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)

14.03.2021

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, 14.03.2021

Andreas Barth

Table of Content

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
BoW	Bag of Words
CBOW	Continuous Bag-of-Words
CNN	Convolutional Neural Net
DLP	Deep Learning Platform
DNN	Deep Neural Net
EDA	Explorative Data Analysis
GloVe	Global Vectors
GPU	Graphical Processing Unit
GTA	General Training Architectures
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
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
SVM	Support Vector Machine
TF	Term Frequency
TF-IDF	Term Frequency Inverse Document Frequency

List of Equations

List of Figures

List of Tables

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.[1] Additional insurance and banking products complement modern mobility solutions for the customers. BMW Bank is an entity within BMW Group Financial Services that serves a number of regional markets, including Germany, Italy, Spain and Portugal.

BMW Bank implemented a new DMS in 2020 to support their 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 DMS can subsequently extract relevant information from the document for further processing (i.e. contract number). Thus a correct and - ideally - fully automatic classification of a document is a vital precondition 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

[1] For more information on BMW Group Financial Services please refer to https://www.bmwgroup.com/en/brands-and-services/financial-services.html

takes over manual inspection and classification. Figure 1-1 schematically summarizes this process.

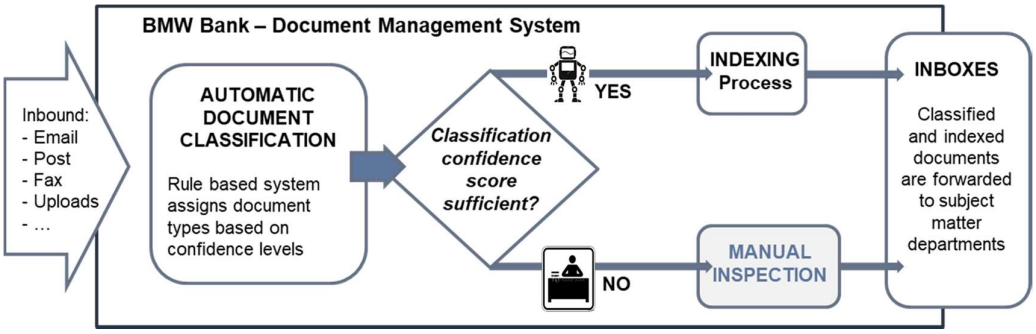


Figure 1-1: BMW Bank DMS

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 documents 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 Optical Character Recognition (OCR) scans of 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 consists 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, marketing 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, thus adding 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 available general (German) language solutions to this very domain specific classification task.

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

1.3 Problem definition

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 asks for 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. But faced with the challenges mentioned above the question arises if those latest state of the art recipes automatically provide better results when applied to the task at hand. Especially the limitation on volume of available training data asks for 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).

This project will survey and apply three different approaches to identify the best approach, thus achieving maximum accuracy for the given multi-class document classification problem in the sparse and noisy data environment provided by the BMW Bank's use case.

1.4 Outline

With the description of this use case set the thesis is structured as summarized below:

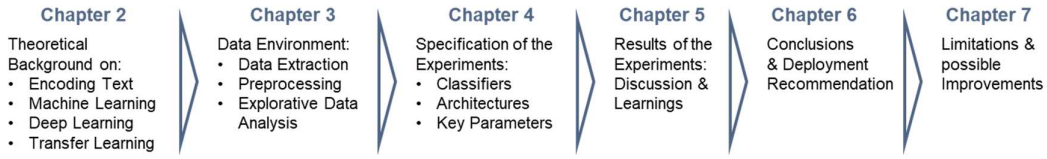


Figure 1-2: Structure of the Thesis Report

Chapter 2 is introducing key concepts in Machine Learning, Deep Learning and Transfer Learning, providing theoretical background for the experiments applied later on. Particular focus is on the process of encoding textual data.

Chapter 3 describes the available data, its extraction process 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 different approaches and developing a “Best of All” approach leveraging the identified individual strengths and weaknesses of the different routes taken.

Chapter 6 summarizes this report and provides a deployment recommendation that is implemented as a minimum viable product (MVP) for BMW Bank.

Chapter 7 names a number of limitations faced during the experiments conducted and ideates on further possible improvements to the approaches taken.

2 Theoretical Background

This chapter is a brief introduction to 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

For this project models stemming from these three groups are applied and benchmarked against each other. Selected representatives of these groups are now 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) assumption does exactly this (Jurafsky & Martin, 2019, p. 58). 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[2] contained in V. For a classification task with thousands of documents and a vocabulary size $|V|$ in the tens of thousands the multi-dimensional vector space can grow large. To observe computational cost $|V|$ commonly gets reduced to the most frequent words, thus the

[2] 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 *x* words) resembles the feature space that Machine Learning algorithms typically expect for input.

size of the vocabulary is 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 (one or zero) in the vector representing this document. Instead of a simple OH representation the term frequency (TF) count can provide the cardinality of every words' frequency. But common words like articles or propositions are ubiquitously used and found in almost every document. Thus counting 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):

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

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

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

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 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 introduces other 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 a 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 adressed 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 based 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 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[3] around the target word co-occur. Provided with contextual input words CBOW predicts the best target word and vice versa Skip-gram predicts the contextual words based on a given input word. Both algorithms are categorized as unsupervised learners. They don't require explicitly labeled data for training as the words within any given real text sequence provide the

[3] Commonly a window size of 4-5 words is applied (Kowsare, et al., 2019, p. 8)

necessary learning signal automatically for free. *"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).

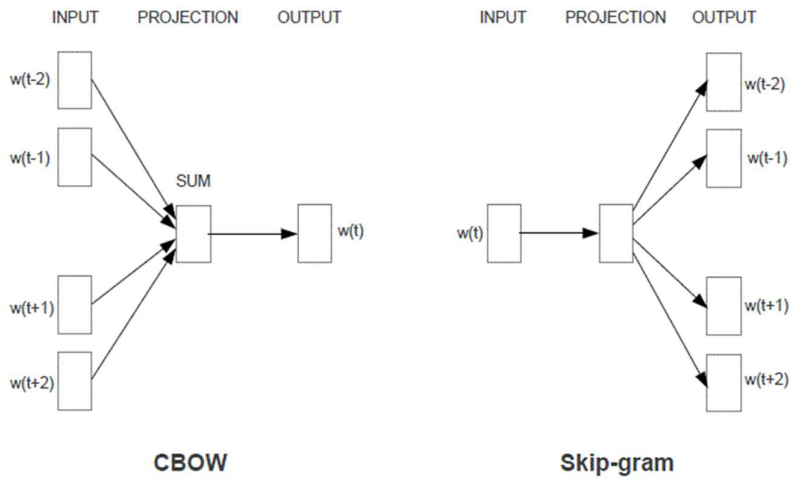


Figure 2-1: The CBOW and Skip-gram Architecture

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 θ 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 θ 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 d is an architectural parameter that can be experimentally chosen.

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. 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 a 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 used commonly in conjunction (like in "it is" or "should have") but with only little descriptive power and words showing together, because they relate to a specific semantic concept (e.g. "aircraft", "flying" and "pilot").

This shortcoming is met by **Global Vectors**. It's combining the strengths of Word2Vec with those of matrix factorization techniques by encoding statistical information about an entire corpus (Pennington, et al., 2014, p. 1532). The fundamental belief of GloVe is that co-occurrence ratios between 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 fully 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 the context of *j* is that a ratio above 1 indicates a stronger relationship of *k* and *i* than *k* with *j*. In contrast 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[4] 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:

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

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:

- $f(X_{ij} = 0) = 0$. 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 $f(X_{ij})$ is valued 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. With the common practice in many real-world applications to leverage available pre-trained embeddings (produced on large corpuses with potent computational resources) 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.

However both algorithms share one 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

[4] 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).

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 GloVe embeddings in combination with neural network architectures. Deep Learning models can start with 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 now 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 (Bishop, 2006, p. 125). *k*-NN is a lazy learner: It simply stores instances of the training data. To classify a new record at inference time it produces a ranking with the distances of this new instance to the existing 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 being more distant to the new record.

k-NN requires a set of stored records, a distance measure to compute the distance between records and a preset value *k*, determining how many neighboring instances should be evaluated for the majority vote when predicting the class label. The choice of the distance function and the *k* number of neighbors determine the accuracy of the classification. If *k* is chosen too small the model is sensitive to noise and outliers in the

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 set to be available 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 and latency. The 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 valued below the threshold.

Being a member of the Generalized Linear Model family Logistic Regression computes the dot product of input features ϕ and their weights w and transforms this weighted sum with the Logistic Function[5] $\sigma(\cdot)$ to present a posterior probability $p(C_1|\phi)$ as result of the calculation (Bishop, 2006, pp. 205, 197).

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

$$with \;\; p(C_2|\phi) = 1 - \; p(C_1|\phi)$$

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

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. 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):

[5] The Logistic Function is frequently referred to as the Sigmoid Function or simple the Sigmoid.

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

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 with a multi-class problem (*m* classes with $m > 2$) the chosen software implementation must adapt a certain method 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 for binary classification but OVR and other strategies can be applied to overcome this 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 a *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 its nearest data points is achieved. Hence the decision function is defined only by a subset of the data, aka the support vectors. They define the maximum margin possible for the

hyperplane that is fit. This property of a maximum margin classifier is particular useful to reduce the generalization error when the model is applied to new, unseen data later.

A Hard Margin SVM constructs a margin with no allowance for any errors. The decision boundary is constructed with a constraint to ensure that all records are kept on the correct side: 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) or the negative class (-1) the objective 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):

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

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

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 word problems present data that isn't strictly linear separable. Thus a Hard Margin Classifier cannot solve such a problem sufficiently. Those situations require a relaxation of the constraint 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: Data points are allowed to be on the wrong side of the decision boundary but they are increasingly penalized with growing distance to that boundary. To relax the constraints imposed by the Hard Margin Classifier a set of slack variables ξ_n is introduced with $\xi_n \geq 0$. Every data point is mapped to one slack variable carrying a penalty in the case of 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 ξ_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 these slack variables the optimization problem now contains two conflicting forces: On the one hand the maximum margin still needs to be found, whereas the total margin violation 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. With these conceptual additions the optimization problem for the Soft Margin Classifier is expressed as:

$$\underset{w,b}{\text{minimize}}\ \frac{1}{2}w^T w + \mathcal{C}\ \sum_{n=1}^{N}\xi_n$$

$$\text{subject to:}$$
$$t_n(w^T x_n + b) \geq 1 - \xi_n \ \text{ and } \ \xi_n \geq 0 \ \text{ for } \ n = 1, \ldots, N.$$

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 shown above 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 basic function (RBF) networks, and three-layer Sigmoid neural nets"* (Joachims, 1998, p. 138).

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. *"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 introduces some of those ideas and provides a brief description of their theoretical foundation. Our 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 predictions and targets. Training a neural network is trying to solve an optimization problem with regards to this function: *"Given neural network parameters θ, find the value of θ that minimizes the cost function $J(\theta)$"* (Goodfellow, 2015). Various loss functions cater for different needs. Choosing an adequate loss function for a specific learning task is a key decision to make.

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 target 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 generates a 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 $y_k(\mathrm{x}, \mathrm{w})$ normalizes this result vector of single probabilities (the outputs of the Sigmoid functions) into a probability distribution, ensuring that the networks' predicted K single probabilities add up to 1 for every document.

