Chapter 10
Natural Language Processing

Natural language processing (NLP) is a vastly growing field that is studying language, communication and text recognition. The purpose of this chapter is to present an introduction to NLP. Important milestones in the field of NLP are the work of Bengio et al. [28, 29] who have introduced the idea of word embedding, the work of Mikolov et al. [275, 276] who have developed word2vec which is an efficient word embedding tool, and the work of Pennington et al. [300] and Chaubard et al. [68] who provide the pre-trained word embedding model GloVe[1] and detailed educational material.[2] An excellent overview of the NLP working pipeline is provided by the tutorial of Ferrario–Nägelin [126]. This overview distinguishes three approaches: (1) the classical approach using bag-of-words and bag-of-part-of-speech models to classify text documents; (2) the modern approach using word embeddings to receive a low-dimensional representation of the dictionary, which is then further processed; (3) the contemporary approach uses a minimal amount of text pre-processing but directly feeds raw data to a machine learning algorithm. We discuss these different approaches and show how they can be used to extract the relevant information from claim descriptions to predict the claim types and the claim sizes; in the actuarial literature first papers on this topic have been published by Lee et al. [236] and Manski et al. [264].

10.1 Feature Pre-processing and Bag-of-Words

NLP requires an extensive feature pre-processing and engineering as different texts can be rather diverse in language, grammar, abbreviations, typos, etc. The current developments aim at automating this process, nevertheless, many of these steps

[1] https://nlp.stanford.edu/projects/glove/.

[2] https://nlp.stanford.edu/teaching/.

M. V. Wüthrich, M. Merz, *Statistical Foundations of Actuarial Learning and its Applications*, Springer Actuarial, https://doi.org/10.1007/978-3-031-12409-9_10

are still (tedious) manual work. Our goal here is to present the whole working pipeline to process language, perform text recognition and text understanding. As an example we use the claim data described in Chap. 13.3; this data has been made available through the book project of Frees [135], and it comprises property claims of governmental institutions in Wisconsin, US. An excerpt of the data is given in Listing 10.1; our attention applies to line 11 which provides a (very) short claim description for every claim.

Listing 10.1 Excerpt of the Wisconsin Local Government Property Insurance Fund (LGPIF) data set with short claim descriptions on line 11

```
1  'data.frame':   5424 obs. of  10 variables:
2   $ PolicyNum   : int 120002 120003 120003 120003 120003 120003 120003 ...
3   $ Year        : int 2010 2007 2008 2007 2009 2010 2007 2007 2009 2007 ...
4   $ Claim       : num 6839 2085 8775 600 34610 ...
5   $ Deduct      : int 1000 5000 5000 5000 5000 5000 5000 5000 5000 5000 ...
6   $ EntityType  : Factor w/ 6 levels "City","County",..: 2 2 2 2 2 2 2 2 2 2 ...
7   $ CoverageCode: Factor w/ 13 levels "CE","CF","CS",..: 12 12 11 11 11 12 ...
8   $ Fire5       : int 4 0 0 0 0 0 0 0 0 0 ...
9   $ CountyCode  : Factor w/ 72 levels "ADA","ASH","BAR",..: 2 3 3 3 3 3 3 3...
10  $ Hazard      : Factor w/ 9 levels "Fire","Hail",..: 3 3 5 5 9 6 3 3 3 3 ...
11  $ Description : chr  "lightning damage" "lightning damage at Comm. Center" ...
```

In a first step we need to pre-process the texts to make them suitable for predictive modeling. This first step is called *tokenization*. Essentially, tokenization labels the words with integers, that is, the used vocabulary is encoded by integers. There are several issues that one has to deal with in this first step such as upper and lower case, punctuation, orthographic errors and differences, abbreviations, etc. Different treatments of these issues will lead to different results, for more on this topic we refer to Sect. 1 in Ferrario–Nägelin [126]. We simply use the standard routine offered in R `keras` [77] called `text_tokenizer()` with its standard settings.

Listing 10.2 Tokenization within R `keras` [77]

```
1  library(keras)
2
3  ## initialize tokenizer and fit
4  tokenizer <- text_tokenizer() %>% fit_text_tokenizer(dat$Description)
5
6  ## number of tokens/words
7  length(tokenizer$word_index)
8
9  ## frequency of word appearances in each text
10 freq.text <- texts_to_matrix(tokenizer, dat$Description, mode = "count")
```

The R code in Listing 10.2 shows the crucial steps in tokenization. Line 4 extracts the relevant vocabulary from all available claim descriptions. In total the 5'424 claim

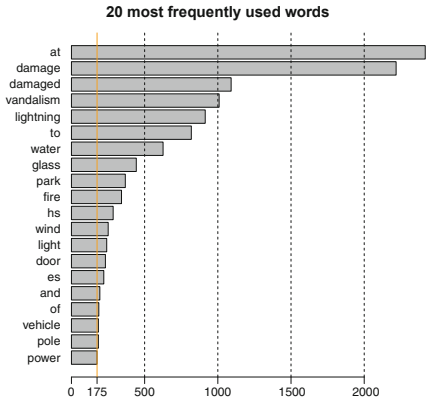

Fig. 10.1 Most frequently used words in the claim descriptions of Listing 10.1

descriptions of Listing 10.1 use $W = 2'237$ different words. This double counts different spellings, e.g., 'color' vs. 'colour'.

Figure 10.1 shows the most frequently used words in the claim descriptions of Listing 10.1. These are (in this order): 'at', 'damage', 'damaged', 'vandalism', 'lightning', 'to', 'water', 'glass', 'park', 'fire', 'hs', 'wind', 'light', 'door', 'es', 'and', 'of', 'vehicle', 'pole' and 'power'. We observe that many of these words are directly related to insurance claims, such as 'damage' and 'vandalism', others are frequent *stopwords* like 'at' and 'to', and then there are abbreviations like 'hs' and 'es' standing for high school and elementary school.

Listing 10.3 Word and text encoding

```
1  maxlen <- max(rowSums(freq.text))
2
3  ## encode the sentences
4  text.seq <- texts_to_sequences(tokenizer, dat$Description)
5
6  ## pad the sentences
7  text.seq.pad <- pad_sequences(text.seq, maxlen = maxlen, padding = "post")
8
9  ## examples
10 lightning/hail damage to equip at airport
11   5  48   2   6 196   1  40   0   0   0   0
12 ##
13 garage door damaged
14  36 14  3  0  0  0  0  0  0  0  0
```

The next step is to assign the (integer) labels $1 \leq w \leq W$ from the tokenization to the words in the texts. The maximal length over all texts/sentences is $T = 11$ words. This step and padding the sentences with zeros to equal length T is presented on lines 1–7 of Listing 10.3. Lines 11 and 14 of this listing give two explicit text examples

$$\texttt{text} = (w_1, \ldots, w_T)^\top \in \mathcal{W}_0^T,$$

where we set for the vocabulary $\mathcal{W}_0$ used

$$\mathcal{W} = \{1, \ldots, W\} \subset \mathbb{N} \qquad \text{and} \qquad \mathcal{W}_0 = \mathcal{W} \cup \{0\}.$$

The label 0 is used for padding shorter texts to the common length $T = 11$. The method of *bag-of-words* embeds $\texttt{text} = (w_1, \ldots, w_T)^\top$ into $\mathbb{N}_0^W$

$$\psi : \mathcal{W}_0^T \to \mathbb{N}_0^W, \qquad \texttt{text} \mapsto \psi(\texttt{text}) = \left(\sum_{t=1}^{T} \mathbb{1}_{\{w_t = w\}} \right)_{w \in \mathcal{W}}^{\top}. \tag{10.1}$$

The bag-of-words $\psi(\texttt{text})$ counts how often each word $w \in \mathcal{W}$ appears in a given $\texttt{text} = (w_1, \ldots, w_T)^\top$; the corresponding code is given on line 10 of Listing 10.2. The bag-of-words mapping ψ is not injective as the order of occurrence of the words gets lost, and, thus, also the semantics of the sentence gets lost. E.g., the bag-of-words of the following two sentences is the same 'The claim is expensive.' and 'Is the claim expensive?'. This is the reason for calling it a "bag of words" (which is unordered). This bag-of-words encoding resembles one-hot encoding, namely, if every text consists of a single word $T = 1$, then we receive the one-hot encoding with W describing the number of different levels, see (7.28). The bag-of-words $\psi(\texttt{text}) \in \mathbb{N}_0^W$ can directly be used as an input to a regression model. The disadvantage of this approach is that the input typically is high-dimensional (and likely sparse), and it is recommended that only the frequent words are considered.

Listing 10.4 Removal of stopwords and lemmatization

```
1 library(textstem)
2 library(tm)
3
4 text.clean <- removeWords(dat$Description, stopwords("english"))
5 text.clean <- lemmatize_strings(text.clean, dictionary = lexicon::hash_lemmas)
```

Additionally, stopwords can be removed. We perform this removal below because frequent stopwords like 'and' or 'to' may not essentially contribute to the understanding of the (short) claim descriptions; the code for the stopword removal is provided on line 4 of Listing 10.4. Moreover, stemming can be performed which means that inflectional forms are reduced to their stem by just truncating pre- and suffixes, conjugations, declensions, etc. Lemmatization is a more sophisticated form of reducing inflectional forms by using vocabularies and morphological analyses; an example is provided on line 5 of Listing 10.4. If we perform these two steps of removing stopwords and lemmatization to our example, the number of different words is reduced from 2'237 to 1'982.

Another step that can be performed is tagging words with part-of-speech (POS) attributes. These POS attributes indicate whether the corresponding words are used

as nouns, adjectives, adverbs, etc., in the corresponding sentences. We then call the resulting encoding bag-of-POS. We refrain from doing this because we will present more sophisticated methods in the next sections.

