Lecture 7: Ensembling & Negative Sampling

Deep Learning for Actuarial Modeling
36th International Summer School SAA
University of Lausanne

Ronald Richman, Salvatore Scognamiglio, Mario V. Wüthrich

2025-09-10

Introduction

Overview

The first part of this lecture presents network ensembling, called nagging. The second part of this lecture present negative sampling which is useful for an unsupervised embedding of text and words.

This lecture covers Chapters 5 and 8 of Wüthrich *et al.* (2025).

Network ensembling

- A critical point of network fitting is that it involves several elements of randomness. Even for a fixed architecture and fitting procedure, one typically has infinitely many equally good fitted models ('solutions').
- The elements of randomness involve:
  1. the initialization of the network weight $\vartheta^{[0]}$ for SGD;
  2. the random partition into learning sample $\mathcal{L}$ and test sample $\mathcal{T}$;
  3. the random partition into training sample $\mathcal{U}$ and validation sample $\mathcal{V}$;
  4. the random partition into the batches $(\mathcal{U}_k)_{k=1}^{\lfloor n/s \rfloor}$.
  5. There are further random items like drop-outs, etc.
- This makes early stopped SGD solutions (highly) non-unique. This non-uniqueness is typical for machine learning solutions.

Ensembling/nagging

- Breiman (1996) introduced *bagging* for regression trees.
- Bagging combines **b**ootstrap and **agg**regat**ing**. Bootstrap is a re-sampling technique, and this is combined with aggregation which has an averaging effect, reducing the randomness.
- We replace the bootstrap by different SGD 'solutions', because network fitting has the above mentioned items of randomness, we naturally receive multiple solutions.
- Ensembling of network predictors was introduced by Dietterich (2000a) and Dietterich (2000b). Subsequently, it was studied in Richman and Wüthrich (2020), where it was called *nagging* for **n**etwork **agg**regat**ing**.

Ensemble predictor

- Having multiple (conditionally) i.i.d. predictors $(\widehat{\mu}_j)_{j=1}^M$, one builds the *ensemble predictor*

$$\widehat{\mu}^{(M)} = \frac{1}{M} \sum_{j=1}^{M} \widehat{\mu}_j.$$

- This ensemble predictor has an estimation uncertainty

$$\sqrt{\mathrm{Var}\left(\widehat{\mu}^{(M)}\right)} = \frac{1}{\sqrt{M}} \sqrt{\mathrm{Var}(\widehat{\mu}_1)} \;\rightarrow\; 0 \qquad \text{for } M \rightarrow \infty.$$

- The important takeaway is that ensembling over conditionally i.i.d. predictors substantially reduces estimation uncertainty.
- Caveat: This does not say anything about a bias of the estimated model for the true model!

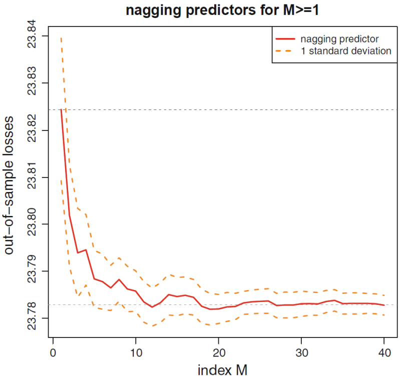
Nagging predictor

- Based on learning sample $\mathcal{L} = (Y_i, \boldsymbol{X}_i, v_i)_{i=1}^n$, choose M conditionally i.i.d. fitted FNNs, where the conditionally i.i.d. applies to the elements of randomness in SGD fitting.
- This gives us M conditionally i.i.d. FNNs $(\mu_{\widehat{\vartheta}_j})_{j=1}^M$, given $\mathcal{L}$.
- This motivates the *nagging predictor*

$$\widehat{\mu}_M^{\text{nagg}}(\boldsymbol{X}) = \frac{1}{M} \sum_{j=1}^M \mu_{\widehat{\vartheta}_j}(\boldsymbol{X}).$$

- This ensembling reduces the fluctuations by a factor $\sqrt{M}$.
- The next plot shows how many FNNs we need to ensemble to arrive at an optimal forecast model.

Out-of-sample Poisson deviance losses as a function of $M \geq 1$.



- This is the French MTPL claims count example.
- The Poisson deviance loss is scaled differently.
- We conclude that we need roughly 10 to 20 i.i.d. fitted FNNs.