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

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

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

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

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

Equation 2-9: Gradient Descent

The learning rate η is an important hyper parameter. It's determining the speed of the training process and has a major part in supporting the algorithm to find the global optimum without getting stuck in local optima: While it will generally accelerate training a too high η might lead to overshooting: The global minimum of the cost function is not found because the algorithm skips over this optimum with a too large update in its local area and perhaps even deteriorates away from the best solution. A too small η might work better for that matter but could conversely prolong the network training substantially and potentially risk not converging because of the process being trapped in local minima, hindering further 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 is selected (Hinton, et al., 2012, p. 29). 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 $\nabla_\theta \mathcal{J}(\theta) \otimes \nabla_\theta \mathcal{J}(\theta)$ multiplied with a decaying factor β. Then the gradient of the current learning step $\nabla_\theta \mathcal{J}(\theta)$ is normalized by the root mean squared average $\sqrt{s + \epsilon}$:[6]

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

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

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

2.3.1.3 Strategies to prevent Vanishing Gradients

The training of neural networks 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[7]. Most of the activation functions used back in the day yield very small output values near zero. Computing the gradients of a deep neural net then means multiplying lots of these small numbers. Because 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 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). 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.

---

[6] A small smoothing term ϵ is added to prevent zero division

[7] This chaining of operations can also produce the unfavorable Exploding Gradients problem, which occurs when values larger than 1 are chained by thousands of multiplications scaling the upgrades of the ANN exponentially large.

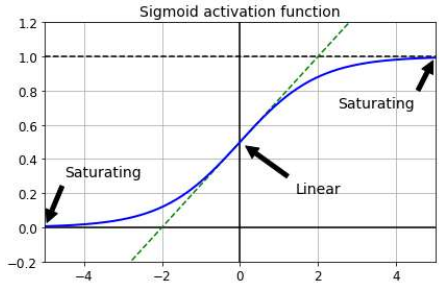


Figure 2-2: Logistic Activation Function Saturation (Géron, 2019, p. 333)

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 fast to compute. *"This activation function is the default activation function recommended for use with most feedforward neural networks"* (Goodfellow, et al., 2016).

$$ReLU\ (z) = \max(0, z)$$

$$LeakyReLU_{\alpha}(z) = \max(\alpha z, z)$$

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

But ReLU still has a limitation in it being not differentiable at zero value 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).

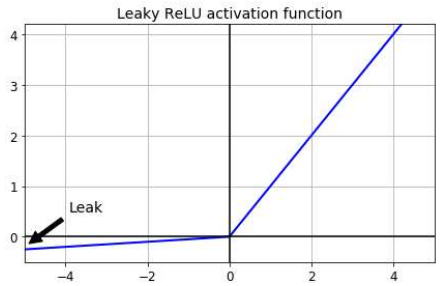


Figure 2-3: Leaky Rectified Linear Unit Function (Géron, 2019, p. 336)

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:

$$\mathcal{N}(\mu, \sigma)\ with\ \mu = 0 \quad and \quad \sigma = \frac{1}{fan_{avg}}$$

$$and \quad fan_{avg} = \frac{(fan_{in} + fan_{out})}{2}$$

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 (BN): *"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 channels its input from lower layers to higher layers applying BN smoothens the various input distributions and prevents an unfavorable high variance between inputs and outputs of a layer.

The key idea of BN is a normalization step built into the network architecture, ideally adding one BN layer before every activation of the network. This way every mini-batch[8] of inputs gets zero centered with a unit variance and then linearly transformed by a scale and a shift parameter vector γ and β. The model learns these two vectors for every layer[9].

[8] 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.

[9] For every layer specific input γ contains one scaling parameter and β one shifting parameter. Every input into the layer is scaled and shifted by its respective parameter set.

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[10] every x_i into $\hat{x}_i$:

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \quad \text{and} \quad \sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

$$\hat{x}_i \leftarrow \frac{(x_i - \mu_{\mathcal{B}})}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

$\hat{x}_i$ is linearly transformed using the scaling vector γ and
the shifting vector β, 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 and the network behaviour is less sensitive to the choice of its activation function. With more stability and robustness in training the applied learning rates can be increased, thus accelerating the training process considerably.

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 $\boldsymbol{L_1}$ and $\boldsymbol{L_2}$ **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).

[10] Note that in the denominator a smoothing term ϵ is added to avoid zero-division.

Batch Normalization in addition to its already described positive contributions (see chapter 2.3.1.4) 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 layers during model training. Every neuron (excluding the ones in 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 later. If a chosen performance metric (i.e. loss or accuracy) on the validation set improves the current weights of this training step are stored. The training terminates when the validation metric doesn't improve beyond a pre-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 weight parametrization, the best version (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 an imperative skill. Regular feed-forward networks can't do that.

Recurrent Neural Nets (RNNs) however 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.[11] The RNN processes the data x with a function f and parameters contained in θ 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 $t+1$. 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.

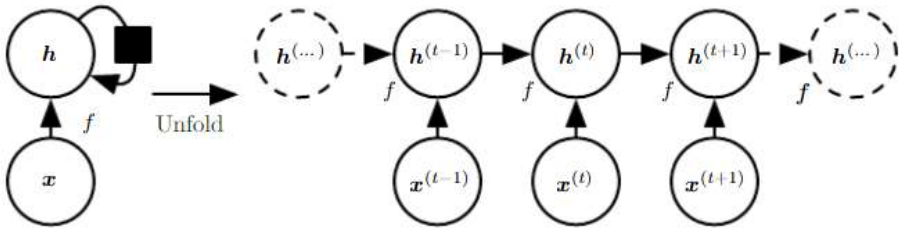


Figure 2-4: Unfolding of an RNN with no Outputs
(Goodfellow, et al., 2016, p. 370)

It's easy to see the 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: It produces no outputs other than the hidden states h. It doesn't have to be like that necessarily: 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 illustrates schematically different architectures of RNNs with variation in number of inputs and outputs.

[11] A time step in this context expresses foremostly the sequential position of an information, not necessarily the passage of a time period.

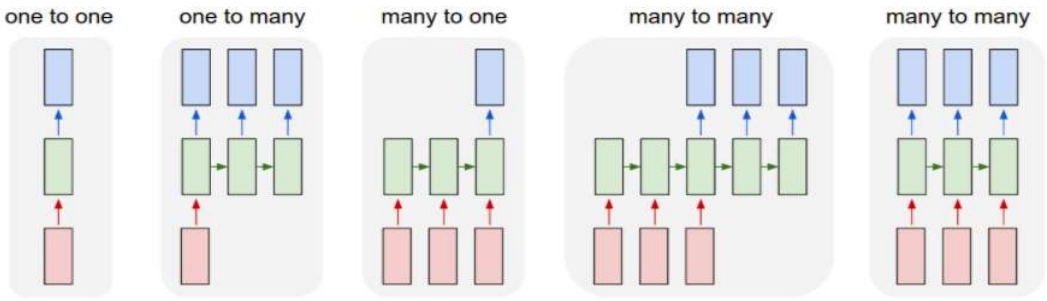


Figure 2-5: Archetypes of different RNN Architectures (Karpathy, 2015, p. 2)

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 assignment. This RNN produces an output vector *o* (indicated by the blue boxes) that can be fed to a multi-layered dense network with a final Softmax activation generating a distribution of probabilities over all classes for every record.

Three 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:

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$
$$h^{(t)} = \tanh\left(a^{(t)}\right)$$
$$o^{(t)} = c + Vh^{(t)}$$

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

The power of RNNs stems from two properties: A collection of hidden states, capable of preserving information through time and the usage of non-linear activation functions (like the hyperbolic tangent 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 though 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 RNNs suffer of two major limitations that ask for additional mitigating strategies, especially when long sequences of text (like documents) are processed:

- The complex chaining of hidden states over a large number of time steps makes RNNs especially deep. Very deep networks are even more susceptive for the vanishing or exploding gradients problem, resulting in an unfavorably instable 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 handicap.

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.

Specifically designed memory cells can learn what part of a previously stored context is not needed any longer and therefore can be removed from memory. Likewise they can determine what part of a new information is worthy of being added to this long short-term memory, thus 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 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.

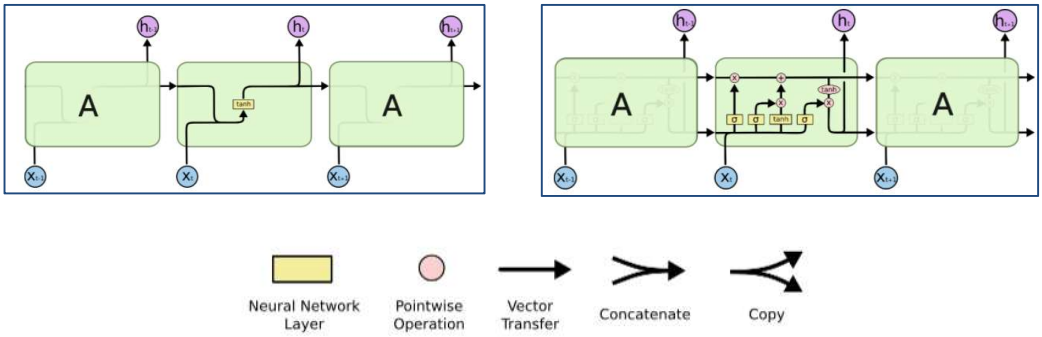


Figure 2-6: Schema of a hidden State in a regular RNN (left Panel) and (in the right Panel) a Memory Cell within an LSTM. (Olah, 2015)

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

- 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 the memory cell will take over some parts of the vector it is applied to (partial filtering).

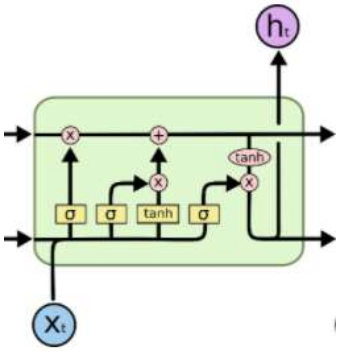
With that the inner workings in the memory cell can be formally expressed (Jurafsky & Martin, 2019, pp. 184-185) and illustrated (Olah, 2015) stepwise as follows:

(1) **Setting the gates**[12]: At time step t each gate f_t, i_t, o_t is computed by a respective Sigmoid σ 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.

$$f_t = \sigma\left(U_f h_{t-1} + W_f x_t\right)$$

$$i_t = \sigma(U_i h_{t-1} + W_i x_t)$$

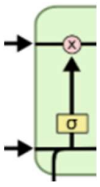
$$o_t = \sigma(U_o h_{t-1} + W_o x_t)$$



[12] The three gates are indicated with σ in the detailed illustrations.

