ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

information
systems

Business Intelligence

– SEMINAR WINTER SEMESTER 2013/2014 –

Detecting Negation Scopes using Hidden Markov Models

– SEMINAR PAPER –

Submitted by:
Nicolas Pröllochs

Advisor:
Prof. Dr. Dirk Neumann

Contents

1 Introduction 1

2 Related Work 1

3 Methods 4
- 3.1 Hidden Markov Models 4
- 3.2 Baum-Welch Algorithm 6

4 Experimental Setup 8
- 4.1 Dataset 8
- 4.2 Implementation 8

5 Results 9

6 Summary 11

A Source Code i

B References vi

C List of Figures vii

D List of Tables viii

Abstract

Detection of negated parts in written sentences is often complex and difficult to determine by a rule-based algorithm. Therefore, it can be useful to apply a machine learning approach to predict negated parts within sentences. Because of their flexibility and their comprehensive learning possibilities, Hidden Markov models can be a suitable option for such a task. This paper provides an overview of this machine learning approach and tests various types of Hidden Markov models to find negated parts in sentences from financial news. In order to estimate the appropriateness of different implementations, we compare our results with a manually labeled dataset.

1 Introduction

Human readers recognize a sentence as negated usually intuitively or by applying grammatical rules. A large part of the sentiment of a sentence depends on categorizing possible negated parts correctly. Also in sentiment analysis, it can be necessary to know which parts of a sentence are negated and which are not. In order to evaluate sentences automatically, rules are necessary to classify the negated phrases.

Not only single words, but also phrases in a sentence are often negated by a simple negation word like “not” e. g. in “The economy has *not* grown”, while it is not always that ordinary. A closer examination of negation in sentences makes clear that such a simple approach does not cover all variations. In addition, negation in a sentence can be implicit, e. g. “The company has invented a new product, it was the first and last time”. For an algorithm, it is difficult to consider these linguistic cases, not to mention peculiarities such as sarcasm or irony. Therefore, it can be senseful to use a machine learning approach and try to train a model which is able to predict the negation scope out of experiences made in other sentences.

This paper now deals with the detection of negation scopes in sentences from financial news using Hidden Markov models. We implement different variants of Hidden Markov models, in order to predict the negated parts of a sentence. In addition, we include supervised and unsupervised learning to train our model and calculate the impact of different implementations on recall, precision, F-score and accuracy of the forecast. For this task, we create a program which is listed in Appendix A.

First, we give an overview of related literature which deals in a similar way with the Hidden Markov models approach (section 2) or negation scope prediction. In the next step, we explain the basic structure and methodology of Hidden Markov models (section 3). Afterwards, we illustrate our experimental setup and the way we implement suitable Hidden Markov models for our dataset (section 4). In connection, we evaluate our experiment and discuss the results (section 5). Section 6 closes with a summary of the main results.

2 Related Work

Hidden Markov models can be found in a wide field of research. Sonnhammer, Heijne, Krogh, et al. achieve good results in the field of biology for prediction of transmembrane helices in protein sequences [6]. Varga and Moore use Hidden Markov models for noise reduction in speech recognition and reach significant improvements [8]. The impact of negations in sentiment

analysis is investigated by Chapman et al., who make use of machine learning and find good results in fields with short, concise sentences [3]. Lexicon-based researches by Dadvar, Hauff, and Jong encounter problems particularly with indirect negations and stylistic devices like sarcasm and metaphors [4].

This paper now uses the machine learning approach to forecast the negation scope of written sentences in financial news. For this purpose, we implement several variations of Hidden Markov models and train them through supervised and unsupervised learning in order to predict negated parts in sentences. We use a manually-prepared corpus of financial market news, where we marked each word as negated or not negated, which serves as our gold standard. Here, in comparison to other studies like from Chapman et al., we also incorporate word types and word stems to improve our forecast. Table 1 illustrates the comparison between related literature and this paper.

Literature	Domain	Investigation object	Experimental data	Gold standard	Methodology
Chapman et al. 2001 [3]	Clinical	Identification of diseases in clinical information from patient databases	1500 medical reports	Expert reviews	Implementation of machine learning to find negated expressions
Dadvar, Hauff, and Jong 2011 [4]	Consumer opinions	Classification of negation scopes in user reviews on a online portal	2000 reviews, thereof 1000 positive and 1000 negative	Consumer ratings on an online portal	Lexicon-based, experiments for different negation windows
Sonnhammer, Heijne, Krogh, et al. 1998 [6]	Chemical	Prediction of transmembrane helices in protein sequences in the field of biology	A set consisting of 83 proteins	Analysis with biochemical and genetic methods	Baum-Welch algorithm
Varga and Moore 1990 [8]	Defense reports	Automatic speech recognition in the presence of interfering signals and noise reduction	Speech and noise data, extracted from the NATO RSG-10 database	Known speech data from NATO RSG-10	Baum-Welch algorithm
This paper	Textual	Prediction of negation scopes in sentences from financial news	590 sentences from financial news	Manually annotated	Baum-Welch algorithm and Viterbi algorithm with words, stems and part of speech

Table 1: Related literature incorporating Hidden Markov models to predict negation scopes.

