ICEB 2009 Proceedings

International Conference on Electronic Business (ICEB)

Winter 12-4-2009

Knowledge Discovery from Financial Text

Samuel W.K. Chan

KNOWLEDGE DISCOVERY FROM FINANCIAL TEXT

Samuel W.K. Chan
Department of Decision Sciences & Managerial Economics
The Chinese University of Hong Kong
swkchan@cuhk.edu.hk

Abstract

The abundance of on-line electronic financial news articles has opened up new possibilities for intelligent systems that could extract and organize relevant knowledge automatically in a usable format. While most typical information extraction systems require a hand-built dictionary of templates and, subsequently, are subject to ceaseless modification to accommodate new patterns that are observed in the text, in this research, we propose a novel text-based decision support system (DSS) that will (i) extract event sequences from shallow text patterns and (ii) predict the likelihood of the occurrence of events using a classifier-based inference engine. We investigated more than 2,000 financial reports with 28,000 sentences. Experiments show the DSS outperforms other similar statistical models.

Introduction

There has long been a strong interest in applying computational intelligence to analysis financial data. Traditionally, attempts have been concerned with forecasting the future based on past price data. One area of limited success in financial prediction comes from textual data. Textual data contain more information than numeric data because the former not only allows us to predict the financial trends, but at the same time, provides us with justification why it is to be done. Meanwhile, the current availability of huge volumes of financial electronic text has created a pressing need for better knowledge discovery and the construction of applications for managing the knowledge that is extracted, most existing literature on financial text mining or knowledge discovery from texts (KDT) relies on identifying a predefined set of keywords. In this approach, a text is usually scanned for a specific type of event template, such as corporate acquisitions. The main goal of the approach is to fill up the values for sets of handcrafted and predefined template slots. As a consequence, the construction of event extraction templates is a fairly laborious activity. It is hard to design templates that anticipate all the possible combinations of events or objects of interest that can be described. Given the brittleness of the approach and the demand for high-level representations, i.e., not just keywords, to take advantage of linguistic knowledge, there has been considerable interest in the development of an automatic means of learning shallow event patterns from text. In this article, we propose a novel inference engine for financial text sequences prediction which brings together the benefits of shallow text processing and classifier-based inferences to produce effective knowledge discovery. In particular, our approach aims at extracting key underlying event sequences from financial texts and then hypothesizing and assessing the incoming, even new and unseen, event sequences in her prediction. Unlike other similar approaches, an inferential mechanism is developed in the engine that can extract event sequences from a collection of relevant texts, and then collate the sequences in such a manner that both explicit and implicit information can be tailored to the needs of users. The task we are addressing and the problem of predicting financial event sequence can be stated as follows: Assume we have been given a corpus of financial documents which demonstrate chains of event sequences, we will explain how to extract all the event sequences from the texts, to discover the cause-consequence pairs underlying the events, and to predict the interesting and unseen relationships between them. This article is organized as follows. Section 2 outlines our system overview, along with issues regarding text preprocessing, shallow parsing, textual information generalization and event sequences extraction. Section 3 describes a classifier-based inference engine for the event sequence prediction. An inferential mechanism is introduced into the engine to produce a set of prediction rules. Numerous features that characterize the sequences and their latent inter-event relations are captured in the engine. The system prototype has been implemented and we have conducted a series of experiments to evaluate and compare the engine with the hidden Markov model. Section 4, followed by a conclusion, provides an overview of our experimental design. We also quantify the outcome along with a detailed analysis of the results in our evaluation.

System Overview

In this section, we will first present an overview of our system and the relevant methodology used throughout this study. Our text-based DSS includes four major components that involve text preprocessing, textual information generalization, event sequence extraction and a classifier-based inference engine. The system architecture of the DSS is shown in Figure 1. For an incoming text, after the text preprocessing, every sentences in the text are analyzed. Textual information is then extracted via a shallow parser and semantic roles assignment in the textual information generalization module. Every piece of textual information may indicate a possible event and a series of consecutive events signal an event sequence which is decisive for the sequence prediction.