10.2 Word Embeddings

The bag-of-words (10.1) can be interpreted as representing each word $w \in \mathcal{W} = \{1, \ldots, W\}$ by a one-hot encoding in $\{0, 1\}^W$, and then aggregating these one-hot encodings over all words that appear in the given $\texttt{text} = (w_1, \ldots, w_T)^\top$. Bengio et al. [28, 29] have introduced the technique of *word embedding* that maps words to a lower dimensional Euclidean space $\mathbb{R}^b$, $b \ll W$, such that proximity in $\mathbb{R}^b$ is associated with similarity in the meaning of the word, e.g., 'rain', 'water' and 'flood' should be more close to each other in $\mathbb{R}^b$ than to 'vandalism' (in an insurance context). This is exactly the idea promoted in the embedding mapping (7.31) using the embedding layers. Thus, we are looking for an embedding mapping

$$\boldsymbol{e} : \mathcal{W} \to \mathbb{R}^b, \qquad w \mapsto \boldsymbol{e}(w), \tag{10.2}$$

that maps each word w (or rather its tokenization) to a b-dimensional vector $\boldsymbol{e}(w)$, for a given embedding dimension $b \ll W$. The general idea now is that similarity in the meaning of words can be learned from the context in which the words are used in. That is, when we consider a text

$$\texttt{text} = (w_1, \ldots, w_{t-1}, w_t, w_{t+1}, \ldots, w_T)^\top,$$

then it might be possible to infer w_t from its neighbors w_{t-j} and w_{t+j}, $j \geq 1$. This explains the context of a word w_t, and using suitable learning tools it should also be possible to learn synonyms for w_t as these synonyms will stand in similar contexts.

More mathematically speaking, we assume that there exists a probability distribution p over the set of all texts of length T (using padding with zeros to common length)

$$\mathsf{T} = \left\{ \texttt{text} = (w_1, \ldots, w_T)^\top \right\} \subseteq \mathcal{W}_0^T,$$

such that a randomly chosen $\texttt{text} \in \mathsf{T}$ appears with probability $p(w_1, \ldots, w_T) \in [0, 1)$. Inference of a word w_t from its context can then be obtained by studying the conditional probablity of w_t, given its context, that is

$$p\left(w_t \middle| w_1, \ldots, w_{t-1}, w_{t+1}, \ldots, w_T\right) = \frac{p(w_1, \ldots, w_T)}{p(w_1, \ldots, w_{t-1}, w_{t+1}, \ldots, w_T)}. \tag{10.3}$$

Since, typically, the probability distribution p is not known we aim at learning it from the available data. This idea has been taken up by Mikolov et al. [275, 276] who designed the word to vector (word2vec) algorithm. Pennington et al. [300] designed an alternative algorithm called global vectors (GloVe); we also refer to Chaubard et al. [68]. We describe these algorithms in the following sections.

10.2.1 Word to Vector Algorithms

There are two ways of estimating the probability p in (10.3). Either we can try to predict the *center word* w_t from its context as in (10.3) or we can try to predict the context from the center word w_t, which applies Bayes's rule to (10.3). The latter variant is called *skip-gram* and the former variant is called *continuous bag-of-words* (CBOW), if we neglect the order of the words in the context. These two approaches have been developed by Mikolov et al. [275, 276].

Skip-gram Approach

Typically, inferring a general probability distribution p over T is too complex. Therefore, we make a simplifying assumption. This simplifying assumption is not reasonable from a practical linguistic point of view, but it is sufficient to receive a reasonable word embedding map $\boldsymbol{e} : \mathcal{W} \to \mathbb{R}^b$. We assume conditional i.i.d. of the context words, given the center word w_t. Choosing a fixed context (window) size $c \in \mathbb{N}$, we try to maximize the log-likelihood over all probabilities p satisfying this conditional i.i.d. assumption

$$\begin{aligned} \ell_{\boldsymbol{W}} &= \sum_{i=1}^{n} \log p\left(w_{i,t-c}, \ldots, w_{i,t-1}, w_{i,t+1}, \ldots, w_{i,t+c} \,\middle|\, w_{i,t}\right) \\ &= \sum_{i=1}^{n} \sum_{-c \le j \le c, j \neq 0} \log p\left(w_{i,t+j} \,\middle|\, w_{i,t}\right), \end{aligned} \tag{10.4}$$

having n independent rows in the observed data matrix $\boldsymbol{W} = (w_{i,t-c}, \ldots, w_{i,t+c})_{1 \le i \le n} \in \mathcal{W}^{n \times (2c+1)}$. Thus, under the conditional i.i.d. of the context words, given the center word, the probabilities (10.4) infer the occurrence of (individual) context words of a given center word $w_{i,t}$ within a symmetric window of fixed size c. In the sequel we directly work with the log-likelihood (10.4), supposed that a context word $w_{i,t+j}$ exists for index j, otherwise the corresponding term is just dropped from the sum in (10.4).

The remaining step is to estimate the conditional probabilities $p(w_{t+j}|w_t)$ from the data matrix $\boldsymbol{W}$. This step will provide us with the embeddings (10.2). This estimation step is received by considering an approach similar to a GLM for

categorical responses, see Sect. 5.7. We make the following ansatz for the context word w_s and the center word w_t (for all j)

$$p\,(w_s|\,w_t) = \frac{\exp\langle \widetilde{\boldsymbol{e}}(w_s), \boldsymbol{e}(w_t)\rangle}{\sum_{w=1}^{W} \exp\langle \widetilde{\boldsymbol{e}}(w), \boldsymbol{e}(w_t)\rangle} \;\in\; (0,1), \tag{10.5}$$

where $\boldsymbol{e}$ and $\widetilde{\boldsymbol{e}}$ are two (different) embedding maps (10.2) that have the same embedding dimension $b \in \mathbb{N}$. Thus, we construct two different embeddings $\boldsymbol{e}$ and $\widetilde{\boldsymbol{e}}$ for the center words and for the context words, respectively, and these embeddings (embedding weights) are chosen such that the log-likelihood (10.4) is maximized for the given observations $\boldsymbol{W}$. These assumptions give us a minimization problem for the negative log-likelihood in the embedding mappings, i.e., we minimize over the embeddings $\boldsymbol{e}$ and $\widetilde{\boldsymbol{e}}$

$$\begin{aligned} -\ell_{\boldsymbol{W}} &= -\sum_{i=1}^{n} \sum_{-c\le j\le c,\, j\neq 0} \log\left(\frac{\exp\langle \widetilde{\boldsymbol{e}}(w_{i,t+j}), \boldsymbol{e}(w_{i,t})\rangle}{\sum_{w=1}^{W} \exp\langle \widetilde{\boldsymbol{e}}(w), \boldsymbol{e}(w_{i,t})\rangle}\right) \\ &= -\sum_{i=1}^{n} \left(\sum_{-c\le j\le c,\, j\neq 0} \langle \widetilde{\boldsymbol{e}}(w_{i,t+j}), \boldsymbol{e}(w_{i,t})\rangle - 2c \log\left(\sum_{w=1}^{W} \exp\langle \widetilde{\boldsymbol{e}}(w), \boldsymbol{e}(w_{i,t})\rangle\right)\right). \end{aligned} \tag{10.6}$$

These optimal embeddings are learned using a variant of the gradient descent algorithm. This often results in a very high-dimensional optimization problem as we have $2bW$ parameters to learn, and the calculation of the last (normalization) term in (10.6) can be very expensive in gradient descent algorithms. For this reason we present the method of negative sampling below.

Continuous Bag-of-Words

For the CBOW method we start from the log-likelihood for a context size $c \in \mathbb{N}$ and given the observations $\boldsymbol{W}$

$$\sum_{i=1}^{n} \log p\left(w_{i,t}\,\middle|\, w_{i,t-c}, \ldots, w_{i,t-1}, w_{i,t+1}, \ldots, w_{i,t+c}\right).$$

Again we need to reduce the complexity which requires an approximation to the above. Assume that the embedding map of the context words is given by $\widetilde{\boldsymbol{e}} : \mathcal{W} \to \mathbb{R}^b$. We then average over the embeddings of the context words in order to predict the center word. Define the average embedding of the context words of $w_{i,t}$ (with a fixed window size c) by

$$\widetilde{e}_{i,t} = \frac{1}{2c} \sum_{-c\le j\le c,\, j\neq 0} \widetilde{\boldsymbol{e}}(w_{i,t+j}).$$

Making an ansatz similar to (10.5), the full log-likelihood is approximated by

$$\sum_{i=1}^{n} \log p\left(w_{i,t} \middle| \widetilde{e}_{i,t}\right) = \sum_{i=1}^{n} \log\left(\frac{\exp\left\langle \widetilde{e}_{i,t}, \boldsymbol{e}(w_{i,t})\right\rangle}{\sum_{w=1}^{W} \exp\left\langle \widetilde{e}_{i,t}, \boldsymbol{e}(w)\right\rangle}\right) \tag{10.7}$$
$$= \sum_{i=1}^{n} \left\langle \widetilde{e}_{i,t}, \boldsymbol{e}(w_{i,t})\right\rangle - \log\left(\sum_{w=1}^{W} \exp\left\langle \widetilde{e}_{i,t}, \boldsymbol{e}(w)\right\rangle\right).$$

Again the gradient descent method is applied to the negative log-likelihood to learn the optimal embedding maps $\boldsymbol{e}$ and $\widetilde{\boldsymbol{e}}$.

Remark 10.1 In both cases, skip-gram and CBOW, we estimate two separate embeddings $\boldsymbol{e}$ and $\widetilde{\boldsymbol{e}}$ for the center word and the context words. Typically, CBOW is faster but skip-gram is better on words that are less frequent.