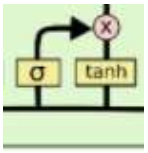
(2) **Removing context from the long term context vector**: The forget gate f_t is elementwise multiplied with the context vector c_{t-1} to remove context that is not needed any longer, resulting in the modified context vector k_t:

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



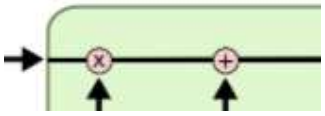
(3) **Computing the new current cell context:** The context to be added currently to the memory cell g_t is the output of a hyperbolic tangent applied to the sum of the multiplication of the input x_t and the hidden state h_{t-1} with their respective weight matrices U_g and W_g. By elementwise multiplication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

$$g_t = \tanh\left(U_g h_{t-1} + W_g x_t\right)$$
$$j_t = g_t \odot i_t$$



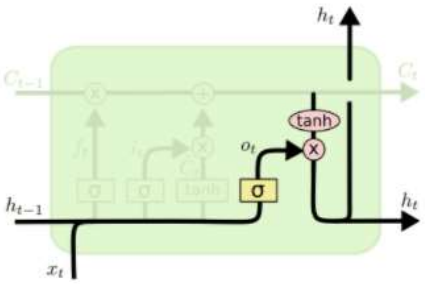
(4) **Updating the long term context vector c_t:** The current cell context j_t is added to the context vector k_t:

$$c_t = j_t + k_t$$



(5) **Writing the hidden state h_t:** The updated context vector c_t is transformed by another hyperbolic tangent and elementwise multiplied with the output gate o_t to produce 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 weights set an LSTM takes the hidden state and the long term context vector from the last time step and combines them with the current input. The previous hidden state and the current input determine the permeability of the gates in combination with their respective weights. 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 current 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 of the sequence (denoted with index ^{f}) up to the current time step *t*.

$$h_t^f = RNN_{forward}(x_1^t)$$

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

But comprehending 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 fully captured with all context considered. In a document classification task the entire context is typically available to support at training and also at inference time. Using all 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*:

$$h_t^b = RNN_{backward}(x_1^n)$$
$$h_t = h_t^f \oplus h_t^b$$

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 form 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).

It's unlikely for all words and sentences in a document to carry the same descriptive power. Some word co-occurrences, collocations or sentences might be much more distinguishing for a specific category. Emphasis should be directed to find exactly those patterns for certain classes. That's the driving insight for using CNNs in 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. It can be formally expressed 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 d_{emb}-dimensional word embedding $v(x_i)$. A *k*-sized window is moving *m = n-k+1* times[13] across the sequence of words and concatenates the corresponding embeddings of one window-slide to a series $(v(x_i) \oplus v(x_{i+1}) \oplus \dots \oplus v(x_{i+k-1}))$ representing one window w_i with $w_i \in \mathbb{R}^{kd_{emb}}$. This window vector w_i is linearly transformed and fed to a non-linear activation function *g()*. A 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 for this a Max Pooling[14] layer is applied producing a single vector c_j, representing every dimension *j* of the *m* vectors with its max value:

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

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

[13] This formulation represents a "narrow convolution", which applies a total number of *m = n-k+1* window movements over the input. Narrow convolutions potentially skip 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 results in *m = n-k+1* window movements and represents a "wide convolution" (Kalchbrenner, 2014).

[14] The intuition behind selecting the maximum 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.

A convolutional layer isn't limited to 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). 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.

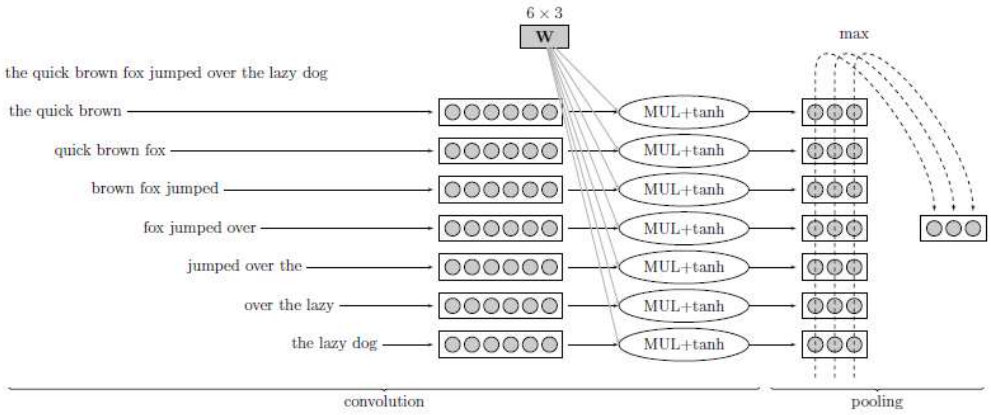


Figure 2-7: Example of 1d-Convolution and Max-Pooling with 3 Filters (Goldberg, 2015, p. 44).

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

It's beneficial to account for the order of the sequential input and the context of words when processing text. While proximity of words is often helpful to capture a semantical concept, it's not necessarily sufficient. Use cases where context is spread across an entire document with multiple parts co-referring to each other are still a challenge for a sequential network architecture: Training LSTMs on very long sequences (like documents) 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 together with contextual focus information. Thus each token carries additional information determining 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. 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. Figure 2-8 illustrates the Transformer architecture:

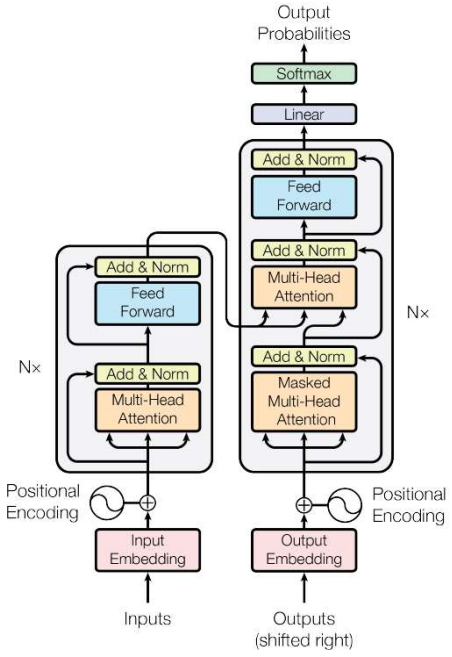


Figure 2-8: The Transformer Model Architecture (Vaswani, et al., 2017, p. 3)

With this project focusing on document classification the decoder (the right part of the model architecture shown in Figure 2-8) is omitted for further discussion and emphasis is given to the encoder (the left part). In 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) and not fed to the decoder. The encoder is made up of *N=6* stacked identical layers. Within each layer the data traverses two consecutive sub-layers: A Multi-Head Attention block and a Feed Forward block with residual connections[15] 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 each time step, the sequential order of the tokens is maintained and there is no need for a dedicated positional encoding. But the encoder module of a Transformer model expects the inputs with such a positional encoding. Like in RNNs the input is presented as embedding vectors.[16] But unlike sequentially working models the Transformer takes in the entire input sequence at once, entailing a big advantage: By usage of GPUs and parallelization training time is significantly reduced. But with no sequential information encoded any longer, the order of the tokens is lost and cannot contribute to the model. Thus the authors introduced a method to preserve it: 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*. The two vectors are added to one combined vector. Thus it suffices to use this one vector, carrying the initial embeddings plus the positional information additionally encoded.

---

[15] The residual connections are added to prevent the gradient signal from thinning out during backpropagation.

[16] The original Transformer model described by Vasvani et al uses 512-dimensional embedding vectors.

2.4.2 Scaled Dot-Product Attention

The Positional Encodings are fed to a Multi-Head Attention layer (see Figure 2-8) that can be decomposed into single attention units called heads. Each head computes a Scaled Dot Product Attention.

The embedding vectors of the text are parallelly kept in three matrices: the query matrix Q, the key matrix K and the value matrix V. Self-Attention makes use of cosine similarity: The cosine of the angle between two vectors approaches 1 the more similar they 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[17] to keep numerical computation stable and then normalized with a softmax function to produce weights that add up to 1.

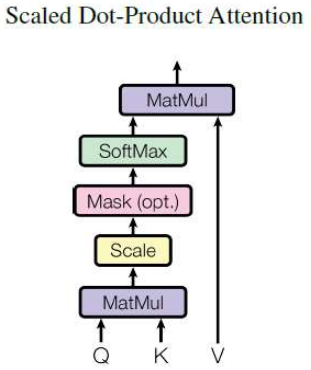


Figure 2-9: Scaled Dot-Product Attention
in the Transformer Model Architecture (Vaswani, et al., 2017, p. 3)

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:

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

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

[17] The authors used $\sqrt{d_k}$ to scale down the matrix multiplication QK^T

2.4.3 Multi-Head Attention

Conventional word vectors represent a meaning of a word relatively stable across different usages. But the relationship expressed by an attention mechanism can be manifold. For a specific input sequence, attention can be given to many 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 implements this concept with the Multi-Head Attention Layer. The Scaled Dot-Product Attention described above using the input matrices *Q*, *K* and *V* is calculated *h* times in parallel and uses a separate weight matrix W^Q, W^K and W^V every time: *"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).

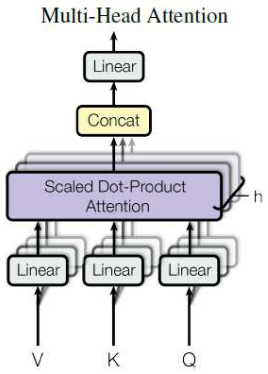


Figure 2-10: Multi-Head Attention
in the Transformer Model Architecture (Vaswani, et al., 2017, p. 3)

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:

$$MultiHead(Q,K,V) = \text{Concat}(head_1, \dots, head_h)W^o$$
$$where\ head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right)\ and\ i = 1, \dots, h$$

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

Attention and Transformers have fueled many research projects: *"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).

Chapter 2 introduced different methods to encode textual input into a numerical representation that can be used for classification algorithms and networks. Leveraging the different encoding techniques shown numerous classification techniques can be grouped into three general approaches to address a document classification task:

(a) Traditional Machine Learning approaches
(b) Deep Learning approaches
(c) Transfer Learning

Within the different approaches different concepts have been introduced together with some of their respective theoretical foundation. All methods presented are applied within the experiments, which are detailed and described in chapter 4 and their results being reported in chapter 5.

Chapter 3 will now continue with a description of the specific data environment for this use case of BMW Bank.

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

In 2020 a team of BMW Bank experts implemented a new DMS. A complex rule-based logic was developed, configured into the software solution, tested and deployed to a production environment. The classifier to be developed within this project is to solve for all documents that this DMS doesn't classify 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 daily to clear this backlog manually by assigning the correct document type for every document. These manually inspected and labeled records are used to train the desired document classifier. A weekly set of training data has been exported from the production system into a hierarchical file structure on a secured sharepoint:

🗁 BatchClass => 🗁 BatchID => 🗁 DocumentID => 🗐 Data Files

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

Content file[18] /Schaden/1182502/2949971/002D0353.txt:	Index file with metadata …/002D0353_index.txt:
'Page 1\nFrom:\t"Kanzlei Meyer" <info@muster.de>\n Date:\tThu, 17 Dec 2020 11:18:36+0100\n To:\t\'"BMW Leasing GmbH\'" schaden@bmw.de>\nSubject:\tKunde: Muster Erik, Muster Weg 5, 12454 Musterstadt Verkehrs-unfall 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 zivil-rechtlichen Schadensregulierung betreffend oben genannten Verkehrsunfall\n beauftragt …	"Schaden","Schaden PDF Image + Text","{BatchID}","1182502","{DocumentID}","2949971","DocumentType","SCHADENSCHREIBEN","ClassificationResultWithConfiden", "SCHADENGUTACHTEN;60;P|SCHADENANZEIGE;40;P|SCHADENSCHREIBEN;20;P","PageCount","1","{DocumentCount}","1","{$InputChannel}","EMAIL","{$sourceSystem}","EMAIL","AutoClassificationConfidence","0",""\n

A weekly batch of training data contains up to 20k documents. The exported file structure is uploaded onto a High Security Data Analytics Platform (HSDAP) equipped with a Python 3.8 environment for data preprocessing and development. The files are

[18] 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..

parsed into a dataframe with one column holding the raw document text and further columns representing metadata for each document. The predictors include the target class information ("DOCTYPE"). Table 3-1 lists the different columns of the dataframe, their data type and a brief description 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 is the basis for further preprocessing steps before the data gets clearance for model training.