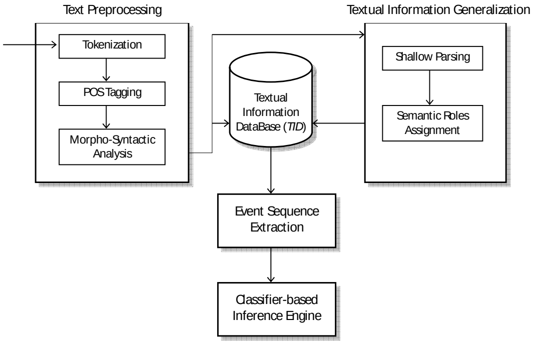

Text Preprocessing
Textual Information Generalization
Tokenization
POS Tagging
Morpho-Syntactic Analysis
Textual Information DataBase (*TID*)
Shallow Parsing
Semantic Roles Assignment
Event Sequence Extraction
Classifier-based Inference Engine

Fig. 1: System architecture of the text-based DSS

Text Preprocessing

The English language usually contains some redundancy and not every piece of English text delivers useful meaning. Sentences tend to include information which is not equally important. One possible way to eliminate the redundancy is to use a preprocessing technique to transform the input text into a pre-defined format. Our preprocessing consists of the sequential application of three major components in a pipeline as depicted in Figure 1. The components include tokenization, part-of-speech tagging and morpho-syntactical analysis. The output of the preprocessing is textual information which contains the most important information in the sentence and ignores the irrelevant. As shown in Figure 1, the first stage of the preprocessing is carried out by a tokenizer, which segments the input text into sentences and phrases. While punctuation signs, such as periods, may hint on the end of sentences as shown in the sentence (P1) below, they can be the end of an abbreviation, such as the word *U.K.*, or can be used in the specification of dates, times, initials of names, or email addresses.

"The pound fell against the dollar and the euro. Reports showed the U.K. economy slumped the most in two decades and home sales dropped to the lowest level since at least 1978." -- The New York Times. (P1)

Similarly, spaces are the delimiter of English words and they are not necessarily true to identify the word boundaries as in the case for many named entities such as *The New York Times*. After the tokenization, all the sentences of the text documents are then tagged with a part-of-speech tagger. Part-of-speech tagging is the task of assigning to each token its corresponding part-of-speech, i.e., its syntactic word category such as noun, adjective, verb or adverb. Different tagsets as well as different paradigms have been applied to the task. In this research, we adopt the Brill's tagger which is a transformation-based approach in the sense that a tag is assigned to each word and changed using a set of predefined rules (Brill, 1994). The basic rationale of the tagger is that it first computes the error score of each candidate rule and then best rules with higher score are selected. The winning candidates will be added into the ruleset until no further rule has a score above a given threshold.

The final step in the text preprocessing is the morpho-syntactical analysis which is used to reduce the number of irrelevant and rare terms in the sentences. The analysis is a language-dependent unit which performs stemming and lemmatization. Stemming is the process for reducing derived words to their base form and removes all prefixes and suffixes for each token. Our dictionary-based stemmer reduces the words *predicts*, *predicted*, *predicting*, *prediction* to the root word *predict*. Similarly, lemmatization is the process of reducing the morphological variants of the words to their corresponding base form. All these two tasks provide the important morphological features of each word. In addition, only words with informative POS tags are included, simply because these words contain relevant and important information to support the future prediction.

Textual Information Generalization

After the lexical processing in the preprocessing module, the textual information is further extracted in the textual information generalization module. In the module, a set of finite-state rules is applied to the sentences and produces a sequence of non-overlapping phrases as output. During the generalization, all the words in the sentences are being categorized into function or content words. While function words are words that have little lexical meaning and they usually serve to express grammatical relationships with other words within a sentence, content words which mainly include nouns, verbs and adjectives will deliver the full meaning of sentences. Technically speaking, the generalization is achieved by a shallow head-driven parser. The shallow parsing is a heuristic approach which consists of matching certain regular expressions and rules over part-of-speech tags of the input sentences. The parser begins with the identification of every potential phrasal heads of the input sentences, which is guided by phrasal patterns registered in the regular expressions. It mainly distinguishes the possible chunks of the sentences. Chunks are non-overlapping and non-recursive. Non-recursive means that chunks are not embedded within other chunks. The chunks extracted exhibit islands of words which build up the syntactic units that include noun phrases, verb phrases, prepositional phrases, location/position phrases, date/time phrases, quantifiers and the others. In particular, the noun phrases are further categorized into proper nouns, such as the names of country, stock, currency, organization, individual, and place, are further tagged by a name entity lexicon. The parser is found to have 92% and 85% *F*-measure for training and testing data respectively (Chan, 2009). In addition, the generalization will also assign the extracted chunks with the type of semantic restrictions they typically refer to.

