Speech Intent Recognition for Robots

Borui Shen
University of Ottawa
School of Electrical Engineering and Computer Science
Ottawa, ON, K1N6N5, Canada
bshen089@uottawa.ca

Diana Inkpen
University of Ottawa
School of Electrical Engineering and Computer Science
Ottawa, ON, K1N6N5, Canada
Diana.Inkpen@uottawa.ca

***Abstract*— We present the design of a dialog manager that performs speech intent recognition, based on a finite state machine which enables simultaneous processing of multiple sub-modules and maintains ordered transitions of system states. The dialog manager is integrated into a spoken dialog system. The application area of this system is targeted on robots, and the core problem that we address is to recognize user's speech intents, which could be either asking questions or giving commands to a robot. Our dialog manager is a sequence classifier based on hidden Markov models, and it uses part-of-speech tags as output symbols. The classifier take sentences of variable lengths as input. It is trained on a small data set and achieves and accuracy of 83%.**

Keywords—Computational Intelligence, Robots, Spoken Dialog System, Sequence Classification, Hidden Markov Models

I. Introduction

Spoken dialog systems are software systems that allow human-machine interaction through speech. They rely on the automatic speech recognition technology, which has been developing rapidly in recent decades. Spoken dialog systems have been applied in multiple areas, including self-serving call centers and mobile personal assistant agents. Spoken dialog systems are also considered as an important component for robots, to allow humans to communicate with robots through speech. One characteristic of a robot dialog system compared to other dialog systems is that robots have the ability to perform certain actions; therefore, a robot dialog system should be able to recognize user's intents when commands are given to the robot.

Building spoken dialog systems usually requires great effort, because the training of an automatic speech recognition engine, natural language understanding and speech generation modules are all essential to conduct conversations [1]. Furthermore, due to the nature of spoken languages, many challenges need to be addressed: there are no clear boundaries between words; there is variability in speech between different speakers; there is noise in the environment, etc. This is why speech recognizers are usually limited to certain domains, in order to improve the recognition accuracy [2]. To avoid these difficulties, our system utilize available cutting-edge tools for some of the modules, such as Google ASR for speech recognition, and focuses the research goal on speech intent recognition.

We consider speech intent recognition as a sequence classification problem. After the comparison of different classification methods, which will be discussed in following sections, a classifier based on hidden Markov models is chosen and implemented; we also evaluated its classification accuracy.

The rest of this article is organized as follows: Section 2 discusses research related to dialog systems and sequence classification methods. Section 3 describes our design and implementation of the robot dialog system. Section 4 presents the sequence classification method and its related algorithms. The performance is evaluated in section 5. The last section, presents our conclusions and future work.

II. Related Work

Researchers in the Sungkyunkwan University had developed spoken dialog system for intelligence service robots [3]. The core components of their system is a dialog manager which makes use of finite state machine to enhance the capability of handling inputs in multiple domains. However, the system in their work is limited to provide information delivery services, similar to a question answering system, but deployed on a robot. The ability of controlling robots through speech commands is not explored.

Another implementation of a robot dialog system [4] is template-based and event-driven. The system is able to keep track of dialog domains and evolve conversations based on context information. By integrating with agents for vision, navigation and radio-frequency identification, the robot perceives events and receives messages from the environment to achieve multimodal interaction. However, the implementation of template-based spoken language understanding limits the flexibility of the input grammar, and the performance is highly dependent on the accuracy of speech recognition.

In terms of sequence classification, which is the core problem of our work, many traditional approaches [5] [6] attempt to define string kernel functions and extract a fixed-length feature vector. Then the sequence can be classified based on a discriminant function or a nearest neighbor method with a distance function [7]. Recent research also proposed the idea of using recurrent neural networks for sequence classification [8], which are able to learn the semantic

978-1-5090-0973-2/16 $31.00 © 2016 IEEE
DOI 10.1109/MCSI.2016.38

association and the relation between sequence labels and classes. However, the training of neural network models requires a large set of training data.

III. System architecture

Our system contains a main module called the dialog manager (DM) and four sub-modules: automatic speech recognition (ASR), spoken language understanding (SLU), action generator (ACT) and text-to-speech (TTS). In order to control the system from an external process, we also developed a system controller program to manipulate the system's runtime behavior. The system is designed to be cross-platform, this is achieved by developing most of the modules in Python programming language, with only the ACT module being written in C and running on an Arduino microcontroller.

