accessed 27.12.2020 12 2020].

Joachims, T., 1998. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In: *ECML*. s.l.:s.n.

Joachims, T., 2001. A Statistical Learning Model of Text Classification for Support Vector Machines. In: *SIGIR '01*. s.l.:s.n.

Jurafsky, D. & Martin, J. H., 2019. *Speech and Language Processing*. 3rd Edition DRAFT ed. s.l.:s.n.

Kalchbrenner, N. G. E. B. P., 2014. A Convolutional Neural Network for Modelling Sentences. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 655-665.

Karpathy, A., 2015. *The Unreasonable Effectiveness of Recurrent Neural Networks*. [Online]  
Available at: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>  
[Accessed 29 12 2020].

Kowsare, K. et al., 2019. Text Classification Algorithms: A Survey. *Information (Switzerland)*, 23 April.

LeCun, Y., Bottou, L. & Orr, B. G. M. K.-R., 1998. Efficient BackProp. In: *Neural Networks tricks of the trade*. s.l.:Springer.

Mikolov, T., Chen, K., Corrado, G. & Dean, J., 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781*.

Murphy, K. P., 2012. *Machine Learning: A Probabilistic Perspective (Adaptive Computation and Machine Learning series)*. Cambridge, Massachusetts: MIT Press.

Ng, A., 2018. *Machine Learning Yearning*. s.l.:s.n.

Nicholson, C., 2020. *A Beginner's Guide to Attention Mechanisms and Memory Networks*.

[Online]

Available at: <https://wiki.pathmind.com/attention-mechanism-memory-network>

[Accessed 13th February 2021].

Olah, C., 2015. *Understanding LSTM Networks*. [Online]

Available at: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

[Accessed 29 12 2020].

Pennington, J., Richard, S. & Manning, C., 2014. GloVe: Global Vectors for Word Representation. pp. 1532-1543.

Raaijmakers, S., 2021est.. *Deep Learning for Natural Language Processing*. s.l.:Manning Early Access Programm (MEAP).

Salton, G. & Buckley, C., 1987. Term Weighting Approaches in Automatic Text Retrieval. *Information Process Management*, November, Issue 24, pp. 513-523.

Sparck Jones, K., 1972. "A statistical Interpretation of Term Specificity and its Application in Retrieval". *Journal of Documentation*, Volume Vol. 28 No. 1, pp. 11-21.

Srivastava, N. et al., 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, Issue 56, pp. 1929-1958.

Tokunaga, T. & Makoto, I., 1994. Text categorization based on weighted inverse document frequency. *Information Processing Japan, SIGNL 94*, p. 33–40..

Vapnik, V. & Chervonenkis, A., 1964. A class of algorithms for pattern recognition learning. „*Avtomat. Telemekh*, Issue 25, pp. 937-945.

Vaswani, V. et al., 2017. Attention Is All You Need. *CoRR*, Issue abs/1706.03762.

Wolf, T. et al., 2020. Transformers: State-of-the-Art Natural Language Processing. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. s.l.:Association for Computational Linguistics, pp. 38-45.