3 Methods

In this chapter, we describe the used methods to determine the negation scope using Hidden Markov models. First of all, we create a test corpus consisting of sentences from financial market news and their associated part of speech. Next, we manually mark the negated words in each sentence, which serves as our gold standard. Afterwards, we implement several variants of Hidden Markov models. Subsequently, we evaluate our model and calculate recall, precision, F-score and accuracy of our forecast in context with our gold standard. Figure 1 shows that in a flow diagram.

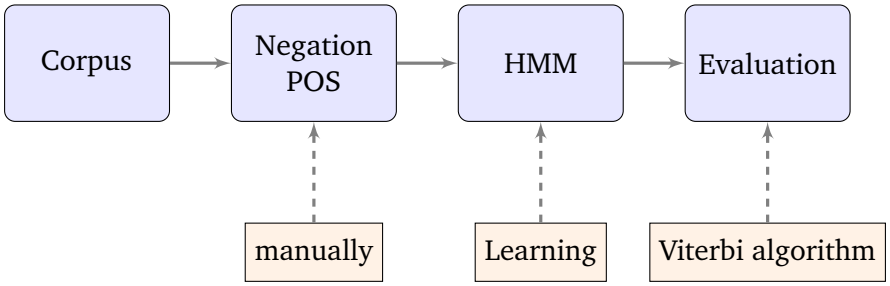

Figure 1: Flow diagram of HMM-based prediction of negation scopes.

3.1 Hidden Markov Models

In some cases, the true states of something we want to investigate are not directly observable, but there are possible effects of these states, which can be observed. Such a system can be described as an Hidden Markov model and is defined by the parameters in the following definition [5].

Definition: An Hidden Markov model is a generative probabilistic model which consists of N not directly observable states and M distinct observation symbols per state, i. e. the size of the emission alphabet. We denote the individual states as $S = \{S_1, S_2, ..., S_N\}$, the observation symbols as $V(t) = \{v_1, v_2, ..., v_M\} \forall t$ and the current state at t as q_t. An Hidden Markov model contains a state transition probability distribution $A = \{a_{ij}\}$, where

$$a_{ij} = P[q_{t+1} = S_j \mid q_t = S_i], \quad 1 \leq i \leq N,\ 1 \leq j \leq N, \tag{1}$$

an observation symbol probability distribution $B = \{b_j(k)\}$ in state S_j, where

$$b_j(k) = P[V_k(\text{at } t) \mid q_t = S_j], \quad 1 \leq j \leq N,\ 1 \leq k \leq M, \tag{2}$$

and an initial distribution $\pi = \{\pi_i\}$, where

$$\pi_i = P[q_1 = S_i], \quad 1 \leq i \leq N. \tag{3}$$

A complete HMM requires specification of all the parameters above. The compact notation reads as

$$\lambda = (A, B, \pi). \tag{4}$$

We are confronted with a double stochastic process. After each period t, the current state can remain or move to another state. Each state $S = \{S_1, S_2, ..., S_N\}$ can reach each other state in one step, that means $a_{ij} > 0$ (for other types of HMM, it is possible to have $a_{ij} \geq 0$). Figure 2 shows that circumstances graphically. As the second stochastic process, each of these states emit a random visible symbol O at step t of the emission alphabet V (Figure 3). In time T, the model generates a sequence of observable states $O = \{O_1, O_2, ..., O_T\}$. Consequently, there are the following three basic problems along with Hidden Markov models.

Evaluation: Computing the probability of an observation sequence $O = \{O_1, O_2, ..., O_T\}$ for a model $\lambda = (A, B, \pi)$ (section 3.1.1).

Decoding: Computing the corresponding state sequence $Q = \{q_1, q_2, ..., q_T\}$ to an observation O and a model λ (section 3.1.2).

Learning: Adjustment of the parameters $\lambda = (A, B, \pi)$ to maximize $P(O \mid \lambda)$ (section 3.2).

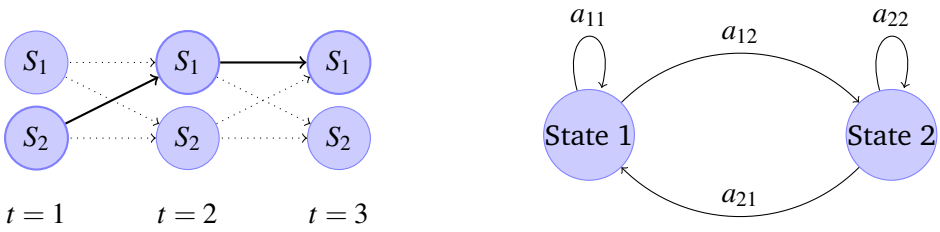

Figure 2: Illustration of transition proceedings for an HMM with two states.