3.2 Anonymization of sensitive Data

Given the sensitive nature of the documents a thorough data risk assessment has to be performed and an anonymization concept to be developed. The following outlined routine of data protection measures and additional cleaning preprocessing is applied to every batch of data retrieved from the source system:

1. **Blacklist check:** To remove particularly sensitive senders (i.e. Rolls Royce© clients) a set of 109k email addresses is matched against the content of each document. Documents are tokenized using an open sourced pre-trained German language model[19] and then scanned for positive matches, which are to be removed. Typically 9% of all documents in a procured data batch are removed by this step.

[19] 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.

2. **Rough cleaning:** The remaining documents are cleaned with a number of string operations: Non-printable characters are removed and typical conversational phrases and boiler plate copy is replaced with a short tag.

3. **Regex rules replacing and tagging critical data:** Sensitive personal data is identified by a set of 32 tailored regular expression rules searching the text for specific banking, financing and automotive expressions with personal data reference. Positive matches are replaced in the text with a specific tag like "<KFZK>" replacing a car registration plate for example.

4. **Named Entity Recognition:** Documents are to be screened for further personal data, specifically peoples' names and addresses. Using the pre-trained German Language Model of Spacy the Named Entity Recognition (NER) functionality is used to identify peoples' names, addresses and company names. Entities found in the text are replaced with a specific tag ("<PER>", "<LOC>" and "<ORG>").

Applied to the example above this routine would produce this text:

'Page 1 From: "<ORG>" <EMAIL> Date: Thu, <DATUM> 11:18:36+0100 To: \'<ORG>\'"<EMAIL> Subject: Kunde: <PER>, Muster Weg 5, <PLZ> <LOC> Verkehrsunfall vom: < DATUM > Pkw, amtl. Kennz.: <KFZKZ> - <PER> ./. <PER> - Unser Zeichen: 1234/20-GG <ANREDE> , Ihr o.g. Kunde hat uns mit der zivilrechtlichen Schadensregulierung betreffend oben genannten Verkehrsunfall beauftragt. …

The preprocessed dataframes are persisted on the HSDAP. Clean and cleared training data is uploaded onto a Deep Learning Platform (DLP) of the BMW AG, equipped with resources for model development and training.

3.3 Explorative Data Analysis (EDA)

The routine described above yielded a total consolidated sample of labeled training data with $N = 62734$ records, accounting only for manually inspected documents that the DMS failed to classify automatically in the first place. As listed in Table 3-1 the data provided the text for each document but also metadata like the input channels, the page volume etc. and - indispensably for model training - the target class information:

3.3.1 Document Types

Document types map documents to a business related process. Based on this documents are allocated to relevant inboxes for further processing. Document types at BMW Bank can describe very department specific intents but they can also be of a more generic nature, relating thematically to different departments. Thus they can be found across

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

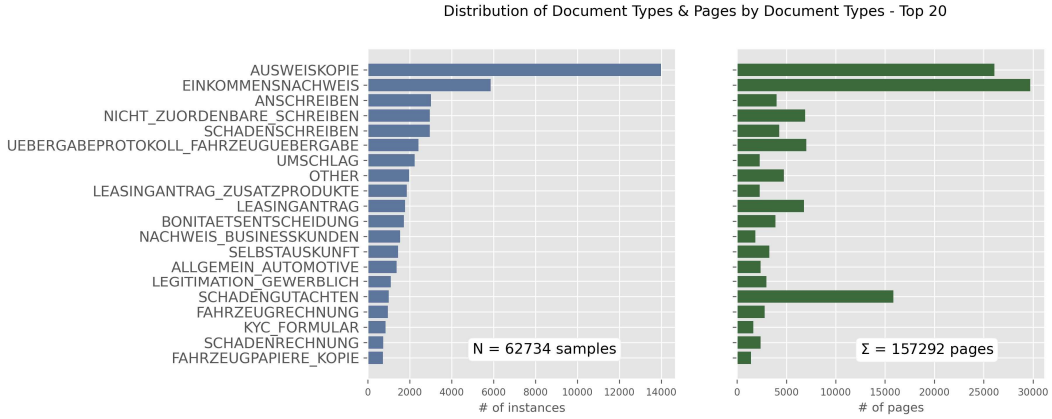


Figure 3-1: Distribution of the Top 20 Document Types
(Instances & Page volume)

The distribution of the 154 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 instances. With emphasis on frequent tasks in daily operations document types with an occurrence less than 100 instances in the sample 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 the plain amount of 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 shown in the left panel. Some exceptions exist where a category with low incidence 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 for model training. It will be addressed adequately with the metrics and the training parameters used (see chapter 4).

3.3.2 Other Metadata

Table 3-1 lists all information retrieved in conjunction with the text. Metadata provided by the DMS can be used for additional features. There are 4 categorical features and 3 numerical features. Table 3-2 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 dedicated inbound routes (i.e. specific inbound email addresses like "schaden@bmw.de"). "Neugeschäft" is the predominant subject accounting for more than 70% of all documents. It describes an electronic upload channel used by retailers to provide new contractual documents for leases and loans. Given the thematic relationship with different departments and business units the batch class feature might support well the document classification task.

The "BATCHCONTENT" feature specifies technical aspects of the documents. More than 80% of documents are derived from an OCR scan of an image. 23% of these are sourced from an image without any further text. Those documents are occurring in the 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 significantly 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 between them. Again the dominance of uploaded content shows in the categories "UPLOAD", "FASTLANE" and "DOCSTORE".

"PAGECOUNT", "DOCCOUNT" and "NBR_DOCTYPES" describe numerical features of the data. "PAGECOUNT" equals the page volume of a document and "DOCCOUNT" describes how many other documents this specific document is inter related with.[20] The number of document types ("NBR_DOCTYPES") is calculated from a candidate list of document types that the DMS has already (unsuccessfully) inferenced on during its initial classification attempt. Figure 3-2 shows the distribution of the 3 numerical predictors split by different batch classes.

[20] Single documents can be interrelated. For example the registration for a new customer might include an application for a lease together with other documentation of income, id-cards, proof of address, etc..

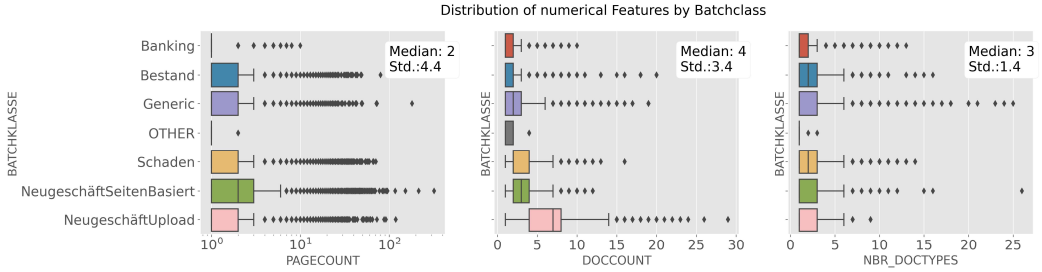


Figure 3-2: Distributions of Numerical Features by Batch Class

Looking at page volume with a median page count of 2 pages and a standard deviation of 4.4 pages, the distribution shown in the left panel of Figure 3-2 illustrates many outliers, some of them even exceeding 100 pages for a document.

Looking at the "DOCCOUNT" distribution (middle panel) the category "NeugschäftUpload" 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 (right panel): The DMS typically estimates around 3 (median) different document types to be valid candidates, but looking at the outliers portrayed there can be uncertainty with listings up to 25 different document types as potential class candidates for a given document.

For model training the categorical features are OH encoded. The numerical features are normalized to prevent unfavorable bias effects of outliers (for details see chapter 4.2).

3.3.3 Text Data

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

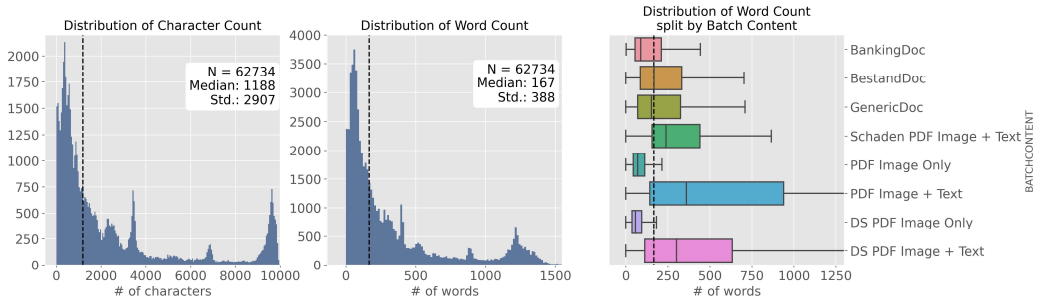


Figure 3-3: Distribution of word and character count in the N documents sample

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.

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 document. Naturally documents with the property "Image Only" (like driving licenses, vehicle registrations or ID-cards) show word count significantly below the average documents. 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 with their respective word count distribution.

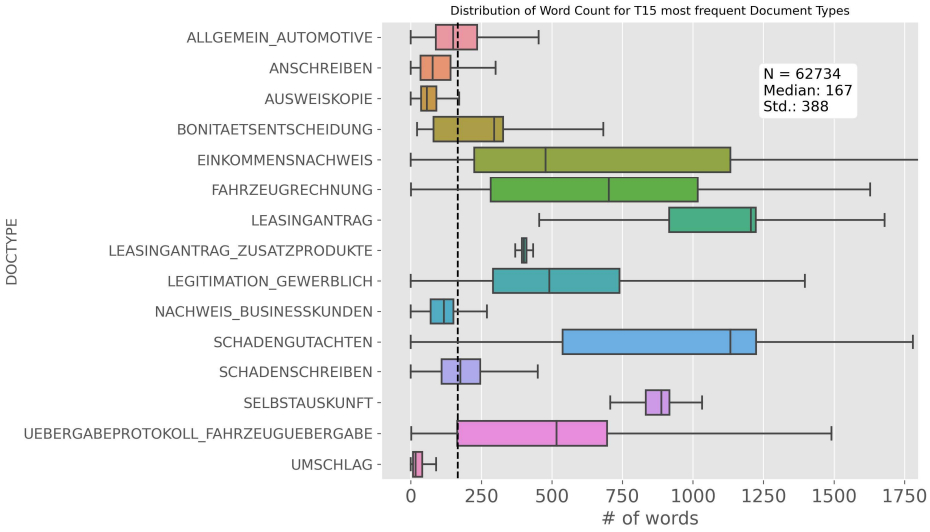


Figure 3-4: Distribution of word count split by Document Type

Documents of the most frequent class "AUSWEISKOPIE" contain a lot less text than the average documents. "SCHADENGUTACHTEN" (damage assessment) on the other hand, an important document type contributing significantly to total page volume (see Figure 3-1) shows a much higher variance and median of word counts for its documents. Given the heterogeneous distributions of word count by different document types the length of a document might be a strongly contributing predictor in the classification task. Though more words in a document might not necessarily help. Potentially more words could introduce more noise and clutter. Given the significance of OCR processed documents combined with the observation that their scan quality is often questionable, it is critical to measure how much "noise" is contained in the data.