From a high-level perspective, the system can be viewed as a dialog manager maintaining a process pool of four sub-modules. The ASR module can be considered as an input module: it receives audio signals constantly from a microphone and transcribes them into texts. The SLU module is the only module that does not communicate with hardware and it can be viewed as a processing module for speech intent recognition and action detection. ACT and TTS are the two output modules: the ACT module controls robot parts by sending commands to a microcontroller, while the TTS module speaks to user by playing audio files with a speaker. In the following parts of this section, the detailed implementation of each module will be discussed.

A. Dialog Manager

As the main process of the spoken dialog system, the dialog manager serves as a mainframe for all submodules and maintains the states of the system in correct order. The runtime logic of the system can be described as a finite state machine, with four states and a set of state transition conditions. At any time, the system can be in one of four states: "CLOSE", "WAITING", "ACTIONING" and "ANSWERING". The transitions are depicted in Figure 1.

The initial state of the system is "CLOSE" where only the main process is running in this state. The system is waken up or suspended with a start or exit signal released by the system controller program. In the "WAITING" state, the system keeps receiving audio chunks from the microphone and the processing is done by both the ASR and the SLU module. Upon reception of a result (either question or command) from the SLU module, the system goes into "ANSWERING" or "ACTIONING" state to render the output.

The implementation of this finite state machine is done with a process pool and each sub-module communicate with the dialog manager using message queues. Owing to the asynchronous feature of multi-processing, the dialog manager is able to receive messages from sub-modules simultaneously. The finite state machine ensures that inputs are processed in correct order and stable transition of system states.

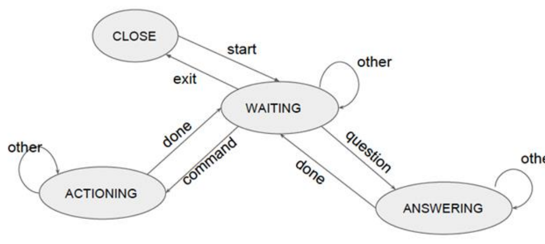

Fig. 1. Finite State Machine for the Dialog Manager

B. Maintaining the Integrity of the Specifications

The system controller runs as an independent process from the dialog system; it communicates with the dialog manager using named pipe. The commands being sent to dialog manager is entered by user in runtime and is considered to have top priority. Acceptable commands including start/exit the dialog system and start/exit a specific sub-module. The purpose of implementing a system controller is to provide flexibility for suspending and resuming the dialog system dynamically, rather than imposing a one-time start to exit logic. Furthermore, it also provides convenience for debugging during runtime.

C. Automatic Speech Recognition

The Automatic Speech Recognition (ASR) module is the input module of the dialog system. When the module is operating, it receives audio signals from a microphone continuously; upon the detection of a user utterance, the recorded audio is saved as a wave file and the actual speech-to-text is done by submitting it to the Google ASR service. Other speech recognizers could have been used or a custom one, but we chose Google ASR because it is a general domain speech recognizer and seems to work well for our system.

Utterance detection is the core task of the ASR module. Audio signals are collected continuously from the microphone, chunk by chunk. By calculating the root mean square value of each audio chunk and considering its amplitude, the start and end of a user utterance can be determined by applying an amplitude threshold. The module starts to record audio signals when the amplitude is higher than the threshold and stops when the amplitude is lower than the threshold for a certain elapsed time.

During the testing phase of the ASR module, we found that when the first word of a user utterance starts with a consonant, this consonant usually has low amplitude and thus is not being recorded, resulting in incorrect transcriptions from the speech recognizer, such as "hey" being transcribed as "a", or "please" being transcribed as "ease". The solution to this problem is to prepend several audio chunks prior to the beginning of an utterance. This is done by implementing a ring buffer data structure with a pointer to its head for holding temporary silent audio chunks, and prepending the data in this ring buffer to the recorded utterance.

The result returned from the Google ASR service is the transcribed text without any punctuation; it can also be empty if the recognizer fails to detect any words. At this point, the ASR module has finished one cycle of routine and the transcribed text is sent back to the dialog manager. Depending on the current state of the system, the text is then relayed to the SLU module for processing.

D. Spoken Language Understanding

The Spoken Language Understanding (SLU) module is the processing module of the dialog system. It takes the transcribed text from the ASR module as input, and interprets user's intent with the sequence classifier that will be discussed in the next section. If the input sequence of words is classified as an interrogative sentence or a declarative sentence, the system generates output as speech. If the classification result indicates that the sequence is an imperative sentence, the output will be a command to the robot.