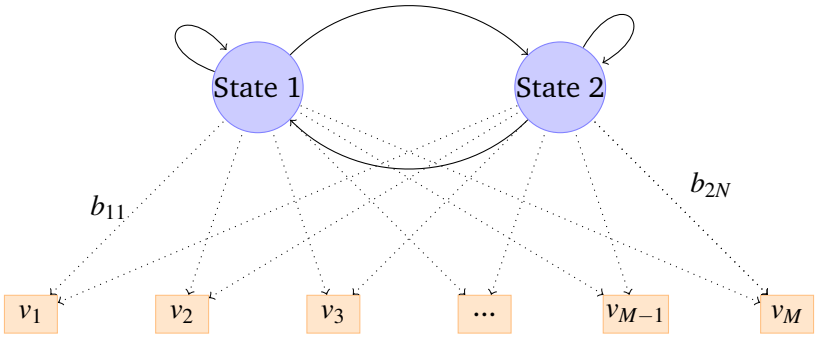

Figure 3: Illustration of emission proceedings for an HMM with two states.

3.1.1 Forward/Backward Algorithm

We can compute the probability for an observation O and a model λ, by summing the joint probability over all possible state sequences via

$$P(O \mid Q) = \sum_{\forall q} P(O \mid Q, \lambda) P(Q \mid \lambda). \tag{5}$$

This can be considered as the probability that λ has produced the observable sequence O. Equation (5) requires $2T \cdot N^T$ calculations (even if we assume only 4 states and 100 observations, we need $2 \cdot 100 \cdot 4^{100} \approx 3 \cdot 10^{62}$ computations), which clarifies that we need a more efficient way to

solve the problem. In order to compute the probabilities for our model efficiently, we solve the problem by dynamic programming [1]. We define

$$\alpha_i(t) = P(O_1,O_2,...,O_t,q_t = S_i \mid \lambda), \tag{6}$$

as the probability of being in state S_i at step t while emitting $\{O_1,O_2,...O_t\}$.

At step $t = 1$, we start with the initial probabilities π, and calculate $\alpha_i(1)$, which is the probability that the model is located at state S_i in $t = 1$ while emitting O_1 depending on the initial probabilities. In the next step, we can compute the probability to move from a state S_i in t to S_j in step $t+1$ while emitting O_{t+1}. In the following, we can solve the problem for the whole observation recursively, by

$$\alpha_i(t) = \begin{cases} \pi_i b_i(O_1), & \text{if } t = 0, \\ \left[\sum_{i=1}^{N} \alpha_i(t-1)a_{ij}\right] b_j(O_t), & \text{otherwise.} \end{cases}$$

The variable $\alpha_i(t)$ denotes the probability to reach state S_i at step t while emitting O_t if the model has already produced $t-1$ elements of O. The total probability of generating the observable sequence O is determined by the sum of $\alpha_i(T)$ over N. In addition, we can calculate a backward variable $\beta_i(t)$ which gives us the probability of the partial observation from $t+1$ to T given state S_i at step t and the model λ. We are able to solve the problem for $\beta_i(t)$ inductively via

$$\beta_i(t) = \begin{cases} 1, & \text{if } t = T, \\ \sum_{i=1}^{N} a_{ij} b_j(O_{t+1})\beta_j(t+1), & \text{otherwise.} \end{cases}$$

3.1.2 Viterbi Algorithm

To find the most probable corresponding state sequence to a given observation O, we choose the states Q which maximize the expected number of correct individual states, i. e.

$$\gamma_i(t) = P(q_t = S_i \mid O,\lambda), \quad \sum_{i=1}^{N} \gamma_i(t) = 1, \tag{7}$$

calculates the probability of being in state S_i at step t given the observation sequence O and the model λ. Using γ, we can compute the most probable state q_t at step t, by

$$q_t = \underset{1 \le i \le N}{\arg\max} \left[\gamma_i(t)\right], \quad 1 \le t \le T. \tag{8}$$

3.2 Baum-Welch Algorithm

The Baum-Welch algorithm is an iterative procedure to adjust the model parameters (A,B,π), in order to maximize the probability of an observation sequence O. Given an HMM and an observation sequence, we can calculate the probability

$$\xi_{i,j}(t) = \frac{\alpha_i(t)a_{ij}b_j(O_{t+1})\beta_j(t+1)}{P(O \mid \lambda)}, \tag{9}$$

of being in state S_i at step t and passing over to state S_j at step $t+1$ using the forward and backward algorithm. Thereby, we are able to calculate the in Equation (7) defined probability $\gamma_i(t)$ of being in state S_i at step t by summing $\xi_{i,j}(t)$ over j via