Results: nagging predictor

model	*in-sample loss*	*out-of-sample loss*	*balance (in %)*
Poisson null model	47.722	47.967	7.36
Poisson GLM	45.585	45.435	7.36
Poisson FNN	44.846	44.925	7.17
nagging predictor	44.849	44.874	7.36

- For the nagging predictor we use $M = 10$ individual network fits.
- As expected, we receive an out-of-sample improvement.

Recommendation

In network predictions, always consider the nagging predictor $\widehat{\mu}_M^{\text{nagg}}$ with $M \in \{10, 20\}$. This will significantly improve the forecast model.

Tokenization of text and words

- When it comes to large unstructured text inputs, a *word embedding* approach is fitted in an *unsupervised learning* manner (using the *context*). We illustrate this.
- For word embedding, we change the notation to $w \in \{1, \ldots, W\}$ labeling all the words in the available vocabulary by integers.
- We start from a sentence `text` of length T

$$\texttt{text} = (w_1, \ldots, w_T) \in \mathbb{N}^T.$$

- The goal is to find a sensible word embedding (WE)

$$\boldsymbol{e}^{\mathrm{WE}} : \mathbb{N} \to \mathbb{R}^b, \qquad w \mapsto \boldsymbol{e}^{\mathrm{WE}}(w),$$

  for embedding space $\mathbb{R}^b$; Bengio, Courville and Vincent (2014).

- Based on unsupervised learning, one tries to learn embedding vectors from the contexts:
  - E.g., 'I'm driving by car to the city' and 'I'm driving my vehicle to the town center' uses similar words in a similar context.
  - Therefore, their embedding vectors should be close in the embedding space $\mathbb{R}^b$ because they are almost interchangeable.
- The goal is to learn such similarity in the meanings from the context.
- There are two different approaches:
  - Predict a *center word* from its *context*; a popular method is *continuous bag-of-words* (CBOW).
  - Predict the *context* from a *center word*; *skip-gram* is a popular approach.
- For simplicity, we only present skip-gram. The other version is quite similar; we refer to Wüthrich *et al.* (2025).

Context of words

- Consider a sentence

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

  where the positional indices $t \in \mathbb{N}$ become important now.

- **Aim:** Predict the context words $(w_s)_{s \neq t}$ knowing the center word w_t.
- Start from a collection of different sentences

$$\mathcal{C} = \{\mathtt{text} = (w_1, \ldots, w_T)\},$$

  to which positive probabilities are assigned

$$p(\mathtt{text}) = p(w_1, \ldots, w_T) > 0.$$

- These probabilities should reflect the frequencies of the sentences $\mathtt{text} = (w_1, \ldots, w_T)$ in speech and texts.

- Applying Bayes' rule, one determines how likely a certain context occurs for a given center word w_t

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

- In general, these probabilities are unknown, and they need to be learned from a learning sample $\mathcal{L}$.
- Learning these probabilities will be based on embedding the words into a low-dimensional embedding space; this is the step where the adjacency of the word embedding is learned.

word2vec: skip-gram approach

- A popular approach is the *word-to-vector* (word2vec) skip-gram approach of Mikolov, Chen, *et al.* (2013) and Mikolov, Sutskever, *et al.* (2013).
- Since this problem is too complex, one solves a simpler problem:
  1. First, one restricts to a fixed small *context (window) size* $c \in \mathbb{N}$
  $$p\left(w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c} \mid w_t\right).$$
  2. Second, one assumes *conditional independence* of the context words, given the center word w_t.
- **Remark.** Real texts do not satisfy this simplification, but this setup is still sufficient to obtain reasonable word embeddings.

- Under the conditional independence assumption, we have log-likelihood on the learning sample $\mathcal{L}$ and for given context size $c \in \mathbb{N}$

$$\ell_{\mathcal{L}} \;=\; \sum_{i=1}^{n} \sum_{t} \sum_{-c \leq j \leq c,\, j \neq 0} \log p\left(w_{i,t+j} \middle| w_{i,t} \right).$$

- Maximize this log-likelihood $\ell_{\mathcal{L}}$ in the conditional probabilities $p(\cdot|\cdot)$ to learn the most common context words of a given center word w_t.
- By embedding all words w, one can learn the embeddings $\boldsymbol{e}^{\mathrm{WE}}(w) \in \mathbb{R}^b$ by letting them enter the conditional probabilities $p(\cdot|\cdot)$ and maximizing the resulting log-likelihood.
- There is one special point: one needs *two different word embeddings* $\boldsymbol{e}^{(1)}(w) \in \mathbb{R}^b$ and $\boldsymbol{e}^{(2)}(w) \in \mathbb{R}^b$ for center and context words, as these two play different roles in the conditional probabilities.