Negative Sampling

There is a computational issue in (10.6) and (10.7) because the probability normalizations in (10.6) and (10.7) aggregate over all available words $w \in \mathcal{W}$. This can be computationally demanding because we need to perform this calculation in each gradient descent step. For this reason, Mikolov et al. [276] turn the log-likelihood optimization problem (10.6) into a binary classification problem. Consider a pair $(w, \widetilde{w}) \in \mathcal{W} \times \mathcal{W}$ of center word w and context word $\widetilde{w}$. We introduce a binary response variable $Y \in \{1, 0\}$ that indicates whether an observation $(W, \widetilde{W}) = (w, \widetilde{w})$ is coming from a true center-context pair (from our texts) or whether we have a fake center-context pair (that has been generated randomly). Choosing the canonical link of the Bernoulli EF (logistic/sigmoid function) we make the following ansatz (in the skip-gram approach) to test for the authenticity of a center-context pair $(w, \widetilde{w})$

$$\mathbb{P}[Y = 1 | w, \widetilde{w}] = \frac{1}{1 + \exp\{-\langle \widetilde{\boldsymbol{e}}(\widetilde{w}), \boldsymbol{e}(w)\rangle\}}. \tag{10.8}$$

The recipe now is as follows: (1) Consider for a given window size c all center-context pairs $(w_i, \widetilde{w}_i) \in \mathcal{W} \times \mathcal{W}$ of our texts, and equip them with a response $Y_i = 1$. Assume we have N such observations. (2) Simulate N i.i.d. pairs $(W_{N+k}, \widetilde{W}_{N+k})$, $1 \le k \le N$, by randomly choosing W_{N+k} and $\widetilde{W}_{N+k}$, independent from each other (by performing independent re-sampling with or without replacements from the data $(w_i)_{1 \le i \le N}$ and $(\widetilde{w}_i)_{1 \le i \le N}$, respectively). Equip these (false) pairs with the response $Y_{N+k} = 0$. (3) Maximize the following log-likelihood as a function of the

embedding maps $\boldsymbol{e}$ and $\widetilde{\boldsymbol{e}}$

$$\ell_{\boldsymbol{Y}} = \sum_{i=1}^{2N} \log \mathbb{P}[\, Y = Y_i \,|\, w_i, \widetilde{w}_i \,] \tag{10.9}$$
$$= \sum_{i=1}^{N} \log\left(\frac{1}{1+\exp\langle -\widetilde{\boldsymbol{e}}(\widetilde{w}_i), \boldsymbol{e}(w_i)\rangle}\right) + \sum_{k=N+1}^{2N} \log\left(\frac{1}{1+\exp\langle \widetilde{\boldsymbol{e}}(\widetilde{w}_k), \boldsymbol{e}(w_k)\rangle}\right).$$

This approach is called *negative sampling* because we sample false or negative pairs $(W_{N+k}, \widetilde{W}_{N+k})$ that should not appear in our texts (as W_{N+k} and $\widetilde{W}_{N+k}$ have been generated independently from each other). The binary classification (10.9) aims at detecting the negative pairs be letting the scalar products $\langle \widetilde{\boldsymbol{e}}(\widetilde{w}_i), \boldsymbol{e}(w_i)\rangle$ be large for the true pairs and letting the scalar products $\langle \widetilde{\boldsymbol{e}}(\widetilde{w}_k), \boldsymbol{e}(w_k)\rangle$ be small for the false pairs. The former means that $\widetilde{\boldsymbol{e}}(\widetilde{w}_i)$ and $\boldsymbol{e}(w_i)$ should point into the same direction in the embedding space $\mathbb{R}^b$. The same should apply for a synonym of w_i and, thus, we receive the desired behavior that synonyms or words with similar meanings tend to cluster.

Example 10.2 (word2vec with Negative Sampling) We provide an example by constructing a word2vec embedding based on negative sampling. For this we aim at maximizing the log-likelihood (10.9) by finding optimal embedding maps $\boldsymbol{e}$ and $\widetilde{\boldsymbol{e}} : \mathcal{W} \to \mathbb{R}^b$. To construct these embedding maps we use the Wisconsin LGPIF data described in Sect. 13.3. The first decision (hyper-parameter) is the choice of the embedding dimension b. English language has millions of different words, and these words should be (in some sense) densely embedded into a b-dimensional Euclidean space. Typical choices of b vary between 50 and 300. Our LGPIF data vocabulary is much smaller, and for this example we choose $b = 2$ because this allows us to nicely illustrate the learned embeddings. However, apart from illustration, we should not choose such a small dimension as it does not allow for a sufficient flexibility in discriminating the words, as we will see.

We consider all available claim texts described in Sect. 13.3. These are 6'031 texts coming from the training and validation data sets (we include the validation data here to have more texts for learning the embeddings; this is different from Sect. 10.1). We extract the claim descriptions from these two data sets and we apply some pre-processing to the texts. This involves transforming all letters to lower case, removing the special characters like !"/&, and removing the stopwords. Moreover, we remove the words 'damage' and 'damaged' as these two words are very common in our insurance claim descriptions, see Fig. 10.1, but they do not further specify the claim type. Then we apply lemmatization, see Listing 10.4, and we adjust the vocabulary with the GloVe database,[3] see also Example 10.4. The latter step is

[3] https://nlp.stanford.edu/projects/glove/.

(tedious) manual work, and we do this step to be able to compare our results to pre-trained word2vec versions.

After this pre-processing we apply the tokenizer, see line 4 of Listing 10.2. This gives us $1'829$ different words. To construct our (illustrative) embedding we only consider the words that appear at least 20 times over all texts, these are $W = 142$ words. Thus, the following analysis is only based on the $W = 142$ most frequent words. Of course, we could increase our vocabulary by considering any text that can be downloaded from the internet. Since we would like to perform an insurance claim analysis, these texts should be related to an insurance context so that the learned embeddings reflect an insurance experience; we come back to this in Remark 10.4, below. We refrain here from doing so and embed these $W = 142$ words into the Euclidean plane ($b = 2$).

Listing 10.5 Tokenization of the most frequent words

```
1  ## applying the tokenizer to the cleaned texts
2  tokenizer <- text_tokenizer(num_words=142+1) %>% fit_text_tokenizer(dat$clean)
3
4  seqs <- texts_to_sequences(tokenizer, dat$clean)
5
6  ## skip-gram of text 1 using a window of size 2
7  skipgrams(sequence=unlist(seqs[[1]]),
8                      vocabulary_size=142, window_size=2, negative_samples=0)
```

Listing 10.5 shows the tokenization of the most frequent words, and on line 4 we build the (shortened) texts $w_1, w_2, \ldots$, only considering these most frequent words $w \in \mathcal{W} = \{1, \ldots, W\}$. In total we receive $4'746$ texts that contain at least two words from $\mathcal{W}$ and, hence, can be used for the skip-gram building of center-context pairs $(w, \widetilde{w}) \in \mathcal{W} \times \mathcal{W}$. Lines 7–8 give the code for building these pairs for a window of size $c = 2$. In total we receive $N = 23'952$ center-context pairs $(w_i, \widetilde{w}_i)$ from our texts. We equip these pairs with a response $Y_i = 1$. For the false pairs, we randomly permute the second component of the true pairs $(W_{N+i}, \widetilde{W}_{N+i}) = (w_i, \widetilde{w}_{\tau(i)})$, where τ is a random permutation of $\{1, \ldots, N\}$. These false pairs are equipped with a response $Y_{N+i} = 0$. Thus, altogether we have $2N = 47'904$ observations $(Y_i, w_i, \widetilde{w}_i)$, $1 \le j \le 2N$, that can be used to learn the embeddings $\boldsymbol{e}$ and $\widetilde{\boldsymbol{e}}$.
Listing 10.6 shows the R code to perform the embedding learning using the negative sampling (10.9). This network has $2bW = 568$ embedding weights that need to be learned from the data. There are two more parameters involved on line 10 of Listing 10.6. These two parameters shift the scalar products by an intercept β_0 and scale them by a constant β_1. We could set $(\beta_0, \beta_1) = (0, 1)$, however, keeping these two parameters trainable has led to results that are better centered around the origin. Of course, these two parameters do not harm the arguments as they only

Listing 10.6 R code for negative sampling

```
center  = layer_input(shape = c(1), dtype = 'int32')
context = layer_input(shape = c(1), dtype = 'int32')
#
centerEmb = center %>%
     layer_embedding(input_dim=142,output_dim=2,input_length=1) %>% layer_flatten()
contextEmb = context %>%
     layer_embedding(input_dim=142,output_dim=2,input_length=1) %>% layer_flatten()
#
response = list(centerEmb, contextEmb) %>% layer_dot(axes = 1) %>%
                layer_dense(units=1, activation='sigmoid', name='response')
#
model = keras_model(inputs = c(center, context), outputs = c(response))
```

replace (10.8) by a slightly different model

$$\mathbb{P}[Y=1|\,w,\widetilde{w}] = \frac{1}{1+\exp\{-\beta_0-\beta_1\langle\widetilde{\boldsymbol{e}}(\widetilde{w}),\boldsymbol{e}(w)\rangle\}} = \frac{e^{\beta_0}}{e^{\beta_0}+e^{-\beta_1\langle\widetilde{\boldsymbol{e}}(\widetilde{w}),\boldsymbol{e}(w)\rangle}},$$

and

$$\mathbb{P}[Y=0|\,w,\widetilde{w}] = 1-\frac{e^{\beta_0}}{e^{\beta_0}+e^{-\beta_1\langle\widetilde{\boldsymbol{e}}(\widetilde{w}),\boldsymbol{e}(w)\rangle}} = \frac{e^{-\beta_0}}{e^{-\beta_0}+e^{\beta_1\langle\widetilde{\boldsymbol{e}}(\widetilde{w}),\boldsymbol{e}(w)\rangle}}.$$

We fit this model using the `nadam` version of the gradient descent algorithm, and the fitted embedding weights can be extracted with `get_weights(model)`. Figure 10.2 shows the learned embedding weights $\boldsymbol{e}(w)\in\mathbb{R}^2$ of all words $w\in\mathcal{W}$. We highlight the words that coincide with the insured hazards in red color, see line 10 of Listing 10.1. The word 'vehicle' is in the first quadrant and it is surrounded by 'pole', 'truck', 'garage', 'car', 'traffic'. The word 'vandalism' is in the third quadrant surrounded by 'graffito', 'window', 'pavilion', names of cites and parks, 'ms' for middle school. Finally, the words 'fire', 'wind', 'lightning' and 'hail' are in the first and fourth quadrant, close to 'water'; these words are surrounded by 'bldg' (building), 'smoke', 'equipment', 'alarm', 'safety', 'power', 'library', etc. We conclude that these embeddings make perfect sense in an insurance claim context. Note that we have applied some pre-processing, and embeddings could even be improved by further pre-processing, e.g., 'vandalism' and 'vandalize' or 'hs' and 'high school' are used.