Without a standardized metric to operationalize "amount of noise" in a corpus an approximation is required: The vocabulary built of the given sample is compared to

vocabularies extracted from publicly available large German text corpora. For that a united vocabulary of the following three publicly available sources[21] is build up and used as a proxy 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 dictionaries are consolidated to a Gold Standard vocabulary with 1.493 million unique words that can be matched against the given vocabulary of the sample corpus with $N = 62734$ documents. Matching the 10k most frequent expressions of this sample corpus with the Gold Standard shows that 38% of these 10k terms are not represented in the Gold Standard.

For further insights this match is additionally performed on the vocabularies generated for every single document type: Table 3-3 shows a split analysis of "noise by document types". The ten "most noisy" document types are listed in the left panel 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	ADRESSAENDERUNG	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 total vocabulary lengths ranging from 3383 words to 199791 words with a median of 11449 words. Regarding the "share of noise" the results range from the noisiest category "DARLEHENSANTRAG" with 62% to the least noisy category "WIDERUF_VERTRAG" with only 16% of the 10k most frequent terms not found in the Gold Standard. Computing the correlation between the vocabulary length and the share of noise yields a Pearson Correlation Coefficient R^2 of 0.32, indicating a positive co-occurrence of longer vocabularies and more noisy text.

[21] © ExplosionAI GmbH. For full documentation please refer to 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/

Across all 59 document types a median noise level of 36% summarizes the assessment that the data in the given sample corpus is indeed very noisy with more than every third token not matching against the Gold Standard benchmark. The data is largely received from preprocessing systems applying OCR technology that in addition to an already poor quality of the initial raw document (poor photocopies, mobile phone pictures etc.) introduces additional bias during processing.

With this insight of the training data being as noisy as shown, applying different classification approaches like 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 published.

4 Experimental Setup

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

(a) **Traditional approaches** using different classic algorithms long established in Machine Learning theory together with a BOW/TF-IDF vectorization of texts.

(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.

The focus of this chapter is a brief specification of the models, their key parameters and architectural choices. Chapter 5 will subsequently follow-up with a detailed report of the results from these three different routes of 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 in both partitions. All experiments are evaluated on the test set with $N_{test} = 15684$ 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 class specific F1 scores is used to express overall model performance.

All experiments are conducted on a Python 3.8 runtime environment. The 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© framework (version 2.3) and the Keras© API. The Transformer Model experiment specified in chapter 4.4 makes use of the Transformers library (version 3.5.1) published by Hugging Face©. The respective websites for theses frameworks provide extensive documentation of the classes and algorithms used in the experiments.

4.2 Machine Learning Approaches

Preprocessing is applied to the train and test data with a Column Transformer object performing OH encoding for the categorical variables, normalization for numerical predictors and a TF-IDF vectorization for the text applying lowercase conversion and limiting the vocabulary size to 10k terms. This pipeline is used for all experiments described in this chapter.

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[22]. 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 considered neighbors *k*, the weighting scheme amongst them and the distance function to compute the neighborhood amongst all instances in the training set. Parameter combinations are applied for finding the best combination using different values for *k*. Table 4-1 shows the key parameter settings applied during the *k*-NN experiments.

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

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

[22] Bayes Theorem is founded on the works of Reverend Thomas Bayes in the 18th 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 given multi-class context. The algorithm is fed a weighting scheme with 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.

# Classfier (class)	Parameters	Parameter Values #
Logistic Regression ***sklearn.linear_model.LogisticRegression()***	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 also accepts a weighting scheme. Scikit-learn offers a specific Linear SVM class optimized for linear problems: By 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. C scales the impact of the slack variables (see Equation 2-7).

# Classfier (class)	Parameters	Parameter Values #
Linear Support Vector Machine ***sklearn.svm.LinearSVC***	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 using 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).

# Classfier (class)	Parameters	Parameter Values #
Stochastic Gradient Descent Classifier ***sklearn.linear_model.SGDClassifier***	penalty:= Regularization norm	> l1 > l2
	loss	> hinge,squared hinge > modified_huber > log_loss
	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 **SVM**s to fit a hyperplane in high-dimensional space even with data not being linearly separable. The Support Vector Classifier class provides for kernel functions like a Polynomial, a Radial-Basis Function or a Sigmoid. As with the Linear SVM the optimal setting of the regularization parameter *C* is important.

# Classfier (class)	Parameters	Parameter Values #
Support Vector Machine ***sklearn.svm.SVC***	C := Regularization Strength	0.55, 0.66, 1.0, 1.5
	kernel := Kernel Function	> linear > poly > rbf > sigmoid
	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. Since the weak learners in the ensemble are built of decision tree models there is a wide range of parameters for controlling the tree production (growing and pruning). For the experiments applied herein the respective default parametrization of these parameters is used.

# Classfier (class)	Parameters	Parameter Values #
Gradient Boosting ***sklearn.ensemble. GradientBoostingClassifier***	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 Deep Learning approaches

This chapter specifies a template for a General Training Architecture (GTA) that is uniformly used for all Deep Learning experiments, which are plugged into the GTA template and are described in detail thereafter.

4.3.1 General Training Architecture

The metadata retrieved with the documents provides interesting features (see chapter 3). The Tensorflow/Keras Functional API allows to design model architectures accepting different inputs (text, categorical and numerical) with individual processing strands to them. With this flexibility a general template (schematically illustrated in **Fehler! Verweisquelle konnte nicht gefunden werden.**) for the Deep Learning experiments is built up of the following components:

- Numerical and categorical input is preprocessed with normalization and OH encoding as described in chapter 4.2.

- For the Deep Learning experiments[23] the textual input is truncated or padded to a sequence length of 512 tokens. Again the vocabulary size is capped to the most frequent 10k terms. The TextVectorization class 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 integers.
- The vectorized sequences of 512 integers (representing one document) are fed to a plain, randomly initialized embedding layer. The layer is kept "trainable" to be able to adjust alongside the training process with the gradient signal propagating backwards through the network. Some experiments apply pre-trained embedding vectors to initialize the embedding (see chapter 4.3.4).
- A 300-dimensional output from the embedding layer is fed to a text model layer, which is implemented with different Deep Learning architectures within the different experiments (see chapters hereafter).
- The output of the text model and the categorical and numerical inputs are concatenated and fed to a dense feed forward network comprised of:
- 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 a softmax activation producing the networks' output vector of probabilities for each class.

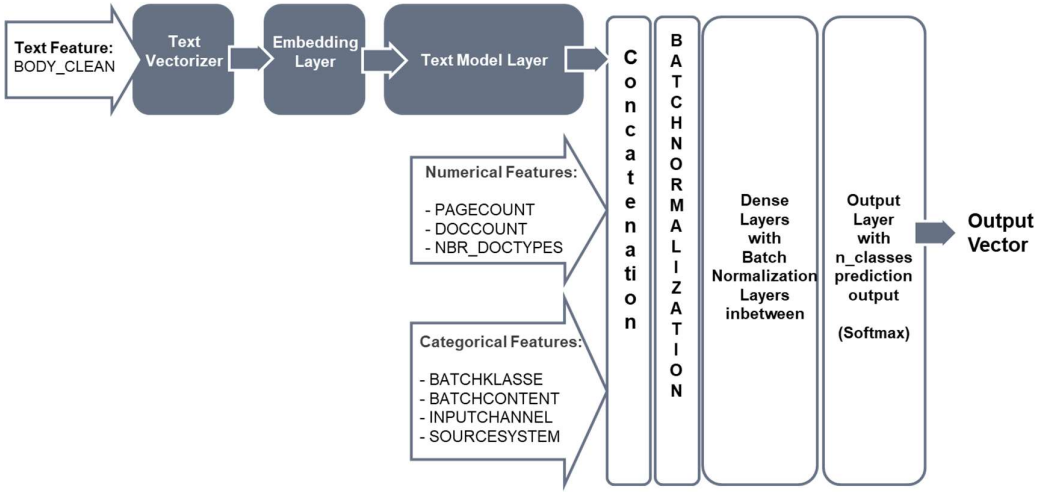


Figure 4-1: General Training Architecture used for the Deep Learning Experiments

[23] The Transformer Model chosen requires a specific preprocessing and tokenization for the input texts. The herein described vectorization and embedding steps are only applicable to the Deep Learning experiments described throughout chapter 4.3.

During training the networks parameters are incrementally updated at each learning step to minimize a cross entropy loss function (described in chapter 2.3.1.1). A sparse categorical cross entropy implementation allows for training with labels encoded as integers (and not as OH-encoded arrays).

The RMSprop optimizer is chosen with all default settings kept but the learning rate is set to 1.0×10^{-4} or 1.0×10^{-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 but 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 placed 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).

4.3.2 Bidirectional Long Short Term Memory

At first the Text Model layer within the GTA is implemented with a Bidirectional Long Short Term Memory model (BiLSTM). It's built of two LSTM layers combined in a bidirectional wrapper, allowing the textual input to be processed in a forward and backward fashion. All hyper parameters are kept within the default settings.

The 600-dimensional output (two 300-dimensional vectors concatenated) of the BiLSTM is fed into the concatenation layer as described above (**Fehler! Verweisquelle konnte nicht gefunden werden.**).

4.3.3 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 before the output is fed to the subsequent layers of the GTA. Within the experiments varying specifications for convolutions with different 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) applied parallelly in one model some experiments use two parallel CNN layers in the architecture with a consolidating concatenation layer thereafter. These convolutions are specified through a combination of 300 filters/5 kernel and 300 filters/7 kernel. All experiments on the CNN layers are run with ReLU activation functions. 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.

4.3.4 CNN with pre-trained Embeddings

The experiments described above make use of a plain Embedding Layer, randomly initialized 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. 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 the contained noise in the 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 of zero vectors in the embedding matrix (see Table 4-7 below). Thus limiting the potentially benefit 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 specific embedding matrix is used for weight initialization of the Embedding layer and all experiments are run with the layer kept "trainable" throughout the learning process.

4.4 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 $BERT_{BASE}$ version and a $BERT_{LARGE}$ version. $BERT_{BASE}$ 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[24] 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 published by HuggingFace offers a general API to a variety of language 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.

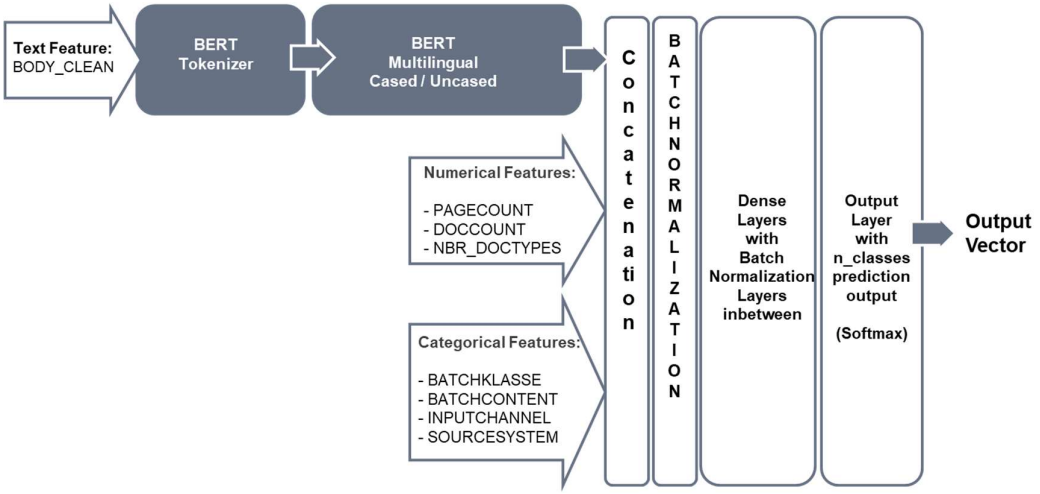


Figure 4-2: BERT Multilingual & General Training Architecture

The BERT implementation used in the experiments trains more than 177 million parameters. With a computational budget to be observed the BERT experiments are

[24] See documentation of the model: https://huggingface.co/bert-base-multilingual-uncased?