$$\gamma_i(t) = \sum_{j=1}^{N} \xi_{i,j}(t). \tag{10}$$

If we sum up $\gamma_i(t)$ over time T, we get the number of transitions made from state S_i. Equally, summation of $\xi_{i,j}(t)$ over T can be interpreted as the expected number of transitions from state S_i to state S_j. Using ξ and γ, we can reestimate the parameters for an HMM as shown in Equation (11), by

$$\hat{\pi}_i = \text{Expected number of times in state } S_i \text{ at step } t = 1 = \gamma_i(1), \tag{11a}$$

$$\hat{a}_{ij} = \frac{\text{Expected number of transitions from state } S_i \text{ to state } S_j}{\text{Expected number of transitions from state } S_i} = \frac{\sum_{t=1}^{T-1} \xi_{i,j}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)}, \tag{11b}$$

$$\hat{b}_j(k) = \frac{\text{Exp. number of times in state } S_j \text{ and observing symbol } v_k}{\text{Exp. number of times in state } S_j} = \frac{\sum_{t \in \{\tau | O(\tau) = v_k\}}^{T} \gamma_j(t)}{\sum_{t=1}^{T} \gamma_j(t)}. \tag{11c}$$

To adjust the parameters of the HMM, we repeat this procedure iteratively until the likelihood function of the model converges. Algorithm 1 shows the Baum-Welch algorithm in pseudocode. It has been proven [2] that the algorithm leads to an increasing likelihood after each iteration I, i. e. $P(O \mid \hat{\lambda}_{I+1}) \geq P(O \mid \lambda_I)$.

Algorithm 1: Baum-Welch algorithm.

Data: a_{ij} and b_{jk} and V and convergence criterion.
begin
  repeat
    $z \leftarrow z + 1$
    $\hat{a}(z) \leftarrow \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(j)}$
    $\hat{b}(z) \leftarrow \frac{\sum_{t \in \{\tau | O(\tau) = v_z\}}^{T} \gamma_j(t)}{\sum_{t=1}^{T} \gamma_j(t)}$
    $\hat{\pi}(z) \leftarrow \gamma_1(i)$
    $a_{ij} \leftarrow \hat{a}_{ij}(z-1)$
    $b_{jk} \leftarrow \hat{b}_{jk}(z-1)$
    $\pi_i \leftarrow \hat{\pi}_i(z-1)$
  until *convergence criterion achieved*
  return a_{ij} *and* b_{jk}
end

4 Experimental Setup

This part describes the used dataset and the adaption of a suitable Hidden Markov Model for our experiments.

4.1 Dataset

We use a dataset consisting of 590 negated sentences from financial market news. Each of these sentences contains at least one negation phrase. The used phrases are shown in Table 2b. In the first step, we assign each word in an observation sequence his corresponding part of speech value. For this task, we use a limited amount of word types, which are listed in Table 2a.

Nouns	Numerals
Adjectives	Articles
Adverbs	Conjunctions
Verbs	Pronouns
Prepositions	Interjections
Negations	

(a) Used word classes.

no	hardly
not	denied/denies
*n't	without
rather	

(b) Used negation phrases.

Table 2: Used word classes and negation phrases in our dataset.

In addition, we manually label each word in a sentence as negated or not negated. Our dataset contains 16302 words, where $\approx 51\,\%$ of these words are negated. In summary, each observation consists of a sequence of words, each of them associated with a word type and negation mark, as shown below.

Word: The word itself, e. g. "successfull".

Part of speech: Associated word type, e. g. "successfull" marked as adjective.

Negation: Decision variable, e. g. "successfull" marked as true after "not".

4.2 Implementation

We specify an Hidden Markov model assuming two not directly observable states $S = \{\text{Negated}, \neg \text{Negated}\}$. Each of this states emits as an observable symbol O at step t a part of speech out of the emission alphabet V. An observation in our model consists of the emitted word types from one sentence. Figure 4 illustrates this setup.

In the next step, we determine the transition probabilities and emission probabilities using the data from our manually annotated dataset. To avoid overfitting, we use varying parts of the data to train our model and the respective rest of the data for the evaluation. In this context, we divide our dataset into several folds and take advantage of 10-fold cross validation [7]. For that, we rotate the training part of the HMM to all parts of the dataset and evaluate recall, precision and accuracy for the respective rest.

We also specify an HMM assuming the original words as emission symbols. Equally, we assume the two hidden states $S = \{\text{Negated}, \neg \text{Negated}\}$ and several probabilities of each state to emit, in this case, an observable word value O at step t.