Another nice observation is that the embeddings tend to build a circle around the origin, see Fig. 10.2. This is enforced by embedding $W=142$ different words into a $b=2$ dimensional space so that dissimilar words optimally repulse each other. ■

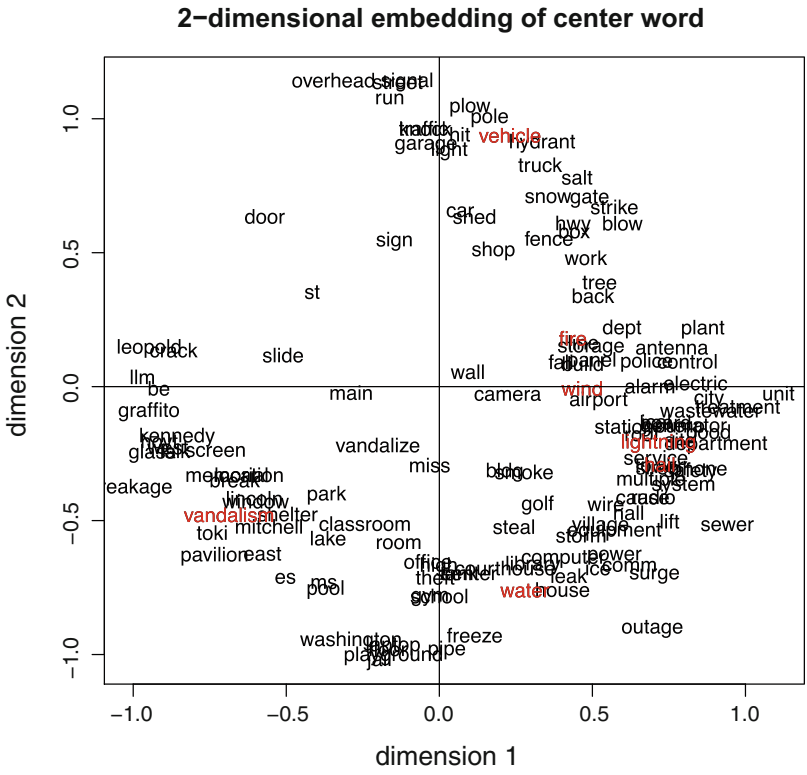

Fig. 10.2 Two-dimensional skip-gram embedding using negative sampling; in red color are the insured hazards 'vehicle', 'fire', 'lightning', 'wind', 'hail', 'water' and 'vandalism'

10.2.2 Global Vectors Algorithm

A second popular word embedding approach is global vectors (GloVe) developed by Pennington et al. [300], we also refer to Chaubard et al. [68]. GloVe is an unsupervised learning method that performs a word-word clustering (center-context pairs) over all available texts. Assume that the tokenization of all texts provides us with the words $w \in \mathcal{W}$. Choose a fixed context window size $c \in \mathbb{N}$ and define the matrix

$$\boldsymbol{C} = \big(C(w,\widetilde{w})\big)_{w,\widetilde{w}\in\mathcal{W}} \in \mathbb{N}_0^{W\times W},$$

with $C(w,\widetilde{w})$ counting the number of co-occurrences of w and $\widetilde{w}$ over all available texts where the word $\widetilde{w}$ appears as a context word of the center word w (for the given window size c). We note that $\boldsymbol{C}$ is a symmetric matrix that is typically sparse as many words do not appear in the context of other words (on finitely many texts). Figure 10.3 shows the center-context pairs $(w,\widetilde{w})$ co-occurrence matrix $\boldsymbol{C}$

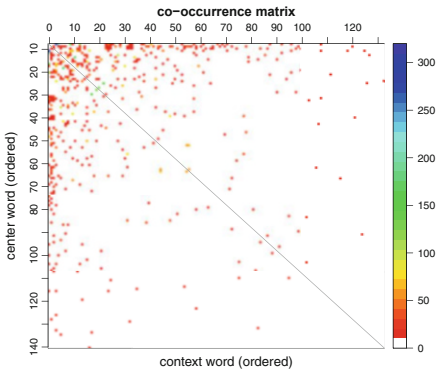

Fig. 10.3 Center-context pairs $(w, \widetilde{w})$ co-occurrence matrix $\boldsymbol{C}$ of Example 10.2; the color scale gives the observed frequencies

of Example 10.2 which is based on $W = 142$ words and 23'952 center-context pairs. The color pixels indicate the pairs that occur in the data, $C(w, \widetilde{w}) > 0$, and the white space corresponds to the pairs that have not been observed in the texts, $C(w, \widetilde{w}) = 0$. This plot confirms the sparsity of the center-context pairs; the words are ordered w.r.t. their frequencies in the texts.

In an empirical analysis Pennington et al. [300] have observed that the crucial quantities to be considered are the ratios for fixed context words. That is, for a context word $\widetilde{w}$ study a function of the center words w and v (subject to existence of the right-hand side)

$$(w, v, \widetilde{w}) \;\mapsto\; F(w, v, \widetilde{w}) = \frac{C(w, \widetilde{w}) / \sum_{\widetilde{u} \in \mathcal{W}} C(w, \widetilde{u})}{C(v, \widetilde{w}) / \sum_{\widetilde{u} \in \mathcal{W}} C(v, \widetilde{u})} \;=\; \frac{\widehat{p}(\widetilde{w}|w)}{\widehat{p}(\widetilde{w}|v)},$$

$\widehat{p}$ denoting the empirical probabilities. An empirical analysis suggests that such an approach seems to lead to a good discrimination of the meanings of the words, see Sect. 3 in Pennington et al. [300]. Further simplifications and assumptions provide the following ansatz, for details we refer to Pennington et al. [300],

$$\log C(w, \widetilde{w}) \;\approx\; \langle \widetilde{\boldsymbol{e}}(\widetilde{w}), \boldsymbol{e}(w) \rangle + \widetilde{\beta}_{\widetilde{w}} + \beta_w,$$

with intercepts $\widetilde{\beta}_{\widetilde{w}}, \beta_w \in \mathbb{R}$. There is still one issue, namely, that $\log C(w, \widetilde{w})$ may not be well-defined as certain pairs $(w, \widetilde{w})$ are not observed. Therefore, Pennington et al. [300] propose to solve a weighted squared error loss function problem to find the embedding mappings $\boldsymbol{e}, \widetilde{\boldsymbol{e}}$ and intercepts $\widetilde{\beta}_{\widetilde{w}}, \beta_w \in \mathbb{R}$. Their objective function is given by

$$\sum_{w, \widetilde{w} \in \mathcal{W}} \chi(C(w, \widetilde{w})) \left(\log C(w, \widetilde{w}) - \langle \widetilde{\boldsymbol{e}}(\widetilde{w}), \boldsymbol{e}(w) \rangle - \widetilde{\beta}_{\widetilde{w}} - \beta_w \right)^2, \tag{10.10}$$

with weighting function

$$x \geq 0 \;\mapsto\; \chi(x) = \left(\frac{x \wedge x_{\max}}{x_{\max}}\right)^{\alpha},$$

for $x_{\max} > 0$ and $\alpha > 0$. Pennington et al. [300] state that the model depends weakly on the cutoff point $x_{\max}$, they propose $x_{\max} = 100$, and a sub-linear behavior seems to outperform a linear one, suggesting, e.g., a choice of $\alpha = 3/4$. Under these choices the embeddings $\boldsymbol{e}$ and $\widetilde{\boldsymbol{e}}$ are found by minimizing the objective function (10.10) for the given data. Note that $\lim_{x \downarrow 0} \chi(x)(\log x)^2 = 0$.

Example 10.3 (GloVe Word Embedding) We provide an example using the GloVe embedding model, and we revisit the data of Example 10.2; we also use exactly the same pre-processing as in that example. We start from $N = 23'952$ center-context pairs.

In a first step we count the number of co-occurrences $C(w, \widetilde{w})$. There are only 4'972 pairs that occur, $C(w, \widetilde{w}) > 0$, this corresponds to the colors in Fig. 10.3. With these 4'972 pairs we have to fit 568 embedding weights (for the embedding dimension $b = 2$) and 284 intercepts $\widetilde{\beta}_{\widetilde{w}}, \beta_w$, thus, 852 parameters in total. The results of this fitting are shown in Fig. 10.4.

The general picture in Fig. 10.4 is similar to Fig. 10.2, e.g., 'vandalism' is surrounded by 'graffito', 'window', 'pavilion', names of cites and parks, 'ms' and 'es'; or 'vehicle' is surrounded by 'pole', 'traffic', 'street', 'signal'. However, the clustering of the words around the origin shows a crucial difference between GloVe and the negative sampling of word2vec. The problem here is that we do not have sufficiently many observations. We have 4'972 center-context pairs that occur, $C(w, \widetilde{w}) > 0$. 2'396 of these pairs occur exactly once, $C(w, \widetilde{w}) = 1$, this is almost half of the observations with $C(w, \widetilde{w}) > 0$. GloVe (10.10) considers these observations on the log-scale which provides $\log C(w, \widetilde{w}) = 0$ for the pairs that occur exactly once. The weighted square loss for these pairs is minimized by either setting $\widetilde{\boldsymbol{e}}(\widetilde{w}) = 0$ or $\boldsymbol{e}(w) = 0$, supposed that the intercepts are also set to 0. This is exactly what we observe in Fig. 10.4 and, thus, successfully fitting GloVe would require much more (frequent) observations. ■

Remark 10.4 (Pre-trained Word Embeddings) In practical applications we rely on pre-trained word embeddings. For GloVe there are pre-trained versions that can be downloaded.[4] These pre-trained versions comprise a vocabulary of 400K words, and they exist for the embedding dimensions $b = 50, 100, 200, 300$. These GloVe's have been trained on Wikipedia 2014 and Gigaword 5 which provided roughly 6B tokens. Another pre-trained open-source model that can be downloaded is spaCy.[5]

[4] https://nlp.stanford.edu/projects/glove/.