The extracted chunks, together with their headwords, syntactic and semantic information, are stored in a textual information database (TID) as shown in Figure 1. The information extracted from the two modules produces a tabulated representation for every sentence in the financial text. To clarify the above discussion, as an example, suppose for the above input sentence (P1), the system will generate a tabulated representation of the sentence as shown in Table 1. In the TID, every sentence is broken down into one or more instance-attribute pairs as shown in the rightmost column of the table. The TID, without any language redundancy, provides a quick reference of the sentence information in form of instance-attribute pairs, without referring back to the piece of text.

Phrase	Headword(s)	Syntactic category	Semantic class	Instance-Attribute-Pair
the pound	Pound	PNP	currency	-
fell against	fell	VP	down	(UK-pound, fell)
the dollar and the euro	dollar, euro	PNP	currency	(modifier, dollar)
U.K. economy	U.K. economy	PNP	economy	-
slumped the most	slump	VP	down	(UK-economy, slump)
In two decades	Two decades	TP	time stamp	(modifier, LPT)
home sales	home sales	NP	property-market	-
dropped to the lowest level	drop	VP	down	(property, down)
since at least 1978	since 1978	TP	time stamp	(modifier, LPT)

Table 1: Structure of the textual information database (TID)

Event Sequence Extraction

Certainly, causality, temporality and spatiality are the intertwining links among all the possible events in a text when behavioral episodes are unfolded in the sentences. It is ascertained that readers attempt to tie each event or fact encountered to the prior text. To model the event sequence underlying in a text, we first assume $U = \{u_1, u_2, \ldots u_m\}$ be a set of m distinct instances comprising the alphabet and $A = \{a_1, a_2, \ldots a_n\}$ be a set of n distinct attributes which describe the possible states of the instances. An event is a non-empty instance-attribute pair $S_i = (u_i , a_i)$ and an event sequence S is an ordered list of events and denoted as $S = \langle S_1 \rightarrow S_2 \ldots \rightarrow S_r \rangle$. In addition, a sequence S is a subsequence of T, if all elements of S can be found in T in the same order (Zaki, 2001; Dong & Pei, 2007), no matter whether S and T are equal in length or whether one or more elements are being inserted into T. For example, $S = \langle A \rightarrow B \rightarrow C \rangle$ is a subsequence of $T = \langle Z \rightarrow A \rightarrow Y \rightarrow B \rightarrow X \rightarrow X \rightarrow C \rightarrow D \rangle$. An event subsequence differs from an event substring in that the events in an event substring must be contiguous, whereas the events in an event subsequence need not be. This explains S is an event subsequence, but not an event substring, of T. Frequent event subsequences are particularly useful in two basic aspects in analyzing the financial news. First, event subsequences capture the common intertwining links among the events delineated in the financial texts. For example, the frequent subsequence S tells us that at least two events A, B will usually occur prior to the event C. Second, the subsequence can also be used for predicting the financial trends or behaviors in the market (Eichinger *et al.*, 2006).

As shown in Figure 1, our event sequence extraction is a process of identifying the relevant event sequence in the dataset and discarding the irrelevant ones. In this research, we shift our event sequence extraction to the problem of approximate matching of event subsequences which specify in the user enquiry S with all the event sequences T stored in the textual information database. Given two event sequences S and T, a longest common subsequence $Q = LCS(S, T)$ is a longest possible subsequence with events in Q also being the events in S and T and they are in the same order (Crochemore *et al.*, 2007; Gusfield, 1997). We compute the *LCS* at the basis of the computation of an optimal alignment between the two sequences. The process utilizes a technique called dynamic programming in which the production of an alignment between two sequences is based on the computation of the edit distance between them. The edit distance, also called the Levenshtein distance, between two sequences is defined to be the minimum number of edit operations to transform the sequence from S to T. The edit operations include change, insertion and deletion. It is an efficient method of identifying the subsequences with closest match, even with some unknown gaps. It is out of the scope of this paper to further explain the algorithm in details, however, interested readers could refer to the above literatures for references. To put it in a nutshell, for a given user input event sequence S, we attempt to extract all the possible relevant sequences in the database. Each extracted event sequence is in form of a series of instance-attribute pairs. Certainly, the attribute of each event in the sequences aren't independent, but rather the attribute of each event depends on the preceding, or subsequent, event in the sequence. For example, the attribute *down* in the instance *property* relies on the attribute of its preceding instance *UK-economy* in that event sequence shown in the sentence (P1). It is interesting to know: Do all extracted event sequences originated from the input *S* follow some particular patterns? Are they on the rolls of a particular family? If confirmative, the frequent event sequences of *S* reveal the strong association among the events and suggest a clue in the prediction. Next, we will clarify how the event sequence prediction is being modeled as an extrapolation of underlying inter-related chain of attributes from which the actual events shown in the sequence are generated.