- Assume the conditional probabilities are modeled by the softmax function

$$p\left(w_s \middle| w_t\right) = \frac{\exp\left\langle \boldsymbol{e}^{(1)}(w_t), \boldsymbol{e}^{(2)}(w_s)\right\rangle}{\sum_{w=1}^{W} \exp\left\langle \boldsymbol{e}^{(1)}(w_t), \boldsymbol{e}^{(2)}(w)\right\rangle} \in (0,1).$$

- If the scalar/dot product between $\boldsymbol{e}^{(1)}(w_t)$ and $\boldsymbol{e}^{(2)}(w_s)$ is large, there is a high probability that w_s is in the context of the center word w_t.
- Collecting everything, one receives the log-likelihood function

$$\ell_{\mathcal{L}} = \sum_{i=1}^{n} \sum_{t} \sum_{-c \le j \le c,\, j \neq 0} \log p\left(w_{i,t+j} \middle| w_{i,t}\right).$$

- Maximizing this log-likelihood $\ell_{\mathcal{L}}$ for the given learning sample $\mathcal{L}$ gives us the two (different) word embeddings.
- Optimization is done by variants of SGD.

Negative sampling

- The above word2vec skip-gram approach is computationally expensive.
- *Negative sampling* turns the above unsupervised learning problem into a supervised learning problem of a lower complexity; see Mikolov, Sutskever, *et al.* (2013).
- For this, we consider pairs $(w, \widetilde{w}) \in \mathcal{W} \times \mathcal{W}$ of center words w and context words $\widetilde{w}$. To each of these pairs we add a binary response variable $Y \in \{0, 1\}$, resulting in observation $(Y, w, \widetilde{w})$.
- There will be two types of center-context pairs:
  1. real ones are from the learning sample $\mathcal{L}$, and we set $Y = 1$, and
  2. fake ones that are generated purely randomly, and we set $Y = 0$.

- Construct these two types of pairs as follows:

  1. Extract all center-context pairs $(w, \widetilde{w})$ from the learning sample $\mathcal{L}$ and assign a response $Y = 1$ to these pairs, for indicating that these are true pairs. This gives the first part of the learning data denoted by

  $$\mathcal{L}_1 = (Y_i = 1, w_i, \widetilde{w}_i)_{i=1}^n.$$

  2. Take all real pairs $(w_i, \widetilde{w}_i)_{i=1}^n$, and randomly permute the index of the context word indicated by a permutation π. This gives a second (fake) learning data set

  $$\mathcal{L}_2 = (Y_{n+i} = 0, w_{n+i}, \widetilde{w}_{n+\pi(i)})_{i=1}^n,$$

  with $Y = 0$ as response.

- Merging real and fake learning data gives us a learning sample $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$ of sample size of $2n$.

- This turns into the supervised logistic regression problem

$$\begin{aligned}
\ell_{\mathcal{L}} &= \sum_{i=1}^{2n} \log \mathbb{P}\left[Y = Y_i \middle| w_i, \widetilde{w}_i \right] \\
&= \sum_{i=1}^{n} \log \left(\frac{1}{1 + \exp\langle -\boldsymbol{e}^{(1)}(w_i), \boldsymbol{e}^{(2)}(\widetilde{w}_i) \rangle} \right) \\
&\quad + \sum_{k=n+1}^{2n} \log \left(\frac{1}{1 + \exp\langle \boldsymbol{e}^{(1)}(w_k), \boldsymbol{e}^{(2)}(\widetilde{w}_k) \rangle} \right).
\end{aligned}$$

- Maximizing this log-likelihood $\ell_{\mathcal{L}}$, one can learn the two embeddings $\boldsymbol{e}^{(1)}$ and $\boldsymbol{e}^{(2)}$.

- For SGD training to work properly in this negative sampling learning, one should randomly permute the instances in $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$, to ensure that all (mini-)batches contain instances of both types.