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 1.0×10^{-4} and class weights (page volume) applied during training.

The textual input is transformed using a dedicated tokenization scheme required for the BERT model: It's using 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).This specific vectorization is performed with a pre-defined tokenizer class available through the HuggingFace API[25]. The BERT model layer is configured for not outputting its hidden states.

[25] Documentation: https://huggingface.co/transformers/model_doc/bert.html#berttokenizerfast

5 Results

The experiments are grouped in three flights as laid out in chapter 4. All Results are reported on the evaluation performed on the test set with N_{test} = 15684 instances (25% of N). After reporting all flights of experiments the best performing models are compared to one another to identify individual strengths and weaknesses. A final conclusive experiment synthesizes these findings into a combined ensemble estimator.

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 pipeline. Chapter 5.1.1 reports the top line results for the different algorithms employed, before the next chapter analyzes the best performing model with an in-depth exploration of its results.

5.1.1 Machine Learning Classifiers – Top Line Results

Two classifiers requiring only little hyper parameter tuning set the **baseline**:

Top Line Results - Machine Learning Models with BOW/TF-IDF: Baseline		
Model	**Parameter**	**Weighted F1**
Multinomial Naïve Bayes	fit_prior = w. uniform distribution	.835
Multinomial Naïve 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

- The Multinomial Naïve Bayes classifier delivers an instant .842 weighted F1 when trained with informed prior-probabilities of the classes (page weights).
- Despite numerous runs with different distance functions and variations of *k* the *k*-NN classifier falls short compared to this. 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.

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_huber, L2	.894
Stochastic Gradient Descent	loss=hinge, L2	.889

Table 5-2 shows the results of these 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_huber, L2	.894
Stochastic Gradient Descent	loss=hinge, L2	.889

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

- All linear models deliver weighted F1 scores far above baseline. The only exception to this is the Logistic Regression with class weights applied. The results of runs with weighted instances employed are generally below those runs without weighting scheme.
- Logistic Regression scores considerably high achieving a weighted F1 of .889.
- Linear SVM also yields better results without weighting applied and scores 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**. Table 5-3 summarizes the performance of SVMs run with different kernel functions[26] and a run of Gradient Boosting Classifier training.

[26] 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

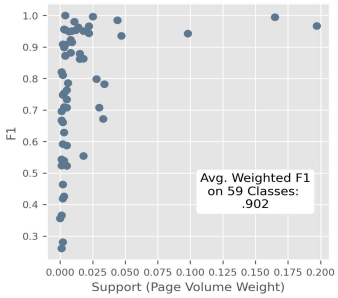
Table 5-3: Non-Linear Model Experiments: Weighted F1 Results

- Using SVMs with varying kernel functions doesn't yield better 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 this tree-based approach (weak learners are trees). Without extensive hyper parameter tuning to control the behavior of the tree production process Gradient Boosting cannot convince.

5.1.2 Exploration on the Linear SVM performance

Summarizing the results reported above the Linear SVM with a low *C* regularization (.55) delivers the best weighted F1 (.902) in the first flight of experiments.

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



Linear SVM Classification Results: Weighted F1-Scores by Classes

Top 15	Wg. F1	Support
FAHRZEUGPAPIERE_ORIGINAL	1.0	.004
BONITAETSENTSCHEIDUNG	.994	.024
AUSWEISKOPIE	.994	.170
LEASINGANTRAG	.977	.045
KYC_FORMULAR	.975	.011
EINKOMMENSNACHWEIS	.974	.186
FAHRZEUGRECHNUNG	.972	.018
LEASINGANTRAG_ZUSATZPRODUKTE	.969	.014
SELBSTAUSKUNFT	.966	.022
FAHRZEUGPAPIERE KOPIE	.961	.009
SCHADENGUTACHTEN	.956	.099
NACHWEIS_BUSINESSKUNDEN	.950	.012
BUERGSCHAFT	.949	.001
RUECKNAHMEGUTACHTEN_TUEV	.942	.007
UEBERGABEPROTOKOLL_FAHRZEUGUEBER	.936	.007

Low 15	Wg. F1	Support
SCHADENRUECKINFORMATION	.242	.001
ABLOESE_INTERN	.299	.001
ANFORDERUNG_UNTERLAGEN	.303	.001
ANFRAGE_VORZEITIGE_VERTRASBEENDI	.337	.001
SCHADENANZEIGE	.440	.002
SCHADENABRECHNUNG	.472	.005
ANTRAGE_VERTRAGSBEENDIGUNG	.490	.003
ALLGEMEIN_AUTOMOTIVE	.553	.019
ANFRAGE_VERTRAGSAENDERUNG	.556	.005
LEASINGANTRAG INFORMATIONEN	.576	.003
RUECKLAUF_VERTRAGSAENDERUNG	.583	.002
ABLOESE_UND_VERSANDVOLLMACHT	.600	.001
ABLOESE_FREMDKREDIT	.603	.002
OTHER	.619	.028
WIDERRUF_VERTRAG	.623	.001

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

11 classes 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) focusing on the most error-prone classes two patterns emerge:

(1) Classes describing a generic semantic concept with less distinctive language are more likely to mislead the classifier. This can be seen when looking at the document types "ALLGMEIN AUTOMOTIVE", "ANSCHREIBEN", "ALLGEMEIN BANKING", "NICHT_ZUORDENBARE_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 with them in the 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 also drive misclassifications. This can be seen with the classes that contain documents being related to the concept of "SCHADEN" (damage). Classes dealing with damage management are frequently misclassified to the label "SCHADENSCHREIBEN". This class is the most generic concept amongst the "SCHADEN"-related categories (with a high incidence).

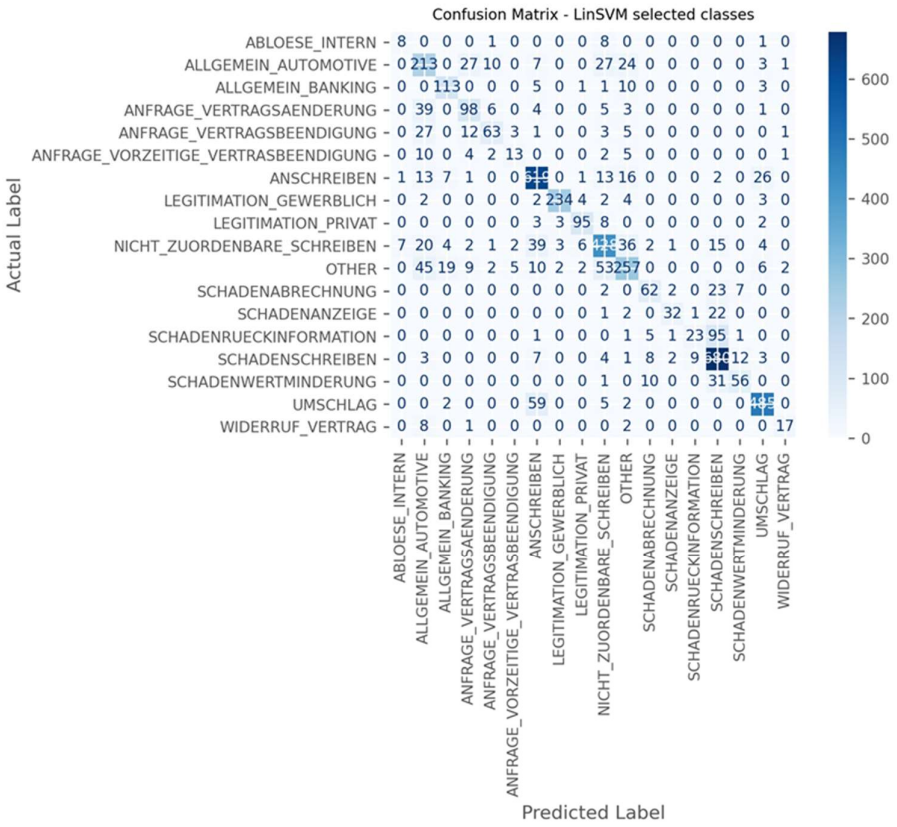


Figure 5-1: Linear SVM: Confusion Matrix for selected error-prone Classes

The coefficients of the Linear SVM classifier can be inspected for further insights to its prediction logic. The trained classifier object holds an array with shape 59 classes and 10030 coefficients, representing the features used in the linear model. The weights can be ranked according to their absolute value as a proxy for feature importance. Table 5-5 shows the approximated feature importance of the 10 most distinctive features for three exemplarily selected categories, which are all related to the concept of damage ("SCHADEN") but show diverging F1 scores.

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

Looking at "SCHADENSCHREIBEN" a category only moderately handled by the classifier (F1 score of .81) the ranking is led by the rather generic terms "eur", "betrag" or "intro". Based on this the classifier is not using very distinctive language to inference on this class. "SCHADENRUECKINFORMATION", the class with the lowest F1 (.242) of all categories shows even less distinctive language used by the model: The most important word is "zwischenzeitlich" (meanwhile). This absence of descriptive language explains the poor performance of this class, with many predictions falsely given to the label "SCHADENSCHREIBEN" (for further insights see also the analysis in chapter 5.4).

"SCHADENGUTACHTEN" on the other hand, a frequently occurring class with an even higher weight yields a strong .956 in weighted F1. The most important feature is the page count information. As previously indicated by the split of word count by document classes (compare Figure 3-4) the model can yield these convincing results for this category by leveraging the available document length information with the PAGECOUNT predictor being the most important feature.

Inspecting classes with even better prediction results Table 5-6 provides insight to another three top scoring categories and their most important features:

Linear SVM Classification - Coefficients (abs*) for selected Classes					
BONITAETSENTSCHEIDUNG [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 very distinctive language regarding the concepts they refer to. "BONITAETSENTSCHEIDUNG" and "LEASINGANTRAG" utilize expressions like "bonitätsprüfung", "darlehensnehmer" and "leasingantrag".

Looking at the category "AUSWEISKOPIE": The expressions "passport", "idd" and "cm" are strong predictors for the classifier in this case. German passports and ID-cards frequently carry the character sequence "idd" in the beginning of their machine readable zone. The sequence "cm" is referring to a person's height measured in centimeters, printed on every passport. Additionally the model utilizes the BATCHCONTENT feature (metadata) with these documents being of "Image only" type.

Summarizing the results of the traditional Machine Learning models: They generally can 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 necessarily hinder the models with BOW/TFIDF vectorization to learn meaningful patterns for most of the classes and in particular not for the important classes carrying a higher weight measured by their page count volume.

Chapter 5.2 will now focus on the Deep Learning Experiments approaching the document classification task by utilization of the Embedding techniques described in chapter 2 and chapter 4.

5.2 Deep Learning Models leveraging Embeddings

The second flight of experiments employs different Deep Learning architectures to the problem at hand. All experiments are run in the GTA framework (see **Fehler! Verweisquelle konnte nicht gefunden werden.**) with the 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 experiments.

The first idea put to test is a BiLSTM architecture (for details see chapter 4.3.1). This 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. Tabel 5-7 reports the weighted F1 scores resulting of those experiments.