Machine learning methods often play a key role in sequence prediction because it is extremely difficult and costly to encode the prescribed prediction models manually. The general form of probabilistic model, called a hidden Markov model (hereafter, abbreviated HMM), has been among the most conventional and accurate methods for situations where certain sequences can be generated from different major states. While HMM is useful when one can think of the underlying attribute chain generates the surface events probabilistically, we explore a decision tree classifier in identifying the most likely sequence of attribute chain that could have emitted the given event sequence (Quinlan, 1993; Li, 2005). One of the advantages of using the decision tree classifier is that sequence prediction generally requires an interpretable model. While some other classifiers are far more enough to produce an accurate prediction, it is absolutely desirable for the classifiers to be explanatory. That is, any prediction model should and could provide a qualitative understanding of the relationship between the joint values of the input variables and the resulting predicted values. As opposed to other classifiers, such as probabilistic or neural network approaches, the decision tree classifiers provide meaning and justification to every verdict they returned in their prediction. The hierarchical structure of a decision tree also renders a relatively fast construction and produce interpretable model.

Classifier-based Inference Engine

Our inference engine is designed to learn a set of prediction rules from the event sequences, so that they can be used to classify or predict the likelihood of the occurrence new observations. A set of classifiers and rules are learnt from the training data in form of *n*-tuple. For each tuple, we include properties of the current event as well as information about its preceding and subsequent events in the training data. The assumption here is that the events in the sequence are not independent, but they are mutually related. One can model this dependence implicitly by including information about the preceding and subsequent events. This tactic is inspired by the methods in natural language processing for tasks such as part-of-speech (POS) tagging, where tags of both preceding and subsequent words are used as features to predict the current POS tag. In this inference engine, for any attribute a_i of the event S_i in the event sequence, we consider the following feature set.

- The attribute a_i of the current target event S_i.
- The attributes of two preceding events, S_{i-1} and S_{i-2}
- The attributes of two subsequent events, S_{i+1} and S_{i+2}
- The relative position of the event S_i found in the event sequence.
- Two major information-theoretic functions, mutual information and likelihood ratio, in quantifying the collocation of the attributes.

To train the classifier to learn the current target attribute, we provide the attributes of its neighbors in the event sequence *S* in the learning. The relative position of the event S_i in the sequence *S* is calculated by dividing the absolute position in the sequence by its length. Two kinds of information-theoretic functions, namely, mutual information (*MI*) and likelihood ratio (*LR*), are applied in quantifying the collocation of events. They quantify the collocation disparity between the events around the target event. *MI* of two random variables is defined as a quantity that measures the mutual dependence of the two variables. It is roughly a measure of how much one event tells us about the other. It compares the probability of observing events *X* and *Y* together to the probability of observing them by chance. The mutual information between particular events *X*, *Y* is defined in Eqn.(1).

$$MI(X,Y) = \log \frac{P(XY)}{P(X) \times P(Y)} \qquad \text{Eqn.(1)}$$

On the other hand, the log *LR* is a formalization of independence which provides another good measure for the collocation between the two events. While there is evidence that sparseness is a problem for *MI*, the likelihood function seems to be a good complement to it. In applying the *LR*, the two alternative hypotheses are examined.

$$H_0\colon P(Y|X) = p = P(\neg Y\,|X)$$
$$H_1\colon P(Y\,|X) = p_1 \neq p_2 = P(\neg Y\,|X) \qquad \text{Eqn.(2)}$$