[5] https://spacy.io/models/en#en_core_web_md.

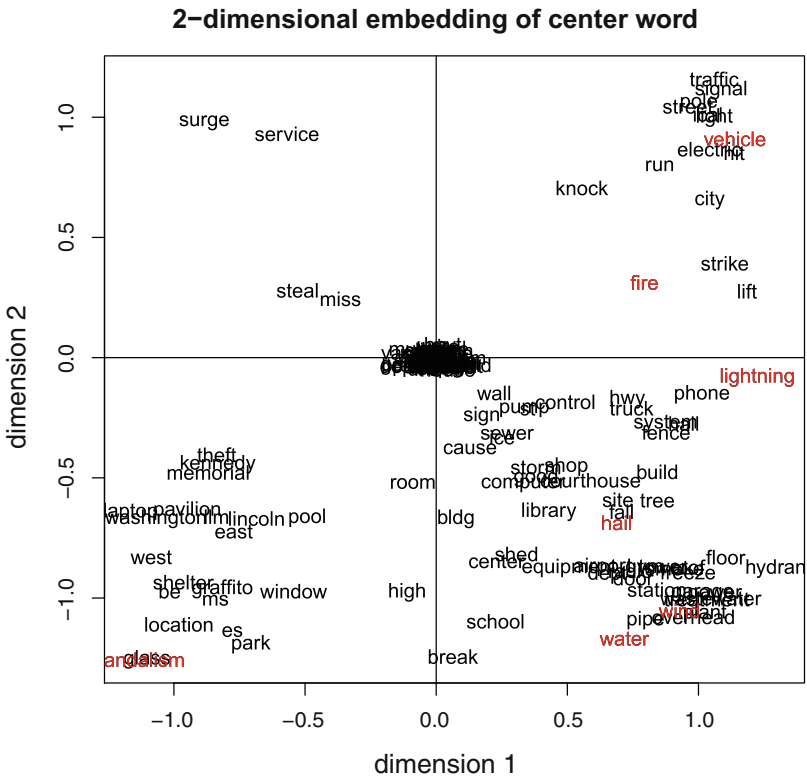

Fig. 10.4 Two-dimensional GloVe embedding; in red color are the insured hazards 'vehicle', 'fire', 'lightning', 'wind', 'hail', 'water' and 'vandalism'

Pre-trained embeddings can be problematic if we work in very specific settings. For instance, the Wisconsin LGPIF data contains the word 'Lincoln' in the claim descriptions. Now, Lincoln is a county in Wisconsin, it is town in Kewaunee County in Wisconsin, it is a former US president, there are Lincoln memorials, it is a common street name, it is a car brand and there are restaurants with this name. In our context, Lincoln is most commonly used w.r.t. the Lincoln Elementary and Middle Schools. On the other hand, it is likely that in pre-trained embeddings a different meaning of Lincoln is predominant, and therefore the embedding may not be reasonable for our insurance problem.

10.3 Lab: Predictive Modeling Using Word Embeddings

This section gives an example of applying the word embedding technique to a predictive modeling setting. This example is based on the Wisconsin LGPIF data set illustrated in Listing 10.1. Our goal is to predict the hazard types on line 10 of Listing 10.1 from the claim descriptions on line 11. We perform the same data cleaning process as in Example 10.2. This provides us with $W = 1'829$ different words, and the resulting (short) claim descriptions have a maximal length of $T = 9$. After padding with zeros we receive $n = 6'031$ claim descriptions given by texts $(w_1, \ldots, w_T)^\top \in \mathcal{W}_0^T$; we apply the padding to the left end of the sentences.

Word2vec Using Negative Sampling We start by the word2vec embedding technique using the negative sampling. We follow Example 10.2, and to successfully embed the available words $w \in \mathcal{W}$ we restrict the vocabulary to the words that are used at least 20 times. This reduces the vocabulary from 1'892 different words to 142 different words. The number of claim descriptions are reduced to 5'883 because 148 claim descriptions do not contain any of these 142 different words and, thus, cannot be classified as one of the hazard types (based on this reduced vocabulary).

In a first analysis we choose the embedding dimension $b = 2$, and this provides us with the word2vec embedding map that is illustrated in Fig. 10.2. Based on these embeddings we aim at predicting the hazard types from the claim descriptions. We have 9 different hazard types: Fire, Lightning, Hail, Wind, WaterW, WaterNW, Vehicle, Vandalism and Misc.[6] Therefore, we design a categorical classification model that has 9 different labels, we refer to Sect. 2.1.4.

Listing 10.7 R code for the hazard type prediction based on a word2vec embedding

```
1  input = layer_input(shape = list(T), name = "input")
2  #
3  word2vec = input %>%
4      layer_embedding(input_dim = W+1, output_dim = b, input_length = T,
5                      weights=list(wordEmb), trainable=FALSE) %>%
6      layer_flatten()
7  # response = word2vec %>%
8      layer_dense(units=20, activation='tanh', name='FNLayer1') %>%
9      layer_dense(units=15, activation='tanh', name='FNLayer2') %>%
10     layer_dense(units=9, activation='softmax', name='output')
11 #
12 model = keras_model(inputs = c(input), outputs = c(response))
```

The R code for the hazard type prediction is presented in Listing 10.7. The crucial part is shown on line 5. Namely, the embedding map $\boldsymbol{e}(w) \in \mathbb{R}^b$, $w \in \mathcal{W}$ is initialized with the embedding weights `wordEmb` received from Example 10.2, and

[6] WaterW relates to weather related water claims, and WaterNW relates to non-weather related water claims.

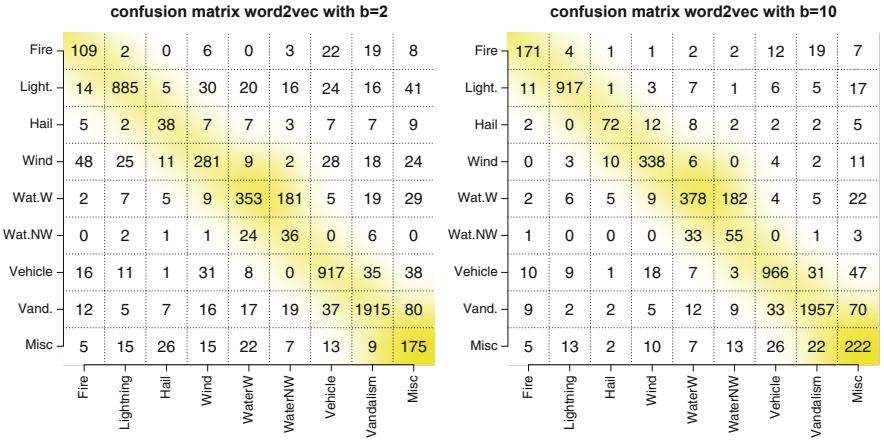

Fig. 10.5 Confusion matrices of the hazard type prediction using a word2vec embedding based on negative sampling (lhs) $b = 2$ dimensional embedding and (rhs) $b = 10$ dimensional embedding; columns show the observations and rows show the predictions

these embedding weights are declared to be non-trainable.[7] These features are then inputted into a FN network with two FN layers having $(q_1, q_2) = (20, 15)$ neurons, and as output activation we choose the softmax function. This model has 286 non-trainable embedding weights, and $r = (9 \cdot 2 + 1)20 + (20 + 1)15 + (15 + 1)9 = 839$ trainable parameters.

We fit this network using the `nadam` version of the gradient descent method, and we exercise an early stopping on a 20% validation data set (of the entire data). This network is fitted in a few seconds, and the results are presented in Fig. 10.5 (lhs). This figure shows the confusion matrix of prediction vs. observed (row vs. column). The general results look rather good, there are only difficulties to distinguish WaterN from WaterNW claims.

In a second analysis, we increase the embedding dimension to $b = 10$ and we perform exactly the same procedure as above. A higher embedding dimension allows the embedding map to better discriminate the words in their meanings. However, we should not go for a too high b because we have only 142 different words and 47'904 center-context pairs $(w, \widetilde{w})$ to learn these embeddings $\boldsymbol{e}(w) \in \mathbb{R}^b$. A higher embedding dimension also increases the number of network weights in the first FN layer on line 9 of Listing 10.7. This time, we need to train $r = (9 \cdot 10 + 1)20 + (20 + 1)15 + (15 + 1)9 = 2'279$ parameters. The results are presented in Fig. 10.5 (rhs). We observe an overall improvement compared to the 2-dimensional embeddings. This is also confirmed by Table 10.1 which gives the deviance losses and the misclassification rates.

[7] The zeros from padding are mapped to the origin.