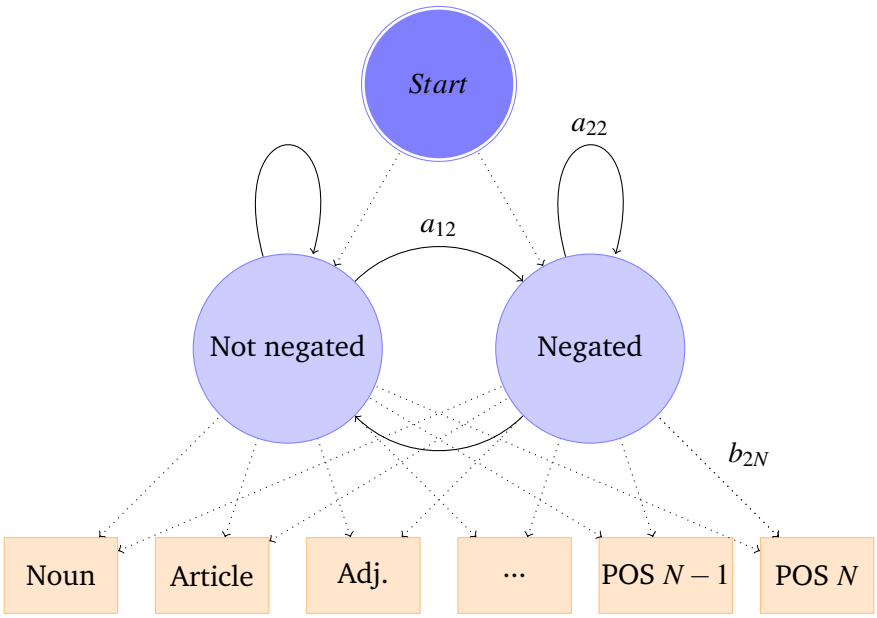

Figure 4: Illustration of a fitted Hidden Markov model for our experiment.

In addition, we implement the Baum-Welch algorithm, whereby we try to determine the transition probabilities and emission probabilities by unsupervised learning. Also in this case, we evaluate the results using the 10-fold cross validation approach in connection with the Viterbi algorithm to calculate precision, accuracy and recall for the whole dataset.

Along with our objective, which is to determine the scope of negation as precisely as possible, we implement a number of different tasks and determine their impact on precision, accuracy and recall.

Task 1 Implementation of supervised learning and usage of word types for negation scope prediction.

Task 2 Implementation of supervised learning and usage of words for negation scope prediction.

Task 3 Implementation of supervised learning and usage of stemmed words for negation scope prediction.

Task 4 Implementation of unsupervised learning and usage of word types for negation scope prediction.

Task 5 Implementation of unsupervised learning and usage of words for negation scope prediction.

Task 6 Implementation of unsupervised learning and usage of stemmed words for negation scope prediction.

5 Results

This section describes the determined results, subject to different approaches and lists them up. We start our studies with supervised learning and train our corpus using 10-fold cross validation based on word types. Afterwards, we use the Viterbi algorithm to calculate recall, precision,

accuracy and F-score of our results for the respective rest of the dataset (task 1). In the next step, we use the real words as basis for our HMM and also evaluate the model using the cross validation approach (task 2). In connection, we test the model with the stems of the words (task 3). Thereafter, we implement unsupervised learning using the Baum-Welch algorithm [1]. In this case, we learn an Hidden Markov model with the assigned part of speech. Subsequently, we evaluate the Viterbi algorithm calculations for the rest of the dataset (task 4). With task 5, we expand the unsupervised learning approach using the word values as basis for the Baum-Welch algorithm and the evaluation. Finally, we repeat this attempt using the word stems in order to implement a suitable HMM for negation scope prediction. The results for the tasks described above are listed in table 3.

Task	**Type**	**Data**	**Recall**	**Precision**	**Accuracy**	***F*-score**
1	Supervised	POS	0.7583542	0.6849005	0.7076915	0.7197582
2	Supervised	Word	0.4884480	0.8408996	0.6937186	0.6179509
3	Supervised	Stem	0.5348978	0.7921892	0.6929825	0.6386021
4	Unsupervised	POS	0.1830618	0.5687865	0.5269306	0.2769790
5	Unsupervised	Word	0.5098162	0.5208022	0.5251732	0.5152507
6	Unsupervised	Stem	0.5053258	0.5321091	0.5352011	0.5183717

Table 3: Results for different implementations of the tasks from section 4.2.

Starting with supervised learning, we use word types for the prediction of negated words in a sentence (task 1). We can reach an accuracy of 0.708 and a F-Score of 0.720 using 10-fold cross validation. Afterwards, we implement task 2, where we use real words as emission symbols. In this experiment, we reach less accurate results than in the previous task. We reach a F-Score of 0.618 which is mainly caused by the fact that the training shares do not include all the emission symbols of the whole dataset. The accuracy of the results is also smaller than in the previous task, we calculate a value of 0.694. In addition, we repeat the experiment for word stems as emission symbols (task 3). Although the emission alphabet is smaller in this case, we do not find much better results than in task 2. The calculated F-Score is located at 0.639 and the accuracy is 0.693.