If c_1, c_2, and c_{12} are the frequencies of events *X*, *Y* and the event *X* followed by event *Y* and *N* is the total number of events in our training, the log of the *LR* is calculated as follows:

$$\begin{aligned}\log LR = \;& \log L(c_{12}, c_1, p) \\ & + \log L(c_2 - c_{12}, N - c_1, p) \\ & - \log L(c_{12}, c_1, p_1) \\ & - \log L(c_2 - c_{12}, N - c_1, p_2)\end{aligned} \qquad \text{Eqn.(3)}$$

where $L(k, n, x) = x^k (1-x)^{n-k}$ and

$$p = \frac{c_2}{N} \quad p_1 = \frac{c_{12}}{c_1} \quad p_2 = \frac{c_2 - c_{12}}{N - c_1}$$

Both mutual information and likelihood ratio reflect the event co-occurrence information in a continuous function instead of discrete values shown in the feature set. Training of the inference engine proceeds as follows. Each attribute in the event sequences produces a training example for the decision tree construction. Given a sequence, the feature set is computed with respect to each target attribute, thereby producing a training *n*-tuple of the target attribute describing the event. All the tuples are presented to the decision tree learning. As the result, a decision tree classifier is learnt from the training examples. The classifier becomes the major component in our inference engine. In the prediction phase, this process is repeated as in the training, producing a prediction *n*-tuple for each unknown attribute in the incoming event sequence. The prediction *n*-tuple is then subject to the verdict of the classifier. At the same time, as in other decision tree classifier, each prediction is associated with a certainty factor, ranging from 0.5 to 1. The factor indicates the confidence on which the prediction is made. One obvious hurdle in applying the inference engine in the event prediction is the problem of attribute noise. Since the attributes of the adjacent events are unknown particularly for the subsequent events S_{i+1} and S_{i+2} as mentioned in the feature set, the majority of the features cannot be evaluated. In other words, the classifier receives a noisy set of features as input due to the attribute dependence on the unknown attributes of their adjacent events. This situation is remedied in two stages in our inference engine. In the first stage, the default one is to use as the baseline attributes, i.e., the most frequent attribute for the event found in the training examples. Although this baseline attribute is clearly far from perfect, it provides a rough estimation, associated with an uncertainty, to all unknown attributes. In our second stage, instead of predicting all event attributes from the beginning of the sequence till the end consecutively, our prediction starts from the attributes with highest uncertainty in the first stage, with the hope that its adjacent events which have higher confidence will shed some light on the current prediction. All the features with attribute dependence in the tree classifiers are all activated during the prediction. All the event attributes in the sequence will be evaluated from the highly unresolved to the most promising ones in one complete cycle. Under this second stage, because of the attribute dependence, the adjacent attributes will surely be altered if the current attribute are being modified. In the same reason, the current attribute will also be affected by the newly altered adjacent attributes. This intertwined cause-and-effect chains in the prediction are settled by allowing the whole prediction mechanism to repeat iteratively for several cycles until the event sequence becomes steady. An event sequence is defined as steady if there is no further modification of attributes during the prediction. Similarly, the certainty factor of attributes predicted by the classifier in the *i*-th cycle will define the precedence order of event attributes in the (*i*+1)-th prediction cycle. Table 2 below shows the pseudocode of the inferential mechanism which describes the main rationale behind the event prediction in more detail.

```
COMPUTE the certainty factor for all attributes $cf(a_k)$
        in the event sequence.
WHILE the prediction is not steady
  NORMALIZE all the certainty factor $cf(a_j)$ for $j = 1\ldots k$
  PUSH the sorted $a_j$ into a stack $s$ with the smallest $cf(a_j)$ at the top
  WHILE $s$ is not empty
    POP $a_j$ from $s$
    READ the attributes of the event $S_{j-2}$, $S_{j-1}$, $S_{j+1}$ and $S_{j+2}$
    GENERATE the prediction features for the attribute of event $S_j$
    ACTIVATE the decision tree classifier for the prediction
    FOR each fired rule in the classifier
      COMPUTE the Laplace estimate $L$ for each attribute class $n$
      $L_n \leftarrow L_n \times \max[cf(a_l)]$ for all adjacent events
      $v_n \leftarrow v_n + L_n$, where $v_n$ is the vote of the attribute class $n$
    ENDFOR
    RETRIEVE the attribute class with the largest average vote, i.e.,
        $cf(a_j) \leftarrow \max\left\{\frac{v_n}{\text{total number of rules fired}}\right\}$
    NORMALIZE all the certainty factor $cf(a_j)$ for $j = 1\ldots k$
  ENDWHILE
ENDWHILE
```