Example: word2vec skip-gram with negative sampling

- We give an example being based on the claims texts of Frees (2020).

```
# load necessary packages
library(tensorflow)
library(keras)
library(data.table)
library(plyr)
library(stringr)
library(textstem)
library(tm)
library(purrr)

# load data
load(file="../Data/data_text.rda")
set.seed(100)
```

```
# remove stopwords, to lower case and remove white space at start and end
dat2 <- data_text %>% mutate(clean = Description %>%
↪  removeWords(stopwords("en")) %>% str_to_lower() %>% str_squish())
# remove numbers
dat2$clean <- str_squish(removeNumbers(dat2$clean))
# remove punctuation
dat2$clean <- str_squish(removePunctuation(dat2$clean,
↪  preserve_intra_word_contractions = FALSE, preserve_intra_word_dashes =
↪  TRUE ))
# remove damaged: because it occurs too frequently
dat2$clean <- str_squish(removeWords(dat2$clean, words=c("damage",
↪  "damaged", "damge", "dmage", "dmgd", "damged", "dmaged", "damgae",
↪  "damgae", "damaging")))
# lemmatize
dat2$clean <- lemmatize_strings(dat2$clean, dictionary =
↪  lexicon::hash_lemmas)
```

```r
# tokenize cleaned data
tokenizer1 = text_tokenizer() %>% fit_text_tokenizer(dat2$clean)
# count the number of used words
text.matrix <- texts_to_matrix(tokenizer1, dat2$clean, mode = "count")
length(colSums(text.matrix)[-1])
```

```
[1] 1819
```

```r
words.used <- colSums(text.matrix)[-1]
# minimal occurrence for embedding
wwww <- 20
(words <- length(words.used[words.used>=wwww]))
```

```
[1] 126
```

Since this is a very small dataset, we only embed the most frequent words, i.e., those that appear at least 20 times over all texts.

```
# we only embed the words that occur at least 20 times
tokenizer2 <- text_tokenizer(num_words=words+1) %>%
↪  fit_text_tokenizer(dat2$clean)
# this gives smaller texts
text.matrix <- texts_to_matrix(tokenizer2, dat2$clean, mode = "count")
# words used
(max.words <- length(colSums(text.matrix)[-1]))
```

```
[1] 126
```

```
# maximal sentence lengths
(maxlen <- max(rowSums(text.matrix)))
```

```
[1] 7
```

```
# tokenized sentences
seqs <- texts_to_sequences(tokenizer2, dat2$clean)
```

```
# true center-context pairs (we select a window size of 2)
jj0 <- 0
for (jj in 1:nrow(dat2)){
    # only consider texts with more than one word
    if (length(unlist(seqs[[jj]]))>1){
    jj0 <- jj0 + 1
    # generate the negative samples ourselves to control the seed
    tt <- skipgrams(sequence=unlist(seqs[[jj]]),
            vocabulary_size=words, window_size=2, negative_samples=0)
    xx <- matrix(unlist(tt$couples), ncol=2, byrow=TRUE)
    yy <- tt$labels
    gram0 <- data.frame(cbind(xx,yy))
    names(gram0) <- c("w1", "w2", "yy")
    if (jj0==1){
      gram <- gram0
      }else{
      gram <- rbind(gram, gram0)
      }}}
```

```r
skipgram <- gram
# generate fake center-context pairs by permuting the context word w2
gram$yy <- 0
set.seed(100)
gram$w2 <- gram[sample(1:nrow(gram)),"w2"]
# merge the two samples and randomize the order
skipgram <- rbind(skipgram, gram)
skipgram <- skipgram[sample(1:nrow(skipgram)),]
skipgram[1:5,]
```