Subsequently, we introduce unsupervised learning using the Baum-Welch algorithm. We start with task 4, where we use word types as observation data. We are able to attain an accuracy of 0.527 and a F-Score of 0.277. After that, we implement task 5 and use the real words as basis for our HMM. With this variant, we are able to receive an accuracy of 0.525 and a F-Score of 0.515. Finally, we use word stems as emission symbols for our model (task 6) and reach an accuracy of 0.535 and a F-Score of 0.518. In the unsupervised experiments, we are able to reach higher recall values for words and stems than for word types as basis for the HMM. The accuracy values for the different bases are quite similar. For all the unsupervised learning tasks, we reach less accurate results than for the corresponding supervised tasks. In addition, experiments with word stems as emission symbols in comparison to word-based experiments do not lead to significant improvements.

It becomes apparent that an HMM which uses supervised learning and word types as observation data is most suitable for our experiment. This is mainly based on the relatively small dataset.

If we use real words or word stems as observation data, we are confronted with the problem that the words in our dataset are too infrequent. There is a amount of words in our dataset which is located in the dataset, but not in the respective training part. The supervised learning approach leads to much better results than the unsupervised learning approach. We find much higher values for accuracy and F-Score which can be explained by the fact that the data is too unspecific. Particularly in the case of using word types as observation data, the Baum-Welch algorithm is not able to find a sufficient adjustment for our dataset.

In general, Hidden Markov models are a suitable option for the task of negation scope prediction, but require a large amount of data. Presumably, the results can be improved, if we use a more specific and larger dataset. In addition, a combination of a rule-based algorithm and a machine learning approach is recommendable.

6 Summary

We implement several variants of Hidden Markov models to recognize negated parts of sentences. For our experiments, we use a dataset which consists of 590 sentences from financial news. In the field of supervised learning, we can attain values for accuracy of 0.708 and a F-Score of 0.720 for word types as observable symbols. We do not find large differences between the implementation with real words and the implementation with word stems. The found accuracy is 0.694 and the F-Score is located at 0.618 for real words. Using unsupervised learning, i. e. the Baum-Welch algorithm, we are able to reach an accuracy value of 0.527 and a F-Score of 0.277 for word types as emission symbols. Also in this case, the reached results for real words and word stems are quite similar. The found accuracy for real words is 0.525 and the F-Score 0.515.

It became clear that our supervised learning implementation is much more reliable than the unsupervised learning implementation, which is not able to predict the negation scope sufficiently. In addition, we find the best results for word types as observable symbols using the supervised learning approach. Our algorithm offers easy exchange options for other datasets and part of speech modifications, still, in further analysis, the dataset should be expanded and consider both machine learning and grammatical rules.

A Source Code

A.1 Main.R

```
library(HMM,RHmm)
source("code/ReadDataset.R")
source("code/Task.R")
source("code/Results.R")
source("code/Analysis.R")
source("code/Training.R")

tasks<-list(new("Task",Supervised=T,Base="word_types",training_ratio=0.1,ID=1),
            new("Task",Supervised=T,Base="words",training_ratio=0.1,ID=2),
            new("Task",Supervised=T,Base="stems",training_ratio=0.1,ID=3),
            new("Task",Supervised=F,Base="word_types",training_ratio=0.1,ID=4),
            new("Task",Supervised=F,Base="words",training_ratio=0.1,ID=5),
            new("Task",Supervised=F,Base="stems",training_ratio=0.1,ID=6))

dataset<-ReadDataset(path="Dataset/reuters1.csv")
results<-list()

for (i in 1:length(tasks)) {
  results[[length(results)+1]]<-Scope_Analysis(task=tasks[[i]],dataset=dataset)
}
rm(i)
```

A.2 ReadDataset.R

```
setClass("Dataset",representation = representation(
  data_neg="list",
  data_text="list",
  data_stems="list",
  data_wc="list"))

ReadDataset<-function(path) {
  Dataset<-read.csv(path, header = F, sep=";")
  sentences<-as.matrix(Dataset[,1])
  stems<-as.matrix(Dataset[,2])
  wc<-as.matrix(Dataset[,3])
  negations<-as.matrix(Dataset[,4])

  data_text<-list()
  data_stems<-list()
  data_wc<-list()
  data_neg<-list()

  for (line in 1:dim(sentences)[1]) {
    data_text[line]<-c(strsplit(sentences[line,],"[[:space:]]+"))
    data_stems[line]<-c(strsplit(stems[line,],"[[:space:]]+"))
    data_wc[line]<-c(strsplit(wc[line,],"[[:space:]]+"))
    data_neg[line]<-c(strsplit(negations[line,],"[[:space:]]+"))
    data_neg[[line]]<-as.logical(data_neg[[line]])
  }

  rm(line,sentences,stems,wc,negations,Dataset)
```

```
    db<-new("Dataset", data_text=data_text, data_neg=data_neg, data_stems=data_stems, ↩
      data_wc=data_wc)
    return(db)
}

CreateHMM<-function(dataset,task) {
  bases<-factor(c("words","stems","word_types"))
  if(task@Base == bases[1]) {
    eleB<-unique(c(unlist(dataset@data_text)))
  } else if(task@Base == bases[2]) {
    eleB<-unique(c(unlist(dataset@data_stems)))
  } else if(task@Base == bases[3]){
    eleB<-unique(c(unlist(dataset@data_wc)))
  }

  eleA<-unique(c(unlist(dataset@data_neg)))
  A_T<-matrix(0,length(eleA),length(eleA))
  colnames(A_T)<-eleA
  rownames(A_T)<-eleA
  B_T<-matrix(0,length(eleA),length(eleB))
  colnames(B_T)<-eleB
  rownames(B_T)<-rownames(A_T)
  Phi_T<-A_T[,1]
  rm(eleA,eleB)

  hmm = initHMM(rownames(A_T),colnames(B_T),
                startProbs=Phi_T,
                transProbs=A_T,
                emissionProbs=B_T)

  return(hmm)
}
```