Top Line Results - Deep Learning Experiments w. (not pre-trained) Embedding Layer		
Model Architecture	**Key Parameters**	**Wght. F1**
Abbreviations: SQL Sequence Length, BS Batchsize, LR Learnrate, CW Classweights, EP Epochs		
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	***.883***

Table 5-7: BiLSTMs – Top Line Classification Results

The scores improve with a more granular learning rate of 1.0×10^{-5} and a batch size of 30. The best performing BiLSTM experiment yields a .883 weighted F1, close to the linear models in general and slightly behind the Linear SVM (see Table 5-2).

The next run of experiments covers different CNN architectures with different convolutional specifications, varying the number of filters and kernel size.

Top Line Results - Deep Learning Experiments w. (not pre-trained) Embedding Layer		
Model Architecture	**Key Parameters**	**Wght. F1**
Abbreviations: SQL Sequence Length, BS Batchsize, LR Learnrate, CW Classweights, EP Epochs		
CNN & Max Pooling		
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	***.94***

Table 5-8: CNNs – Top Line Classification Results

With these models training much faster than BiLSTMs the experiments are applied with a number of 120 epochs each to get a maximum read on their performance. The first CNN 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 further to .927. Finally building on this a substantial even better result is achieved with an architecture that provides for 2 parallel CNN layers employing two different kernel sizes (5 and 3) and learning 300 filters each. This model yields .94 in weighted F1, outperforming all other experiments by far.

Employing pre-trained embedding vectors might even improve on the CNNs results reported above? Utilizing the pre-trained sources specific embedding matrices are created 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 contained in the embedding matrices used (see chapter 4.3.4).

Top Line Results - Deep Learning Experiments - w. pre-trained Embedding Layer		
Model Architecture	**Key Parameters**	**Wght. F1**
Abbreviations: SQL Sequence Length, BS Batchsize, LR Learnrate, CW Classweights, EP Epochs		
CNN & Max Pooling with pretrained Embeddings		
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=True, 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 similar to the experiments reported in Table 5-8 simply increasing the kernel size (7 instead of 5) doesn't buy more mileage in F1 score.

Surprisingly this pattern of the pre-trained embedding vectors being superior to just using plain embeddings changes when comparing the results of the 2-layered CNNs (with and without pre-trained embeddings). While the experiment using the deepset.AI GloVe vectors achieves the best scores amongst the experiments with pre-trained embeddings it still falls short compared to the performance reported on the 2-layered CNN experiment using only plain embedding vectors (compare with Table 5-8). The impact of the CNN learning multiple patterns at the same time by applying two layers instead of just one contributes stronger to the models' prediction quality than the potential benefit of using pre-trained embedding vectors.

Summarizing this flight with the Deep Learning experiments: With enough tweaking of settings and computation time allowed the application of BiLSTMs can be leveled to achieve results slightly behind the linear models. CNNs easily surpass the linear models (with one exception) in all experiments run. Applying pre-trained vectors doesn't guarantee necessarily better results. The generally promising idea of leveraging pre-trained vectors cannot proof its superiority here. Applying multiple CNN layers in parallel yields (slightly) better results and proofs to handle this noisy data environment even better: The best performing Deep Learning experiment is a 2-layered CNN trained with simple plain embedding vectors.

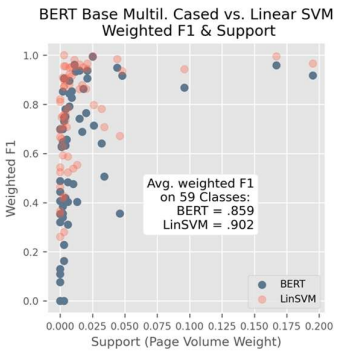
5.3 Transformer Model: BERT

The last flight of experiments covers the BERT Base Multilingual model with self-attention mechanics. The experiments employ the cased and uncased version and both of them are run respectively with class weights applied and without weighing scheme.

Top Line Results - Deep Learning Experiments - BERT Base Multilingual		
Model Architecture	**Key Parameters**	**Wght. F1**
Abbreviations: SQL Sequence Length, BS Batchsize, LR Learnrate, CW Classweights, EP Epochs		
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 better 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).



BERT Multilingual cased: Weighted F1-Scores by Classes					
Top 15			**Low 15**		
	Wg. F1	Support		Wg. F1	Support
BONITAETSENTSCHEIDUNG	.994	.024	ABLOESE_INTERN	.077	.0004
KYC_FORMULAR	.962	.011	ANFRAGE_VORZ_VERTRASBEENDIGUNG	.109	.0003
AUSWEISKOPIE	.959	.170	WIDERRUF_VERTRAG	.130	.001
LEASINGANTRAG	.949	.045	LEASINGANTRAG_INFORMATIONEN	.162	.003
SELBSTAUSKUNFT	.938	.022	SCHADENRUECKINFORMATION	.228	.003
LEASINGANTRAG_ZUSATZPRODUKTE	.937	.014	SCHADENABRECHNUNG	.311	.006
NACHWEIS_BUSINESSKUNDEN	.934	.012	RUECKLAUF_VERTRAGSAENDERUNG	.32	.0004
EINKOMMENSNACHWEIS	.918	.186	SCHADENANZEIGE	.339	.002
UEBERGABEPROTOKOLL_FAHRZEUGUEBER(	.917	.047	ABLOESE_UND_VERSANDVOLLMACHT	.352	.0002
LEASINGANTRAG_KALKULATIONSBLATT	.906	.003	NICHT_ZUORDENBARE_SCHREIBEN	.355	.046
DARLEHENSANTRAG_ZUSATZPRODUKT	.906	.021	SCHADENWERTMINDERUNG	.356	.002
AUSKUNFTEIEN_INFOS	.896	.007	VOLLMACHT	.385	.002
SCHADENGUTACHTEN	.868	.099	ANFRAGE_VERTRAGSAENDERUNG	.402	.006
FAHRZEUGRECHNUNG	.864	.018	ALLGEMEIN_AUTOMOTIVE	.403	.013
FAHRZEUGPAPIERE_KOPIE	.854	.009	VM_KONTOSCHLIESSUNG	.407	.0004

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

The scatterplot in the left panel 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. BERT lags behind on the more important (high support) categories as well as on the classes with less impact. Even on its top performing categories the BERT model falls short when compared to the other models results: The weighted F1 score on "Ausweiskopie" for instance (.959), one of the most important categories (with high support) lags significantly behind the score achieved by the Linear SVM (.994), suggesting that BERT's theoretically superior horsepower in comprehension cannot be utilized in many of the more relevant classes for the given business case, where linear models using plain BOW vectorization seem to do better.

The potential benefit of this widely recognized and success-proven large pre-trained model, the large amount of its parameters (177 Mio.) and the convincing idea of a self-attention transformer mechanic described in chapter 2.4 don't show their often attributed and quoted superiority in this particular document classification problem: The BERT Multilingual cased model scores higher than the uncased version but remains only moderately above the baseline performance. This might be due to the memory constraints mentioned above limiting the sequence lengths of the BERT

experiments to ¾ of the inputs used for the Deep Learning experiments. The influence of the given noisy data might also prevent the BERT architecture to leverage its strength in comprehending semantics 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 the varying flights 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 in detail against the 2-layered CNN (2L-CNN). The 2L-CNN yields a weighted F1 of .94 with a 3.8 percentage points advantage over the LinSVM (.902).

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 values describe classes with superiority of the 2L-CNN and negative values vice versa. The support for each class is indexed on the y-axis. The LinSVM scores better 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 $^{2}/_{3}$ 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.

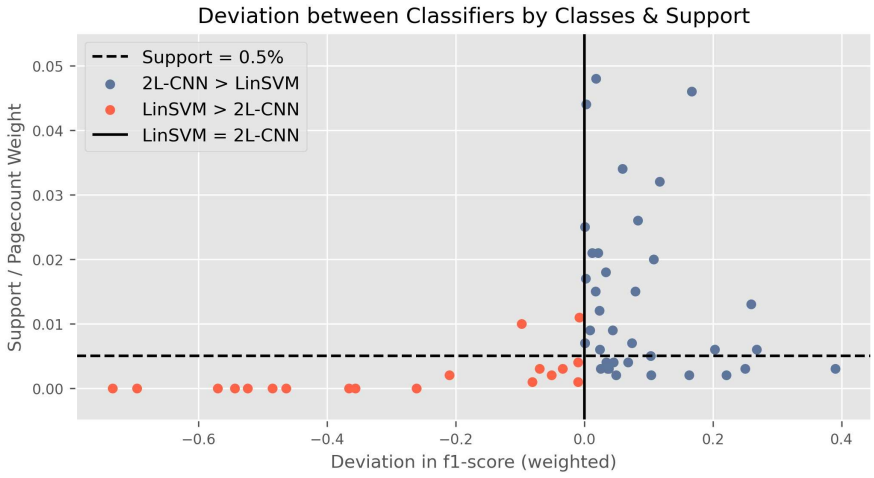


Figure 5-2: Comparison 2-layered CNN and Linear SVM:
Deviation in Classification Performance - Breakdown by Classes & their Support

The dashed line in Figure 5-2 marks a page count weight of .005. Above this threshold only 3 classes[27] are found where the LinSVM is superior to the 2L-CNN. But 29 classes above this threshold account for classes with the 2L-CNN outperforming the LinSVM. This 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) and 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^2 (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^2 scores indeed manifest a positive correlation between the document length and the classifier performance in general. But the two classifiers don't show much different R^2-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 $^2/_3$ 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 contained noisiness of the categories? Chapter 3.3.3 established a proxy to operationalize the amount of noise in the documents for every class. Figure 5-4 is a slight variation of Figure 5-3 with the y-axis now depicting this proxy "Share of Noise" for each class (instead of the page volume weight).

[27] Note that 1 outlier for the LinSVM scoring higher than the 2L-CNN is excluded from the graph.

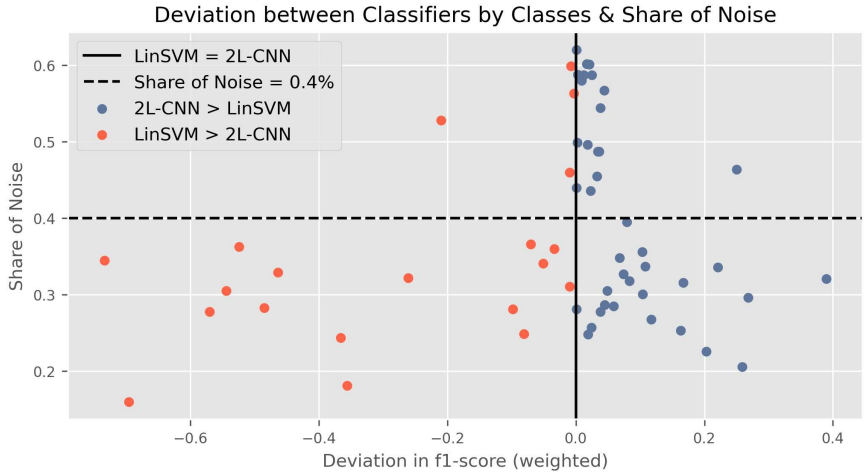


Figure 5-3: Comparison 2-layered CNN and Linear SVM:
Deviation in Classification Performance
Breakdown by Classes & their Share of Noise

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 than the LinSVM in classes with noisier data (above the threshold of 40%). But this advantage scales low. With less noisy data both classifiers can show superiority on different classes and 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.