Table 10.1 Hazard prediction results summarized in deviance losses and misclassification rates

	Number of parameters		Deviance loss	Misclassification rate
	Embedding	Network		
word2vec negative sampling, $b = 2$	286	839	0.1442	19.9%
word2vec negative sampling, $b = 10$	1'430	2'279	0.0912	13.7%
FN GloVe using all words, $b = 50$	91'500	9'479	0.0802	11.7%
LSTM GloVe using all words, $b = 50$	91'500	3'369	0.0802	12.1%
Word similarity embedding, $b = 7$	12'810	1'739	0.1396	21.1%

Pre-trained GloVe Embedding In a next analysis we use the pre-trained GloVe embeddings, see Remark 10.4. This allows us to use all $W = 1'892$ words that appear in the $n = 6'031$ claim descriptions, and we can also classify all these claims. I.e., we can classify more claims, here, compared to the 5'883 claims we have classified based on the self-trained word2vec embeddings. Apart from that, all modeling steps are chosen as above. Only the higher embedding dimension $b = 50$ from the pre-trained `glove.6B.50d` increases the size of the network parameter to $r = (9 \cdot 50 + 1)20 + (20 + 1)15 + (15 + 1)9 = 9'479$ parameters; remark that the 91'500 embedding weights are not trained as they come from the pre-trained GloVe embeddings. Using the `nadam` optimizer with an early stopping provides us with the results in Fig. 10.6 (lhs). Using this pre-trained GloVe embedding leads to a further improvement, this is also verified by Table 10.1. Using the pre-trained GloVe is two-fold. On the one hand, it allows us to use all words of the claim descriptions, which improves the prediction accuracy. On the other hand, the embeddings are not adapted to insurance problems, as these have been trained on Wikipedia and Gigaword texts. The former advantage overrules the latter shortcoming in our example.

All the results above have been using the FN network of Listing 10.7. We made this choice because our texts have a maximal length of $T = 9$, which is very short. In general, texts should be understood as time-series, and RN networks are a canonical choice to analyze these time-series. Therefore, we study again the pre-trained GloVe embeddings, but we process the texts with a LSTM architecture, we refer to Sect. 8.3.1 for LSTM layers.
Listing 10.8 shows the LSTM architecture used. On line 9 we set the variable `return_sequences` to true which implies that all intermediate steps $z_t^{[1]}$, $1 \le t \le T$, are outputted to a time-distributed FN layer on line 10, see Sect. 8.2.4 for time-distributed layers. This LSTM network has $r = 4(50 + 1 + 10)10 + (10 + 1)10 + (90 + 1)9 = 3'369$ parameters. The flatten layer on line 11 of Listing 10.8 turns the $T = 9$ outputs $z_t^{[2]} \in \mathbb{R}^{q_2}$, $1 \le t \le T$, of dimension $q_2 = 10$ into a vector of size $Tq_2 = 90$. This vector is then fed into the output layer on line 12. At this stage, one could reduce the dimension of the parameter by setting a max-pooling layer in between the flatten and the output layer.

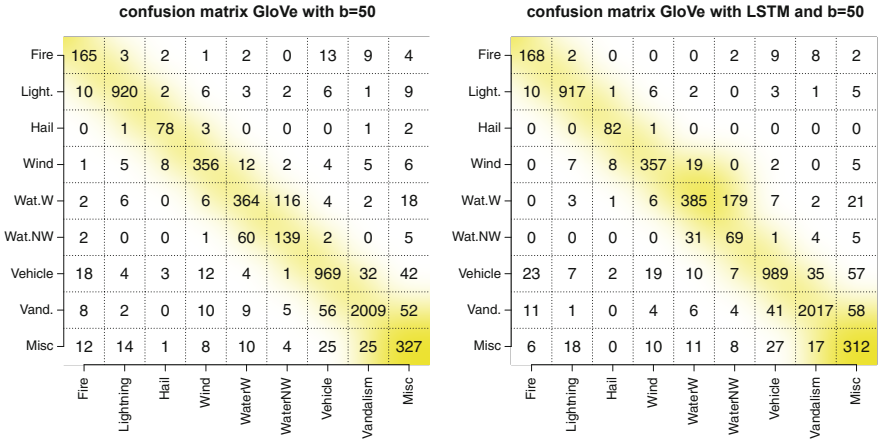

Fig. 10.6 Confusion matrices of the hazard type prediction using the pre-trained GloVe with $b = 50$ (lhs) FN network and (rhs) LSTM network; columns show the observations and rows show the predictions

Listing 10.8 R code for the hazard type prediction using a LSTM architecture

```
input = layer_input(shape = list(T), name = "input")
#
word2vec = input %>%
    layer_embedding(input_dim = W+1, output_dim = b, input_length = T,
                    weights=list(wordEmb), trainable=FALSE) %>%
    layer_flatten()
#
response = word2vec %>%
 layer_lstm(units=10, activation='tanh', return_sequences=TRUE,
                                         name='LSTM') %>%
 time_distributed(layer_dense(units=10, activation='tanh', name='FNLayer')) %>%
 layer_flatten() %>%
 layer_dense(units=9, activation='softmax', name='output')
#
model = keras_model(inputs = c(input), outputs = c(response))
```

We fit this LSTM architecture to the data using the pre-trained GloVe embeddings. The results are presented in Fig. 10.6 (rhs) and Table 10.1. We receive the same deviance loss, and the misclassification rate is slightly worse than in the FN network case (with the same pre-trained GloVe embeddings). Note that the deviance loss is calculated on the estimated classification probabilities $\widehat{\boldsymbol{p}}(\boldsymbol{x}) = (\widehat{p}_1(\boldsymbol{x}), \ldots, \widehat{p}_9(\boldsymbol{x}))^\top$, and the labels are received by

$$\widehat{Y} = \widehat{Y}(\boldsymbol{x}) \;=\; \underset{k=1,\ldots,9}{\arg\max}\, \widehat{p}_k(\boldsymbol{x}).$$

Thus, it may happen that the improvements on the estimated probabilities are not fully reflected on the predicted labels.

Word (Cosine) Similarity In our final analysis we work with the pre-trained GloVe embeddings $\boldsymbol{e}(w) \in \mathbb{R}^{50}$ but we first try to reduce the embedding dimension b. For this we follow Lee et al. [236], and we consider a *word similarity*. We can define the similarity of the words w and $w' \in \mathcal{W}$ by considering the scalar product of their embeddings

$$\text{sim}^{(u)}(w, w') = \langle \boldsymbol{e}(w), \boldsymbol{e}(w') \rangle \qquad \text{or} \qquad \text{sim}^{(n)}(w, w') = \frac{\langle \boldsymbol{e}(w), \boldsymbol{e}(w') \rangle}{\|\boldsymbol{e}(w)\|_2 \|\boldsymbol{e}(w')\|_2}. \tag{10.11}$$

The first one is an unweighted version and the second one is a normalized version scaling with the corresponding Euclidean norms so that the similarity measure is within $[-1, 1]$. In fact, the latter is also called cosine similarity. To reduce the embedding dimension and because we have a classification problem with hazard names, we can evaluate the (cosine) similarity of all used words $w \in \mathcal{W}$ to the hazards $h \in \mathcal{H} = \{\texttt{fire}, \texttt{lightning}, \texttt{hail}, \texttt{wind}, \texttt{water}, \texttt{vehicle}, \texttt{vandalism}\}$. Observe that `water` is further separated into weather related and non-weather related claims, and there is a further hazard type called `misc`, which collects all the rest. We could choose more words in $\mathcal{H}$ to more precisely describe these water and other claims. If we just use $\mathcal{H}$ we obtain a $b = |\mathcal{H}| = 7$ dimensional embedding mapping

$$w \in \mathcal{W}_0 \;\mapsto\; \boldsymbol{e}^{(a)}(w) = \left(\text{sim}^{(a)}(w, \texttt{fire}), \ldots, \text{sim}^{(a)}(w, \texttt{vandalism})\right)^\top \in \mathbb{R}^{b=7}, \tag{10.12}$$

for $a \in \{u, n\}$. This gives us for every $\texttt{text} = (w_1, \ldots, w_T)^\top \in \mathcal{W}_0^T$ the pre-processed features

$$\texttt{text} \;\mapsto\; \left(\boldsymbol{e}^{(a)}(w_1), \ldots, \boldsymbol{e}^{(a)}(w_T)\right)^\top \in \mathbb{R}^{T \times b}. \tag{10.13}$$

Lee et al. [236] apply a max-pooling layer to these embeddings which are then inputted into GAM classification model. We use a different approach here, and directly use the unweighted ($a = u$) text representations (10.13) as an input to a network, either of FN network type of Listing 10.7 or of LSTM type of Listing 10.8. If we use the FN network type we receive the results on the last line of Table 10.1 and Fig. 10.7.

Comparing the results of the word similarity through the embeddings (10.12) and (10.13) to the other prediction results, we conclude that this word similarity approach is not fully competitive compared to working directly with the word2vec or GloVe embeddings. It seems that the projection (10.12) does not discriminate sufficiently for our classification task.

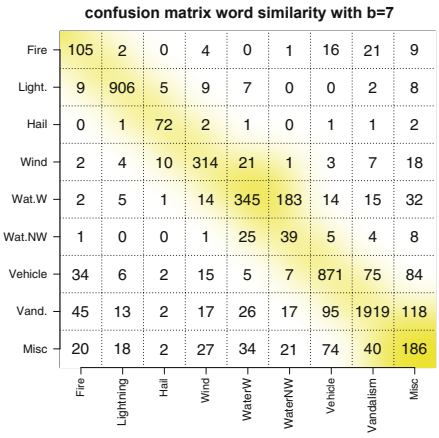

Fig. 10.7 Confusion matrix of the hazard type prediction using the word similarity (10.12)–(10.13) for $a = u$; columns show the observations and rows show the predictions

10.4 Lab: Deep Word Representation Learning

All examples above have been relying on embedding the words $w \in \mathcal{W}$ into a Euclidean space $\boldsymbol{e}(w) \in \mathbb{R}^b$ by performing a sort of unsupervised learning that provided word similarity clusters. The advantage of this approach is that the embedding is decoupled from the regression or classification task, this is computationally attractive. Moreover, once a suitable embedding has been learned, it can be used for several different tasks (in the spirit of transfer learning). The disadvantage of the pre-trained embeddings is that the embedding is not targeted to the regression task at hand. This has already been discussed in Remark 10.4 where we have highlighted that the meaning of some words (such as Lincoln) depends very much on its context.