A.3 Task.R

```
setClass("Task",representation = representation(
  ID="numeric",
  Supervised="logical",
  Base="character",
  training_ratio="numeric"),
        prototype = prototype(ID=0,Supervised=T))
setMethod("show",signature(object="Task"),
          function(object) {
            cat("----------------------------------------\n")
            cat(toString(Sys.time()),"\n")
            cat("Supervised: ",toString(object@Supervised),"\n")
            cat("Base: ",toString(object@Base),"\n")
            cat("Training ratio: ",toString(object@training_ratio),"\n")
            cat("----------------------------------------\n")
          })
```

A.4 Results.R

```
setClass("Result",representation = representation(
                                          Precision="numeric",
```

```
                                                    Recall="numeric",
                                                    Fmeasure="numeric",
                                                    Accuracy="numeric",
                                                    Prediction="list",
                                                    TrueValues="list",
                                                    Task="Task"),
          prototype = prototype(Fmeasure=0))

setMethod("show",signature(object="Result"),
           function(object) {
             cat("----------------------------------------\n")
             cat(toString(Sys.time()),"\n")
             cat("Result for task with ID: ",object@Task@ID,"\n")
             cat("With base: ",object@Task@Base,"\n")
             cat("Supervised: ",toString(object@Task@Supervised),"\n")
             cat("CV ratio: ",toString(object@Task@training_ratio),"\n\n")

             cat("Recall: ",object@Recall,"\n")
             cat("Precision: ",object@Precision,"\n")
             cat("F-measure: ",object@Fmeasure,"\n")
             cat("Accuracy: ",object@Accuracy,"\n")
             cat("----------------------------------------\n")
           })

Evaluate_results<-function (prediction,trueValues,task) {
  precision<-sum(as.logical(unlist(prediction)) & unlist(trueValues))
    /sum(as.logical(unlist(prediction)))
  recall <- sum(as.logical(unlist(prediction)) & unlist(trueValues)) /
    sum(unlist(trueValues))
  Fmeasure <- 2 * precision * recall / (precision + recall)
  accuracy <- (sum(as.logical(unlist(prediction)) & unlist(trueValues))
               + sum(!as.logical(unlist(prediction)) & !unlist(trueValues)))
                 /length(unlist(trueValues))
  result<-new("Result")
  result@Recall<-recall
  result@Precision<-precision
  result@Fmeasure<-Fmeasure
  result@Accuracy<-accuracy
  result@Prediction<-prediction
  result@TrueValues<-trueValues
  result@Task<-task
  return(result)
}
```

A.5 Training.R

```
Training<-function(hmm,data_true,data_base) {

  for (x in colnames(hmm$emissionProbs)) {
    n<-which(unlist(data_base)==x)
    hmm$emissionProbs[1,which(colnames(hmm$emissionProbs)==x)] =
      sum(!unlist(data_true)[n])
    hmm$emissionProbs[2,which(colnames(hmm$emissionProbs)==x)] =
      sum(unlist(data_true)[n])
  }

  hmm$startProbs[1]<-sum(!sapply(data_true,function(x) x[1]))
  hmm$startProbs[2]<-sum(sapply(data_true,function(x) x[1]))
```

```
    j<-sapply(data_true,function(x) x[-1])
    h<-sapply(data_true,function(x) x[-length(x)])
    hmm$transProbs[1,1]<-sum(!unlist(h)&!unlist(j))
    hmm$transProbs[1,2]<-sum(!unlist(h)&unlist(j))
    hmm$transProbs[2,1]<-sum(unlist(h)&!unlist(j))
    hmm$transProbs[2,2]<-sum(unlist(h)&unlist(j))

    hmm$startProbs<-hmm$startProbs / sum(hmm$startProbs)
    for (i in 1:dim(hmm$transProbs)[1]) {
      hmm$transProbs[i,]<-hmm$transProbs[i,]*1/sum(hmm$transProbs[i,])
    }
    for (i in 1:dim(hmm$transProbs)[1]) {
      hmm$emissionProbs[i,]<-hmm$emissionProbs[i,]*1/sum(hmm$emissionProbs[i,])
    }
    rm(i)
    return(hmm)
  }
```