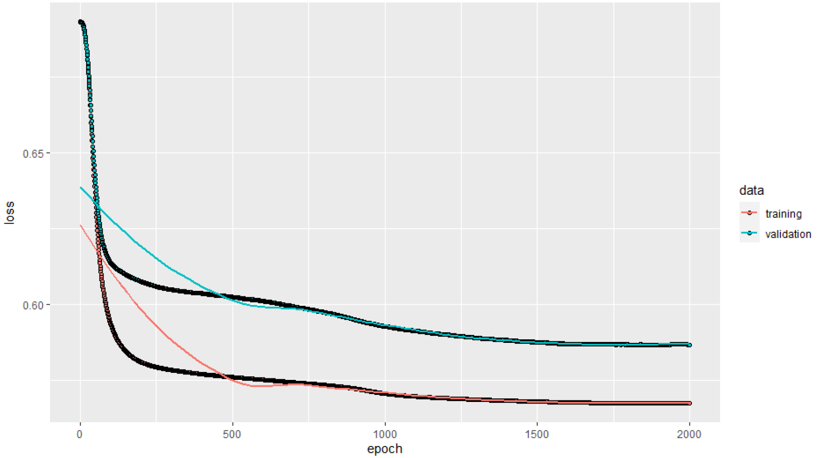
```
      w1 w2 yy
3779  52 91  1
2931  96 45  1
22557 12 36  0
8282  34 77  1
78     3 52  1
24227  1 93  0
```

```
network.word2vec <- function(seed, W1, b1){
    tf$keras$backend$clear_session()
    set.seed(seed)
    set_random_seed(seed)
    center  <- layer_input(shape = c(1), dtype = 'int32')
    context <- layer_input(shape = c(1), dtype = 'int32')
    centerEmb = center %>%
      layer_embedding(input_dim = W1, output_dim = b1, input_length = 1,
                      name = 'centerEmb') %>% layer_flatten()
    contextEmb = context %>%
      layer_embedding(input_dim = W1, output_dim = b1, input_length = 1,
                      name = 'contextEmb') %>% layer_flatten()
    response = list(centerEmb, contextEmb) %>%
               layer_dot(axes = 1) %>%
               layer_dense(units=1, activation='sigmoid')
    keras_model(inputs = c(center, context), outputs = c(response))
    }
```

```r
# center-context pairs input data
center  <- as.matrix(skipgram$w1-1)
context <- as.matrix(skipgram$w2-1)

# embedding dimension
b1 <- 2
model <- network.word2vec(seed=100, words, b1)
model %>% compile(loss = "binary_crossentropy", optimizer = "nadam")
```

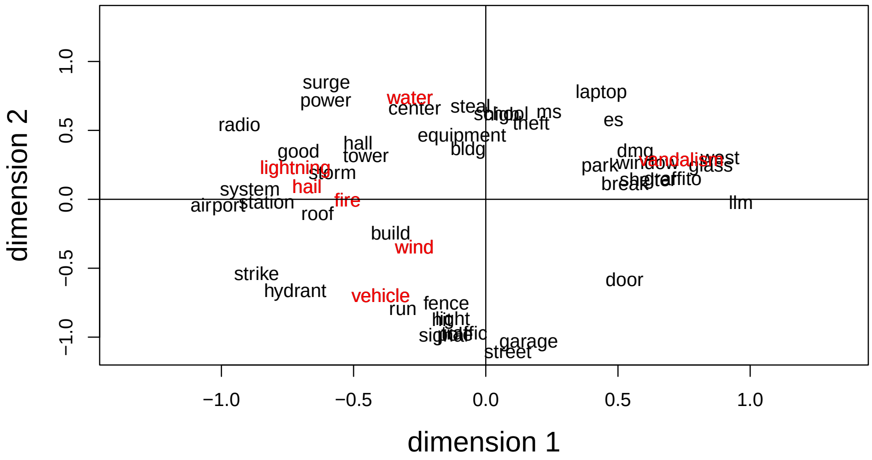
```
Model: "model"

___________________________________________________________________________
 Layer (type)               Output Shape                 Param    Connected to
                                                          #
===========================================================================
 input_1 (InputLayer)     [(None, 1)]                  0        []
 input_2 (InputLayer)     [(None, 1)]                  0        []
 centerEmb (Embedding)    (None, 1, 2)                 252      ['input_1[0][0]']
 contextEmb (Embedding    (None, 1, 2)                 252      ['input_2[0][0]']
 )
 flatten (Flatten)        (None, 2)                    0        ['centerEmb[0][0]']
 flatten_1 (Flatten)      (None, 2)                    0        ['contextEmb[0][0]']
 dot (Dot)                (None, 1)                    0        ['flatten[0][0]',
                                                                 'flatten_1[0][0]']
 dense (Dense)            (None, 1)                    2        ['dot[0][0]']
===========================================================================
Total params: 506 (1.98 KB)
Trainable params: 506 (1.98 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
fit <- model %>% fit(list(center, context),  skipgram$yy,
                     validation_split=0.2, batch_size=5000, epochs=epochs,
                     ↪  verbose=0)
```