Recent NLP aims at pre-processing a text as little as necessary, but tries to directly feed the raw sentences into RN networks such as LSTM or GRU architectures. Computationally this is much more demanding because we have to learn the embeddings and the network weights simultaneously, we refer to Table 10.1 to indicate the number of parameters involved. The purpose of this short section is to give an example, though our NLP database is rather small; this latter approach usually requires a huge database and the corresponding computational power. Ferrario–Nägelin [126] provide a more comprehensive example on the classification of movie reviews. For their analysis they evaluated approximately 50'000 movie reviews each using between 235 and 2'498 words. Their analysis was implemented on the ETH High Performance Computing (HPC) infrastructure Euler[8], and their run times have been between 20 and 30 minutes, see Table 8 of Ferrario–Nägelin [126].

[8] https://scicomp.ethz.ch/wiki/Euler

Since we neither have the computational power nor the big data to fit such a NLP application, we start the gradient descent fitting in the initial embedding weights $\boldsymbol{e}(w) \in \mathbb{R}^b$ that either come from the word2vec or the GloVe embeddings. During the gradient descent fitting, we allow these weights to change w.r.t. the regression task at hand. In comparison to Sect. 10.3, this only requires minor changes to the R code, namely, the only modification needed is to change from `FALSE` to `TRUE` on lines 5 in Listings 10.7 and 10.8. This change allows us to learn adapted weights during the gradient descent fitting. The resulting classification models are now very high-dimensional, and we need to carefully assess the early stopping rule, otherwise the model will (in-sample) over-fit to the learning data.

In Fig. 10.8 we provide the results that correspond to the self-trained word2vec embeddings given in Fig. 10.5, and the corresponding numerical results are given in Table 10.2. We observe an improvement in the prediction accuracy in both cases by letting the embedding weights being learned during the network fitting, and we receive a misclassification rate of 11.6% and 11.0% for the embedding dimensions $b = 2$ and $b = 10$, respectively, see Table 10.2.

Figure 10.8 (rhs) illustrates how the embeddings have changed from the initial (pre-trained) embeddings $\boldsymbol{e}^{(0)}(w)$ (coming from the word2vec negative sampling) to the learned embeddings $\widehat{\boldsymbol{e}}(w)$. We measure these changes in terms of the unweighted similarity measure defined in (10.11), and given by

$$\left\langle \boldsymbol{e}^{(0)}(w), \widehat{\boldsymbol{e}}(w) \right\rangle. \tag{10.14}$$

The upper horizontal line is a manually set threshold to identify the words w that experience a major change in their embeddings. These are the words 'vandalism', 'lightning', 'grafito', 'fence', 'hail', 'freeze', 'blow' and 'breakage'. Thus, these words receive a different embedding location/meaning which is more favorable for our classification task.

A similar analysis can be performed for the pre-trained GloVe embeddings. There we expected bigger changes to the embeddings since the GloVe embeddings have not been learned in an insurance context, and the embeddings will be adapted to the insurance prediction problem. We refrain from giving an explicit analysis, here, because to perform a thorough analysis we would need (much) more data.

We conclude this example with some remarks. We emphasize once more that our available data is minimal, and we expect (even much) better results for longer claim descriptions. In particular, our data is not sufficient to discriminate the weather related from the non-weather related water claims, as the claim descriptions seem to focus on the water claim itself and not on its cause. In a next step, one should use claim descriptions in order to predict the claim sizes, or to improve their predictions if they are based on classical tabular features, only. Here, we see some potential, in particular, w.r.t. medical claims, as medical reports may clearly indicate the severity of the claim as well as these reports may give some insight into the recovery process. Thus, our small example may only give some intuition of what is possible with

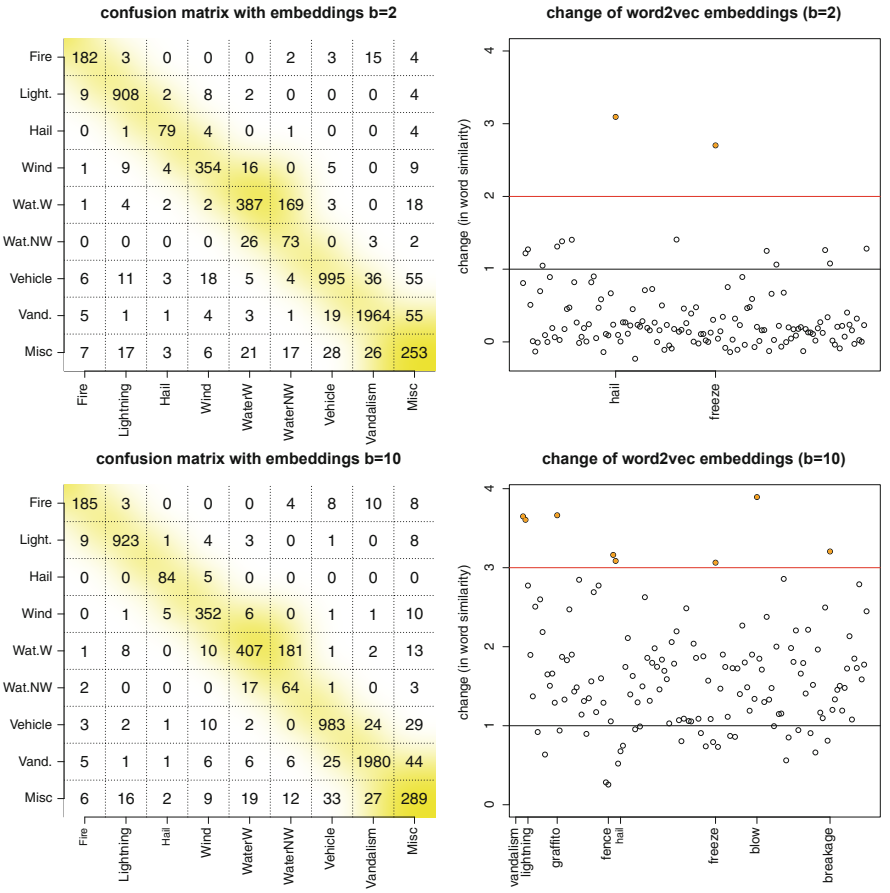

Fig. 10.8 Confusion matrices and the changes in the embeddings compared to the pre-trained word2vec embeddings of Fig. 10.5 for the dimensions $b = 2$ and $b = 10$

Table 10.2 Hazard prediction results summarized in deviance losses and misclassification rates: pre-trained embeddings vs. network learned embeddings

	Number of parameters		Deviance loss	Misclass. rate
	Non-trainable	Trainable		
word2vec negative sampling, $b = 2$	286	839	0.1442	19.9%
word2vec improved embedding, $b = 2$		1'125	0.0814	11.7%
word2vec negative sampling, $b = 10$	1'430	2'279	0.0912	13.7%
word2vec improved embedding, $b = 10$		3'709	0.0714	10.5%

(unstructured) text data. Unfortunately, the LGPIF data of Listing 10.1 did not give us any satisfactory results for the claim size prediction, this for several reasons. Firstly, the data is rather heterogeneous ranging from small to very large claims and any member of the EDF struggles to model this data; we come back to a different modeling proposal of heterogeneous data in Sect. 11.3.2. Secondly, the claim descriptions are not very explanatory as they are too short for a more detailed information. Thirdly, the data has only 5'424 claims which seems small compared to the complexity of the problem that we try to solve.

10.5 Outlook: Creating Attention

In text recognition problems, obviously, not all the words in a sentence have the same importance. In the examples above, we have removed the stopwords as they may disturb the key understanding of our texts. Removing the stopwords means that we pay more attention to the remaining words. RN networks often face difficulty in giving the right recognition to the different parts of a sentence. For this reason, *attention layers* have gained more popularity recently. Attention layers are special modules in network architectures that allow the network to impose more weight on certain parts of the information in the features to emphasize their importance. The attention mechanism has been introduced in Bahdanau et al. [21]. There are different ways of modeling attention, the most popular one is the so-called *dot-product attention*, we refer to Vaswani et al. [366], and in the actuarial literature we mention Kuo–Richman [231] and Troxler–Schelldorfer [354].

We start by describing a simple attention mechanism. Consider a sentence $\texttt{text} = (w_1, \ldots, w_T) \in \mathcal{W}_0^T$ that provides, under an embedding map $\boldsymbol{e} : \mathcal{W}_0 \to \mathbb{R}^b$, the embedded sentence $(\boldsymbol{e}(w_1), \ldots, \boldsymbol{e}(w_T))^\top \in \mathbb{R}^{T \times b}$. We choose a weight matrix $U_Q \in \mathbb{R}^{b \times b}$ and an intercept vector $\boldsymbol{u}_Q \in \mathbb{R}^b$. Based on these choices we consider for each word w_t of our sentence the score, called *query*,

$$\boldsymbol{q}_t = \tanh\left(\boldsymbol{u}_Q + U_Q \boldsymbol{e}(w_t)\right) \in (-1, 1)^b. \tag{10.15}$$

Matrix $Q = (\boldsymbol{q}_1, \ldots, \boldsymbol{q}_T)^\top \in \mathbb{R}^{T \times b}$ collects all queries. It is obtained by applying a time-distributed FN layer with b neurons to the embedded sentence $(\boldsymbol{e}(w_1), \ldots, \boldsymbol{e}(w_T))^\top$.