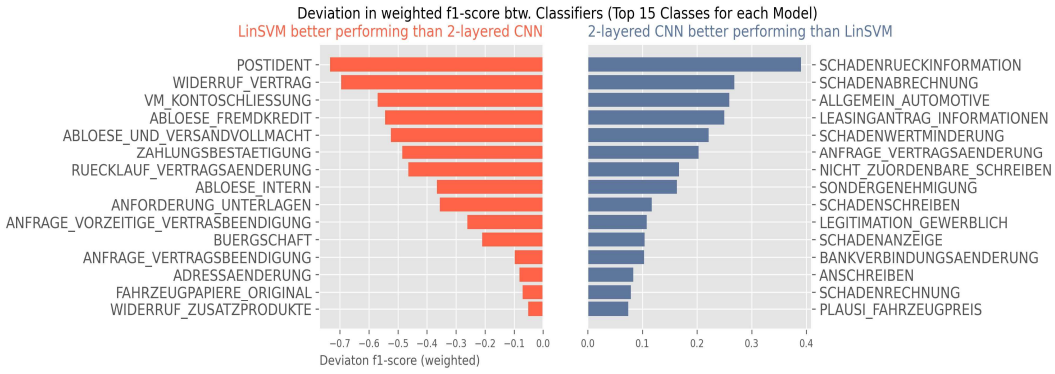


Figure 5-4: Comparison 2-layered CNN and Linear SVM:
Top 15 Classes with Superiority vs. one another

Looking at the 2L-CNNs strengths in the right panel of Figure 5-4 two patterns emerge:

(1) The 2LCNN performs better on classes related to damage management (document types beginning with "SCHADEN" in their label): Amongst the Top 15 six categories deal with damage management of some sort.

(2) The 2LCNN is suited better for generic classes with more heterogeneity of themes and thus less decisive language. Document Types like "ANSCHREIBEN", "ALLGEMEIN_AUTOMOTIVE" and "NICHT_ZUORDENBARE_SCHREIBEN" are handled much better by the 2LCNN than by the LinSVM.

To illustrate those two patterns mentioned above Table 5-13 shows a breakdown on the errors that both models produce on two exemplarily selected classes. The left panel illustrates the mistakes on the category "SCHADENRUECKINFORMATION". Both models make mistakes with their predictions landing on other "SCHADEN"-related labels. But the 2LCNN produces significantly less misclassifications (39% vs. 79%) and distinguishes much better between the false label "SCHADENSCHREIBEN" and the true label than the LinSVM which struggles in particular to distinguish between those two labels.

Misclassifications for selected Classes

SCHADENRUECKINFORMATION
n_{test} = 129 (1.0) - Accuracy*

Class	LinSVM	2LCNN
SCHADENRUECKINFORMATION	***.209***	***.671***
SCHADENSCHREIBEN	.674	.341
SCHADENABRECHNUNG	.070	.023
SCHADENWERTMINDERUNG	.008	.008
ALLGEMEIN_AUTOMOTIVE	.008	--
ANSCHREIBEN	.008	--
SCHADENGUTACHTEN	.008	--
BANKVERBINDUNGSAENDERUNG	.008	--
NICHT_ZUORDENBARE_SCHREIBEN	.008	.008
SCHADENRECHNUNG	--	.008

ALLGEMEIN_AUTOMOTIVE
n_{test} = 344 (1.0) - Accuracy*

Class	LinSVM	2LCNN
ALLGEMEIN_AUTOMOTIVE	.616	.770
NICHT_ZUORDENBARE_SCHREIBEN	.081	.049
ANFRAGE_VERTRAGSAENDERUNG	.070	.026
OTHER	.064	.044
ANFORDERUNG_FAHRZEUGPAPIERE	.044	.009
ANSCHREIBEN	.032	.006
ANFRAGE_VERTRAGSBEENDIGUNG	.029	.084
WIDERRUF_ZUSATZPRODUKTE	.017	--
UMSCHLAG	.012	.006
RUECKLAUF_VERTRAGSAENDERUNG	.009	--
ANFORDERUNG_UNTERLAGEN	.009	--
ADRESSAENDERUNG	.006	--
EINKOMMENSNACHWEIS	.003	--
VOLLMACHT	.003	--
ZAHLUNGSBESTAETIGUNG	.003	--
RUECKNAHMEPROTOKOLL	.003	--
UEBERGABEPROTOKOLL_FAHRZEUGUEBI	--	.003
AUSKUNFTEIEN_INFOS	--	.003

Table 5-13: True Positives & False Positives for selected Classes[28]

The right panel emphasizes the advantage of the 2LCNN on the truly generic class "ALLGEMEIN_AUTOMOTIVE". A category collecting all sorts of different documents which

[28] 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.

are related broadly to automotive intents and don't fit for a dedicated more specific category. While the LinSVM spreads its misclassifications here across a range of 15 labels, the 2L-CNN keeps this variance at bay selecting from only 9 different false labels and shows its superiority with a significantly higher overall accuracy (.77) than the LinSVM (.616).

5.5 Interim Conclusion on the three Approaches

With all three experimental flights reported and discussed in detail Table 5-14 provides a brief summary and interim conclusion of the results so far:

Interim Summary of the experimental Approaches		
I Machine Learning Models w. BOW/TF-IDF Vectorization	II Deep Learning Models w. Embedding Layers	III Transfer Learning & Transformer Networks
Models Multinomial Naïve Bayes k-NN Classifier Logistic Regression Classifier *Linear Support Vector Machine* Stochastic Gradient Descent SVM w. Kernel Functions Gradient Boosting	**Architectures** Bidirectional LSTM Convolutional Neural Net *CNN w. two convolutional Layers* CNN w. pre-trained Embeddings	**Model** *BERT Base Multilingual Cased* BERT Base Multilingual Uncased
Learnings Linear Models achieve superior performance (especially LinSVM) yielding the highest scores within this group. Provided with BOW/TF-IDF feature space they can find distinctive expressions and linear combinations thereof to handle most classes well. The lag with generic categories and with categories sharing a common theme (i.e. damage). Generally linear models deal well with the high level of noise in the data.	**Learnings** CNNs show convincing, superior results across all experiments. Using pre-trained embedding vectors improves the results but applying multiple convolutional layers is even more instrumental and shows the biggest impact on the results with a 2-lay. CNN achieving the highest score of all. Learning parallel patterns through multiple convolutions improves results on interrelated and on more generic classes giving superiority over linear models.	**Learnings** The expected advantage of large pretraining and the multi-headed self attention concept does not show in this use case. BERTs' conceptional semantic power goes unrecognized in the given document classification task: Despite their simplicity other models (i.e. the LinSVM see left panel) can handle the noisy data much better with a much smaller computational footprint.

Table 5-14: Interim Summary of the different Approaches

As laid out in chapter 5.4 the Deep Learning approach using CNNs and the linear models with BOW/TF-IDF show different strengths and weaknesses when applied to the documents in this use case. Those models can learn different aspects of the underlying data. Hence a final experiment is applied to synthesize those individual advantages of the two more promising approaches into one final and potentially superior solution.

5.6 Ensemble Estimator: Best of all Worlds

Consequently those individual strengths of each can be synthesized into an ensemble classifier, combining the best of both worlds approach (omitting the not convincing Transformer model approach). Instead of a voting scheme this ensemble estimator could use the probability distributions produced by the single classifiers in the ensemble team. For each instance the 2LCNN outputs an array of probabilities over the 59 different classes. Because the LinSVM doesn't generate a probability distribution like the 2LCNN, it is exchanged in this final experiment with the Logistic Regression Classifier (LogReg) trained before. It's also 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 thus takes the probabilistic prediction output vectors of the 2L-CNN and the LogReg model for input, compares them with each other and simply chooses the prediction with the highest confidence, the class with the highest probability assigned. Applied to N_{test} this ensemble estimator yields the results reported in Table 5-15:

Classification Performance: Linear Models vs. 2LCNN vs. Ensemble				
Weighted f1-score on all Classes & selected Classes				
Class	LinSVM	LogReg	2LCNN	Ensemble
TOTAL (59 Classes)	***.902***	***.889***	***.94***	***.948***
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 5-15: Overview Classifier Performance with Ensemble Classifier

Across all 59 classes applying the Ensemble estimator yields a weighted F1 score of .948, surpassing the so best performing single classifier, the 2LCNN (.94). Looking at exemplarily selected classes shows that 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 5-15 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.

Cherry picking pays back: The ensemble estimator performs superior to all experiments applying single models or ideas of the Deep Learning universe. With all technical complexity put aside the ensemble estimator provides a successful synthesis of individual strengths to tackle the noisy data environment in this document classification project.

6 Final Conclusions

This project employs 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 particularly noisy data environment: Documents arrive in a big variety of technical formats and lengths with external OCR and other preprocessing steps employed beforehand. Those add significant levels of noise to the data. To operationalize the contained "Share of Noise" in the documents a proxy has been established. Common public sources of German language are benchmarked against the training corpus and the amount of non-matches is used to indicate the varying noise level of the different document types (classes). With additional compliance requirements enforced 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 this rather traditional approach can deliver substantial results regardless of the high share of noise in the data. For some classes with highly descriptive terms and language the linear models yield 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 this use case.

The third group of experiments covers the application of the Transformer architecture. BERT, a pioneering and widely recognized implementation of this idea is put to the test. But the experiments don't show the theoretical strength and superiority one would expect. 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 to be recognized by each approach in the data. Consequently the strengths of these two approaches can be forged into one ensemble estimator exploiting the properties of these approaches. A combination of a 2-layered CNN and a Logistic Regression Classifier 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 (MVP) within this project: Given the complexity 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[29] for an initial first step implementation. Further ideas should be developed thereafter to replace the plain linear models (see chapter 7 for further possible improvements). At time of writing the Linear SVM is actively deployed into a secured BMW Bank cloud environment and is fully functionally serving a REST-API web service that can be connected to the production system should BMW Bank choose to do so.

[29] Logistic Regression Classifiers can produce a probability distribution for each prediction. This could be an important additional property for the subsequent processing of the inference result within the DMS logic.

7 Limitations & possible Improvements

Several limitations are observed during the experiments providing room for further development and improvements in the future:

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. In this project only limited effort is applied to de-noise the text with specific preprocessing measures. The quality of the input data thus could be further improved with more thorough data cleaning to filter and transform noise. Additional steps would be the development of more regex-based preprocessing routines or the application of language normalization techniques like stemming, lemmatization and stop-word-filtering.

Anonymization: The very sensitive nature of the data enforces strict data protection and information security measures to the raw data (see chapter 3.2). One key element of the anonymization procedure is the application of a pre-trained SpaCy German Language model to identify and replace names of people, organizations and locations with a generic label. The SpaCy authors report an F1 of .842 on this NER-task[30]. While chosen as a much better method than just anonymizing with regex or lookup rules the as-is provided NER 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, pre-training of the language model used to improve and tailor the NER functionality.

Feature selection: All classifiers are exposed to the preprocessed and vectorized textual input plus the categorical and numerical features provided within the metadata 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 for the use case 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 better parametrizations 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

[30] See documentation: https://github.com/explosion/spacy-models/releases//tag/de_core_news_md-3.0.0

compare their performance. But deep neural nets allow for very complex architectures with many strategic choices: 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 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[31]. 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 derived 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 contribute much better to the training with better classification results 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 observed memory limitation on the DLP, limiting the potential capacity of this model. In addition the applied BERT Base 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.

Ensemble: Chapter 5.6 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

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

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 a team of more specialized classifiers trained on individual batch classes. The application of the individual model would then be triggered by the meta 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.

.

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 Networs Cost Functions.* [Online]

Available at: 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, Masachusetts: 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.