Table 2: Inferential mechanism for event prediction

As shown in Table 2, in order to take into the account of the effects from its adjacent events, the vote v_n of the attribute class n relies on the certainty of its neighbors. In other words, the vote will always be diminished by its neighbors, even though a large certainty factor may be deducted in the classifier. This is somewhat unreasonable and the situation will also be deteriorated as the certainty factors are propagated along the computational chain before the final verdict becomes steady. To maintain the impacts of its adjacent events without withering away the deduction from the classifier, a normalized certainty factor (NCF) is defined to alleviate the possible negative impacts during the normalization as shown in the mechanism. Without the normalization, the certainty factors will get smaller and smaller after each prediction cycle. The NCF at position *j* of an event sequence is defined as follows.

$$NCF_j = \left(\frac{CF_j - CF_{\min}}{CF_{\max} - CF_{\min}} \times \left(1 - LB\right) \right) + LB \qquad \text{Eqn.(4)}$$

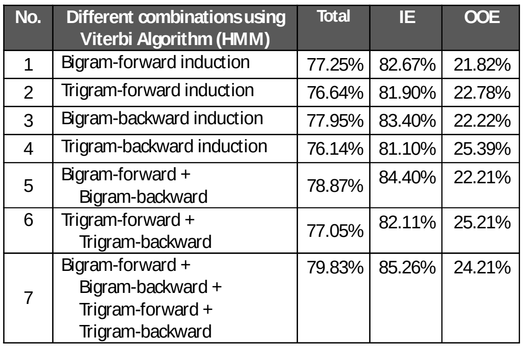
where

- CF_j be the certainty factor of the attribute at position *j*.
- $CF_{\min}$ be the minimum certainty factor found in the sequence.
- $CF_{\max}$ be the maximum certainty factor found in the sequence.
- *LB* be the lowest bound of the NCF.

Experiments and Evaluation

We have conducted an experiment to measure the performance of our approach in predicting the financial event sequences and make a head-to-head comparison with a statistical model. More than 28,000 financial sentences from 2,000 financial reports have been analyzed. They are all extracted from two financial websites, `www.bloomberg.com & www.quamnet.com` which is a popular financial portal. Quamnet is supported by a team of professional financial analysts. It is well-known for its independent financial research and analysis on Hong Kong's stock market. They are all used for gathering the statistics in analyzing sentence patterns in financial events. All the sentences are then subject to our shallow parsing strategy and over 70% of the sentences can be further analyzed and being parsed into the event sequences as described in Section II. Each sequence contains at least 4 events.

Statistical models of sequence prediction, particularly the Markov chain models, have been extensively studied due to their elegant theoretical framework. However, while each state of Markov chain models corresponds to an observable event which may be too restrictive to be applicable to many problems of interest, it is more general to extend the concept of Markov chain models to include the case where the observation is a probabilistic function of the states (Rabiner, 1989; Skounakis *et al.*, 2003). The resulting model is called a hidden Markov model (HMM). The word hidden in the name indicates that for a specific sequence of events, it is not clear what state the Markov model is in. HMMs are useful for situations where certain sequences can be generated from different important states. In our evaluation, in order to identify the attributes that best explain the event sequence, we apply the Viterbi algorithm which efficiently computes the most likely event sequence. The algorithm is a dynamic programming algorithm for choosing the best state sequence that maximizes the likelihood of the state sequence for the given observation sequence. It is to find the most likely sequence of hidden states that result in a sequence of observed events, especially in the context of hidden Markov models (Forney, 1973). Table 3 below shows our evaluation of the Viterbi algorithm using different set of parameters. The bigram model looks at pairs of events and uses the conditional probability that an event S_i will follow another event S_{i-1}. The trigram model uses the conditional probability of one event given the two preceding events. Our forward induction is the prediction that starts from the first event of a sequence, and then infers consecutively till the end. The backward induction is similar to the forward counterpart but starts at the end of the sequence and sweeps backward through the events in the sequence. Additionally, in any real world predictions, we cannot guarantee that the complete information about the event sequences are all reflected in our training cases, we address the event prediction problems with the open world assumption. That is, it is allowed to have some unknown events found in the testing cases, but absent in the training cases. Hence, we define in-Event (IE) be the events that can be found in our training data while out-of-Event (OOE) is an event that are unknown to the training data.