These queries $\boldsymbol{q}_t$ are evaluated with a so-called *key* $\boldsymbol{k} \in \mathbb{R}^b$ giving us the *attention weights*

$$\alpha_t = \frac{\exp\langle \boldsymbol{k}, \boldsymbol{q}_t \rangle}{\sum_{s=1}^T \exp\langle \boldsymbol{k}, \boldsymbol{q}_s \rangle} \in (0, 1) \qquad \text{for } 1 \le t \le T. \tag{10.16}$$

Using these attention weights $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_T)^\top \in (0,1)^T$ we encode the sentence `text` as

$$\begin{aligned} \texttt{text} = (w_1, \ldots, w_T) \;\mapsto\; \boldsymbol{w}^* &= \sum_{t=1}^{T} \alpha_t \boldsymbol{e}(w_t) \\ &= (\boldsymbol{e}(w_1), \ldots, \boldsymbol{e}(w_T))\, \boldsymbol{\alpha} \;\in\; \mathbb{R}^b. \end{aligned} \tag{10.17}$$

Thus, to every sentence `text` we assign a categorical probability vector $\boldsymbol{\alpha} = \boldsymbol{\alpha}(\texttt{text}) \in \Delta_T$, see Sect. 2.1.4, (6.22) and (5.69), which is encoding this sentence `text` to a b-dimensional vector $\boldsymbol{w}^* \in \mathbb{R}^b$. This vector is then further processed by the network. Such a construction is called a *self-attention mechanism* because the text $(w_1, \ldots, w_T) \in \mathcal{W}_0^T$ is used to formulate the queries in (10.15), but, of course, these queries could also be coming from a completely different source. In the above set-up we have to learn the following parameters $U_Q \in \mathbb{R}^{b\times b}$ and $\boldsymbol{u}_Q, \boldsymbol{k} \in \mathbb{R}^b$, assuming that the embedding map $\boldsymbol{e} : \mathcal{W}_0 \to \mathbb{R}^b$ has already been specified.

There are several generalizations and modifications to this self-attention mechanism. The most common one is to expand the vector $\boldsymbol{w}^* \in \mathbb{R}^b$ in (10.17) to a matrix $W^* = (\boldsymbol{w}_1^*, \ldots, \boldsymbol{w}_q^*) \in \mathbb{R}^{b\times q}$. This matrix W^* can be interpreted as having q neurons $\boldsymbol{w}_j^* \in \mathbb{R}^b$, $1 \le j \le q$. For this, one replaces the key $\boldsymbol{k} \in \mathbb{R}^b$ by a matrix-valued key $K = (\boldsymbol{k}_1, \ldots, \boldsymbol{k}_q) \in \mathbb{R}^{b\times q}$. This allows one to calculate the attention weight matrix

$$\begin{aligned} A = \big(\alpha_{t,j}\big)_{1\le t\le T, 1\le j\le q} \;&=\; \left(\frac{\exp\langle \boldsymbol{k}_j, \boldsymbol{q}_t\rangle}{\sum_{s=1}^T \exp\langle \boldsymbol{k}_j, \boldsymbol{q}_s\rangle} \right)_{1\le t\le T, 1\le j\le q} \\ &= \operatorname{softmax}(QK) \;\in\; (0,1)^{T\times q}, \end{aligned}$$

where the softmax function is applied column-wise. I.e., the attention weight matrix $A \in (0,1)^{T\times q}$ has columns $\boldsymbol{\alpha}_j = (\alpha_{1,j}, \ldots, \alpha_{T,j})^\top \in \Delta_T$, $1 \le j \le q$, which are normalized to total weight 1, this is equivalent to (10.16). This is used to encode the sentence `text`

$$\begin{aligned} (\boldsymbol{e}(w_1), \ldots, \boldsymbol{e}(w_T)) \in \mathbb{R}^{b\times T} \;\mapsto\; W^* &= (\boldsymbol{e}(w_1), \ldots, \boldsymbol{e}(w_T))\, A \\ &= \left(\sum_{t=1}^{T} \alpha_{t,j} \boldsymbol{e}(w_t) \right)_{1\le j\le q} \;\in\; \mathbb{R}^{b\times q}. \end{aligned} \tag{10.18}$$

Mapping (10.18) is called an *attention layer*. Let us give some remarks.

Remarks 10.5

- Encoding (10.18) gives a natural multi-dimensional extension of (10.17). The crucial parts are the attention weights $\boldsymbol{\alpha}_j \in \Delta_T$ which weigh the different words

$(w_t)_{1\leq t\leq T}$. In the multi-dimensional case, we perform this weighting mechanism multiple times (in different directions), allowing us to extract different features from the sentences. In contrast, in (10.17) we only do this once. This is similar as going form one neuron to a layer of q neurons.

- The above structure uses a self-attention mechanism because the queries involve the words themselves, and the weight matrix $U_Q \in \mathbb{R}^{b\times b}$ and the intercept vector $\boldsymbol{u}_Q \in \mathbb{R}^b$ are learned with gradient descent. Concerning the key $K \in \mathbb{R}^{b\times q}$ one often chooses another self-attention mechanism by choosing a (non-linear) function $K = K(w_1, \ldots, w_T)$ to infer optimal keys.
- These attention layers are also the building blocks of *transformer models*. Transformer models use attention layers (10.18) of dimension $W^* \in \mathbb{R}^{b\times T}$ and skip connections to transform the input

$$W = (\boldsymbol{e}(w_1), \ldots, \boldsymbol{e}(w_T)) \in \mathbb{R}^{b\times T} \;\mapsto\; \frac{W + W^*}{2} \in \mathbb{R}^{b\times T}. \tag{10.19}$$

  Stacking multiple of these layers (10.19) transforms the original input W by weighing the important information in feature W for the prediction task at hand. Compared to LSTM layers this no longer sequentially screens the text but it directly acts on the part of the text that seems important.
- The attention mechanism is applied to a matrix $(\boldsymbol{e}(w_1), \ldots, \boldsymbol{e}(w_T))^\top \in \mathbb{R}^{T\times b}$ which presents a numerical encoding of the sentence $(w_1, \ldots, w_T)^\top \in \mathcal{W}_0^T$. Kuo–Richman [231] propose to apply this attention mechanism more generally to categorical feature components. Assume that we have T categorical feature components $x_1, \ldots, x_T$, after embedding them into b-dimensional Euclidean spaces we receive a representation $(\boldsymbol{e}(x_1), \ldots, \boldsymbol{e}(x_T))^\top \in \mathbb{R}^{T\times b}$, see (7.31). Naturally, this can now be further processed by putting different attention on the components of this embedding exactly using an attention layer (10.18), alternatively we can use transformer layers (10.19).

Example 10.6 We revisit the hazard type prediction example of Sect. 10.3. We select the $b = 10$ word2vec embedding (using negative sampling) and the pre-trained GloVe embedding of Table 10.1. These embeddings are then further processed by applying the attention mechanism (10.15)–(10.17) on the embeddings using one single attention neuron. Listing 10.9 gives the corresponding implementation. On line 9 we have the query (10.15), on lines 10–13 the key and the attention weights (10.16), and on line 15 the encodings (10.17). We then process these encodings through a FN network of depth $d = 2$, and we use the softmax output activation to receive the categorical probabilities. Note that we keep the learned word embeddings $\boldsymbol{e}(w)$ as non-trainable on line 5 of Listing 10.9.
Table 10.3 gives the results, and Fig. 10.9 shows the confusion matrix. We conclude that the results are rather similar, this attention mechanism seems to work quite well, and with less parameters, here. ■

Listing 10.9 R code for the hazard type prediction using an attention layer with $q = 1$

```
input = layer_input(shape = list(T), name = "input")
#
word2vec = input %>%
    layer_embedding(input_dim = W+1, output_dim = b, input_length = T,
                    weights=list(wordEmb), trainable=FALSE) %>%
    layer_flatten()
#
attention = word2vec %>%
 time_distributed(layer_dense(units=b, activation='tanh')) %>%
 time_distributed(layer_dense(units=1, activation='linear',
                                       use_bias=FALSE)) %>%
 layer_flatten() %>%
 layer_dense(unit=T, activation='softmax', weights=list(diag(T)),
                                        use_bias=FALSE, trainable=FALSE)
#
response = list(attention, word2vec) %>% layer_dot(axes=1) %>%
          layer_dense(units=20, activation='tanh') %>%
          layer_dense(units=15, activation='tanh') %>%
          layer_dense(units=9, activation='softmax')
#
model = keras_model(inputs = c(input), outputs = c(response))
```

Table 10.3 Hazard prediction results summarized in deviance losses and misclassification rates

	Number of parameters		Deviance loss	Misclassification rate
	Embedding	Network		
word2vec negative sampling, $b = 10$	1'430	2'279	0.0912	13.7%
word2vec attention, $b = 10$	1'430	799	0.0784	12.0%
FN GloVe using all words, $b = 50$	91'500	9'479	0.0802	11.7%
GloVe attention, $b = 50$	91'500	4'079	0.0824	12.6%

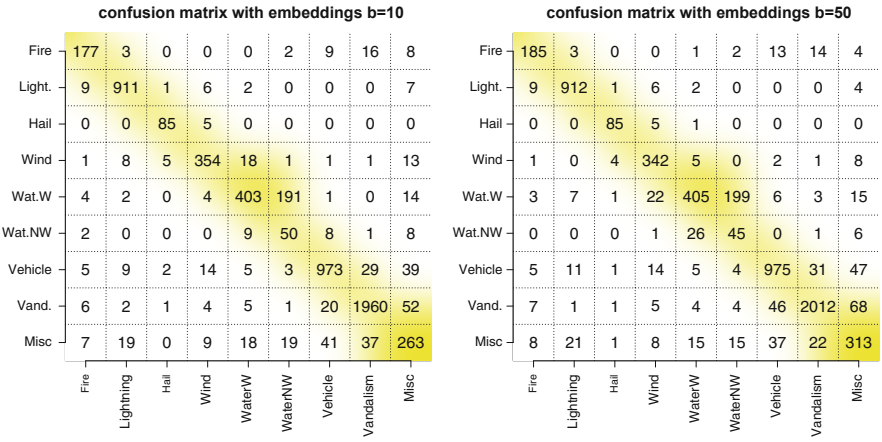

Fig. 10.9 Confusion matrices of the hazard type prediction (lhs) using an attention layer on the word2vec embeddings with $b = 10$, and (rhs) using an attention layer on the pre-trained GloVe embeddings with $b = 50$; columns show the observations and rows show the predictions