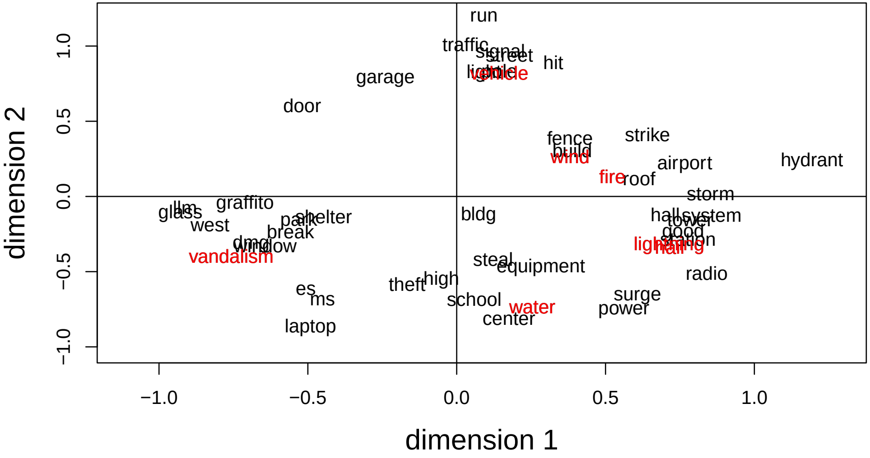
```
# function to extract the weights by layer name
get_embedding_values = function(layer_name){
  embedding = model %>% get_layer(layer_name) %>% get_weights()
  temp = embedding[[1]] %>% data.table()
  temp %>% setnames(names(temp),paste0("dim",seq(1:length(names(temp)))))
  temp}

# indices of hazards insured
hazard <- c(str_to_lower(sort(unique(dat2$Hazard))), "water")
index <- unlist(tokenizer2$word_index)[unlist(tokenizer2$index_word) %in%
↪  hazard]
index

# extract embedding of center words
variable <- "center"
embed_dims = get_embedding_values(paste(variable, "Emb", sep=""))

# we print the center word embeddings of the 50 most frequent words
```

2−dimensional embedding of center word



Red color shows the insured hazards. Naturally, we should select higher embedding dimensions, but $b = 2$ can nicely be illustrated.

2−dimensional embedding of context word



Conclusions word2vec

- Naturally, one should select higher-dimensional embedding dimensions than $b = 2$, and principal component analysis (PCA) can be used to illustrate these embeddings.
- Pre-trained embeddings can be downloaded, e.g., an embedding GloVe is available for embedding dimensions 50, 100, 200, 300. This is trained on large corpus of the internet; Pennington, Socher and Manning (2014).
- The difficulty with pre-trained embeddings is that they may be pre-trained in a different context, e.g., ‘policy’ may have different meanings in insurance and machine learning.

Copyright

References I

Bengio, Y., Courville, A. and Vincent, P. (2014) 'Representation learning: A review and new perspectives'. Available at: https://arxiv.org/abs/1206.5538.

Breiman, L. (1996) 'Bagging predictors', *Machine Learning*, 24, pp. 123–140. Available at: https://doi.org/10.1007/BF00058655.

Dietterich, T.G. (2000a) 'An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization', *Machine Learning*, 40, pp. 139–157. Available at: https://doi.org/10.1023/A:1007607513941.

References II

Dietterich, T.G. (2000b) ‘Ensemble methods in machine learning’, in *Multiple Classifier Systems: First International Workshop, MCS 2000, Lecture Notes in Computer Science*, pp. 1–15. Available at: https://doi.org/10.1007/3-540-45014-9_1.

Frees, E.W. (2020) ‘Loss data analytics’. Available at: https://github.com/OpenActTexts/Loss-Data-Analytics.

Mikolov, T., Sutskever, I., *et al.* (2013) ‘Distributed representations of words and phrases and their compositionality’. Available at: https://arxiv.org/abs/1310.4546.

References III

Mikolov, T., Chen, K., *et al.* (2013) 'Efficient estimation of word representations in vector space'. Available at: https://arxiv.org/abs/1301.3781.

Pennington, J., Socher, R. and Manning, C. (2014) 'GloVe: Global vectors for word representation', in A. Moschitti, B. Pang, and W. Daelemans (eds) *Proceedings of the 2014 conference on empirical methods in natural language processing*. Doha, Qatar: Association for Computational Linguistics, pp. 1532–1543. Available at: https://aclanthology.org/D14-1162.pdf.

Richman, R. and Wüthrich, M.V. (2020) 'Nagging predictors', *Risks*, 8(3). Available at: https://www.mdpi.com/2227-9091/8/3/83.

References IV

Wüthrich, M.V. *et al.* (2025) 'AI Tools for Actuaries', *SSRN Manuscript* [Preprint]. Available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5162304.