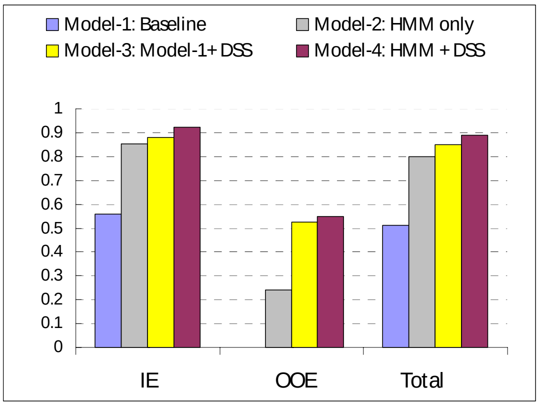
No.	Different combinations using Viterbi Algorithm (HMM)	Total	IE	OOE
1	Bigram-forward induction	77.25%	82.67%	21.82%
2	Trigram-forward induction	76.64%	81.90%	22.78%
3	Bigram-backward induction	77.95%	83.40%	22.22%
4	Trigram-backward induction	76.14%	81.10%	25.39%
5	Bigram-forward + Bigram-backward	78.87%	84.40%	22.21%
6	Trigram-forward + Trigram-backward	77.05%	82.11%	25.21%
7	Bigram-forward + Bigram-backward + Trigram-forward + Trigram-backward	79.83%	85.26%	24.21%

Table 3: Results of the HMM in event prediction, where Total, IE and OOE represent the overall accuracy in all events, in-events and out-of-events respectively.

While the increase of *n*-gram does not show any significant improvement over the accuracy as compared with the bigram counterparts, three different combinations in studying their synergy effects are further investigated as described in trials 5-7. For example, in trial 7, we incorporate the forward and backward probabilities for both bigrams and trigrams during the predictions. However, the results show that it is not necessary to produce any significant positive synergy effect, even though there involve heavy computational resources. To justify our approach and evaluate the performance of our text-based DSS, we compare our system using several prototypes which include,

- Model-1: a baseline model in which attributes are estimated using the most frequent attributes for the events found in the training cases;
- Model-2: the most outstanding combination of HMM, i.e., trial-7, as shown in Table 3;
- Model-3: first approximates by Model-1, and then refines using our text-based DSS;
- Model-4: first by Model-2 and then refined by our DSS.

Figure 2 summarizes the evaluation results of the above four models. As shown in the figure, the overall accuracy of Model-3 and Model-4 are higher than that of the one-component models. While all the models, except Model-1, demonstrate a strong predictive power for the in-events (IE), our DSS model exhibits a total accuracy (Total) higher than the HMM. Specifically, the DSS model appears to have a significant improvement on the predictive power for the OOE with the average over 50% in Model-3 and Model-4. The differences between the Model-2 and our DSS are statistically significant. Another important point of comparison is that the DSS is trained with the features described in Section III. HMM, on the other hand, computes the probability of any sequence of events simply by finding the likelihood of events and multiplying the transition probabilities together. On the other hand, the learning and predicting mechanism in our DSS relies on the interdependences between the target attribute and the features. These features reflect the properties of the preceding and subsequent events as well as their collocation information. All these features are all considered as a whole in every single instant during the training of the inference engine. As a result, it is not surprising that our DSS outperforms the HMM.



F ig. 2: Comparison of the performances among four different models

Next, we experiment with the sensitivity of the inference engine to several options of labeling the training data. In Model-4, all features, including the target feature, of the training instances are computed in terms of actual attributes of the events in the training event sequences. In our sensitivity analysis, we gradually replace the actual attributes in the training cases by the most probable attributes for the events found in all training examples, whereas the targets still correspond to the correct target attribute. In other words, the quality of the training data is deliberately deteriorated to an extent that they are only approximated by a maximum likelihood estimation based on the probability density function. It is not surprising the performance of the engine begins to get worse as compared to the one trained from the actual datasets.