A.6 Analysis.R

```
  chunk <- function(x,n){
    numOfVectors <- floor(length(x)/n)
    elementsPerVector <- c(rep(n,numOfVectors-1),n+length(x) %% n)
    elemDistPerVector <- rep(1:numOfVectors,elementsPerVector)
    split(x,factor(elemDistPerVector))
  }

  Scope_Analysis<-function(task,dataset) {
    if (task@Supervised) {
      require(HMM)
    } else {
      if (require(HMM, quietly=TRUE)) {
        detach("package:HMM", unload=TRUE)
      }
      require(RHmm)
    }

    bases<-factor(c("words","stems","word_types"))
    if (task@Base == bases[1]) {
      data_base<-dataset@data_text
    } else if (task@Base == bases[2]) {
      data_base<-dataset@data_stems
    } else if (task@Base == bases[3]) {
      data_base<-dataset@data_wc
    } else {
      stop("Base does not exist!")
    }

    n<- floor(length(dataset@data_neg)*task@training_ratio)
    numvectors<-floor(length(dataset@data_neg)/n)

    data_base_chunks<-chunk(data_base,n)
    data_neg_chunks<-chunk(dataset@data_neg,n)

    prediction<-list()
    trueValues<-list()

    if (task@Supervised) {
      hmm<-CreateHMM(dataset,task)
```

```
    for (part in 1:numvectors) {

      hmm_trained<-Training(hmm, unlist(data_neg_chunks[-part],recursive=F),
        unlist(data_base_chunks[-part], recursive=F))

        for (i in 1:length(data_neg_chunks[[part]])) {
          vit<-viterbi(hmm_trained,data_base_chunks[[part]][[i]])
          prediction[[length(prediction)+1]]<-vit
          trueValues[[length(trueValues)+1]]<-data_neg_chunks[[part]][[i]]
        }
        rm(i,vit)
      }

  } else {
    for (part in 1:numvectors) {
      hmm_trained<-HMMFit(unlist(data_base_chunks[-part], recursive=F),
        dis="DISCRETE", nStates=2, control=list(nInit=500))

        for (i in 1:length(data_neg_chunks[[part]])) {
          vit=viterbi(hmm_trained,data_base_chunks[[part]][[i]])$states
          prediction[[length(prediction)+1]]<-as.logical(vit-2)
          trueValues[[length(trueValues)+1]]<-data_neg_chunks[[part]][[i]]
        }
        rm(i,vit)
    }
  }

  result<-Evaluate_results(prediction,trueValues,task)
  capture.output(show(result),append=TRUE,file="Output/output.txt")
  return(result)
}
```

B References

[1] L. E. BAUM. *An equality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes*. In: *Inequalities*, Vol. 3 (1972), pp. 1–8.

[2] L. E. BAUM and G. R. SELL. *Growth transformations for functions on manifolds*. In: *Pacific J. Math*, Vol. 27, No. 2 (1968), pp. 211–227.

[3] W. W. CHAPMAN et al. *A simple algorithm for identifying negated findings and diseases in discharge summaries*. In: *Journal of biomedical informatics*, Vol. 34, No. 5 (2001), pp. 301–310.

[4] M. DADVAR, C. HAUFF, and F. DE JONG. *Scope of negation detection in sentiment analysis*. In: (2011).

[5] L. R. RABINER. *A tutorial on hidden Markov models and selected applications in speech recognition*. In: *Proceedings of the IEEE*, Vol. 77, No. 2 (1989), pp. 257–286.

[6] E. L. SONNHAMMER, G. VON HEIJNE, A. KROGH, et al. *A hidden Markov model for predicting transmembrane helices in protein sequences.* In: *Ismb*. Vol. 6. 1998, pp. 175–182.

[7] M. STONE. *Cross-validatory choice and assessment of statistical predictions*. In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1974), pp. 111–147.

[8] A. VARGA and R. MOORE. *Hidden Markov model decomposition of speech and noise*. In: *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on*. IEEE. 1990, pp. 845–848.

C List of Figures

1 Flow diagram of HMM-based prediction of negation scopes. 4
2 Illustration of transition proceedings for an HMM with two states. 5
3 Illustration of emission proceedings for an HMM with two states. 5
4 Illustration of a fitted Hidden Markov model for our experiment. 9

D List of Tables

1 Related literature incorporating Hidden Markov models to predict negation scopes. 3
2 Used word classes and negation phrases in our dataset. 8
3 Results for different implementations of the tasks from section 4.2. 10