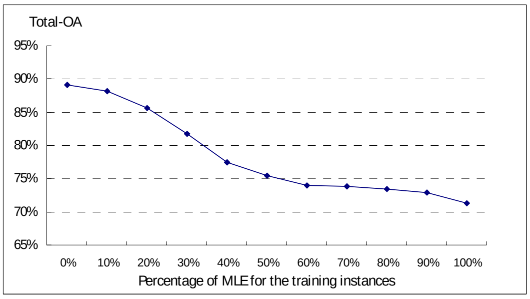


Fig. 3: Change of Total accuracy with the increase of percentage of MLE for training instances

However, as shown in Figure 3, the degeneration is slow-crawling with the total overall accuracy drops gradually from 89.09 to 71.34%. It is interesting to note that the performance of the engine using 100% of maximum likelihood estimation (MLE) for the actual dataset is still acceptable, at least it is much better than the performance of our baseline model in Model-1. In other words, a "simple-minded" inference engine using 100% MLE performs reasonably well in predicting the event sequences. This tolerance characteristic explains why we can alleviate any possible lack of inter-dependence features by adopting the maximum likelihood estimation in our OOE prediction.

Conclusions

The current one-size-fits-all design in knowledge discovery from text (KDT) ignores the fundamental variety of domains which may involve a fairly laborious annotation. Thus, most approaches are forced to depend on impoverished text models like bags of words or *n*-grams. As a result, the predictions that they are making ought to be based on the meanings of those words in context. That lack of semantics causes misinterpretation of meaning that results in failing to extract the relevant knowledge from text. On the other hand, in most of the current KDT systems, the information that is sought must be explicitly stated, but not implied, in the text. This lack of inferential capability can pose significant problems when knowledge is extracted from documents that expect the reader to draw simple deductions. In this research, we have developed an inferential mechanism in a text-based DSS that can extract knowledge from a collection of relevant texts. The DSS does not rely on any bag of words, but on a shallow language model. As a consequence, the incorrectly conflated information is excised and the proper coreference of event sequence can be accomplished. In addition, instead of learning from a limited set of hand-crafted rules or templates, our DSS collates the prior event sequences in a manner that both explicit and implicit knowledge could participate into the inferences. This mechanism provides a robust and efficient means to handle the predictions in financial text which demonstrates a high propensity that the extracted event sequences are being finite at various level of analysis. Certainly, further investigation has to be conducted to extend the work to a fully unrestricted functionality text domain. However, we have demonstrated one of the alternatives in the knowledge discovery from real text.

Acknowledgements

The work described in this paper was partially supported by the grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project Nos. CUHK470605 and CUHK440607)

References

[1] Brill, E. (1994). Some advances in rule based part of speech tagging. *Proceedings of The Twelfth National Conference on Artificial Intelligence* (AAAI-94), Seattle, Washington.

[2] Chan, S.W.K. (2009). Shallow semantic labeling using two-phase feature-enhanced string matching. *Expert Systems with Applications*, 36, 9729-9736.

[3] Crochemore, M., Hancart, C., & Lecroq, T. (2007). *Algorithms on Strings*. Cambridge University Press.

[4] Dong, G., & Pei, J. (2007). *Sequence Data Mining*. Springer.

[5] Eichinger, F., Nauck, D. D., & Klawonn, F. (2006). Sequence Mining for Customer Behaviour Predictions in Telecommunications. *Proceedings of ECML/PKDD 2006 Workshop on Practical Data Mining: Applications, Experiences and Challenges*, Berlin.

[6] Forney, G. D. (1973). The Viterbi algorithm. *Proceedings of the IEEE*, 61, 3, 268–278.

[7] Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press.

[8] Li, X.-B. (2005). A scalable decision tree system and its application in pattern recognition and intrusion detection. *Decision Support Systems*, 41, 112-130.

[9] Quinlan, J.R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

[10] Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77, 257-286.

[11] Skounakis, M., Craven, M., & Ray, S. (2003). Hierarchical Hidden Markov Models for Information Extraction. *Proceedings of the 18th International Joint Conference on Artificial Intelligence.*

[12] Zaki, M.J. (2001). SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42, 31-60.