Speech intent recognition is the processing step that SLU performs for every input. It is basically a classifier with three trained hidden Markov models (HMM), one for each of the three sentence types (declarative, interrogative, and imperative) and a softmax function to normalize the probability. The recognition result is the sentence type with the highest probability.

Action detection is performed only if the sequence is classified as an imperative sentence; this is done by using bag of words (BOW) and verb relevance as features, and the classification is done by a Support Vector Machine (SVM) classifier. For each action that the robot can perform, a set of sample commands is listed for constructing the BOW vectors for the training data. However, if a word is missing from the dictionary, the BOW vector cannot capture any information about it. Thus, we also extract verbs from the samples and construct a verb relevance vector as an extra feature. For each verb in a word sequence, its similarities to all the verbs in the verb vectors are computed based on WordNet [9] definitions. The final feature vector for a sequence is the concatenation of the BOW vector and the verb relevance vector.

For sequences that are classified as declarative or interrogative sentences, they are submitted to either a dialog service or a question answering (QA) service, to get the response text. We implemented interfaces to both the IBM Watson dialog service and the api.ai QA service. The IBM Watson dialog service requires some predefined configuration beforehand, in the format of a rule-based xml file, while the api.ai service simply returns an answer (may be empty) corresponding to the submitted question.

E. Output Generator

The output generator is composed of the Text-to-Speech (TTS) module and the Action (ACT) module, rendering a speech output and an action output, respectively. The TTS module submits text strings received from the dialog manager and downloads the synthesized speech audio, then plays back the audio with a speaker which it connects to.

The ACT module receives messages from the dialog manager and composes shell commands based on the message content. It requires the connection to a microcontroller which runs corresponding programs on the Robot Operating System (ROS) in order to move robot's parts.

IV. Sentence Classification

As mentioned in the introduction section, a robot dialog system accepts not only questions from the user, but also commands to a robot, and expects the robot to perform specific actions. Thus, a sentence classifier is required to recognize if a user utterance is a question or a command.

The length of sentences (number of words) is variable; however, for most of classifiers, such as logistic regression or SVM, a feature vector of fixed size is required. Though this problem can be solved by using BOW as features, the size of the BOW vector will be very large in the setting of a general domain dialog system, which implies high computational cost. Another problem of using BOW is that it does not captures the sequential information of a sentence; even with n-grams, the sequential information is limited by the parameter n.

Based on these consideration, we implement the sequence classification using HMM, which allows variable-length sequences as input and captures sequential dependency information. Though HMM is more generally used for sequence modeling, its capability of predicting the probability of a sequence for a class has inspired some researchers [10][11][12] to use it for classification tasks.

A. Hidden Markov Model

HMM is a statistical model with an underlying stochastic process that is not observable, but can only be observed from another stochastic process that produce observable symbols [13]. The key idea of HMM is a finite model that describes a probability distribution over an infinite number of possible sequences [14]. It has been applied in multiple areas, such as protein structure prediction [15] and speech recognition [16].

A HMM can be defined by a set of hidden states (S) and a set of output symbols (K). The state initial probability (∏) describes how probable a specific state may present at the beginning of a sequence. Any two states in a HMM model are connected with a state transition probability (A), describing how possibly one state can appear right after the other in a sequence. Each state can emit output symbols based on their emission probabilities (B). Given a hidden Markov model with predefined S and K, and a set of sample observation sequences (O), the parameters of this HMM (∏,A,B) can be learned with the Baum-Welch algorithm, which is a forward-backward algorithm based on the expectation maximization method. With these trained parameters, a Viterbi algorithm can be used to predict the hidden states of new observation sequences; this process is much faster than the training phase because a dynamic programming method can be applied.

To use HMM for our sequence classification problem, we define the output symbols as all the possible part-of-speech (POS) tags as defined by the Penn Treebank Project [17], containing 36 symbols. We define the number of hidden states as a hyperparameter to be fit. The objective is to learn the HMM parameters for each class (imperative, declarative and interrogative sentence) and use it for later predictions. The preprocessing, training and testing steps are discussed in the following parts of this section in detail.

B. Preprocessing

To get data for training the sequence classifier, 450 sentences were manually collected: 150 samples for each sentence type. The preprocessing step first removed all the punctuation marks in the data set, because the output text strings from the speech recognizer do not include punctuations, and then extracted POS tags for each word. The testing of a new sentence is done by first converting the sentence into a POS tag sequence, and then feeding the tag sequence to the three trained HMMs. The Viterbi algorithm is used to predict the hidden states sequence of each class and the associated probability value. Lastly, a softmax function is applied to normalize these three values and get the probability for each class.

The testing of an unknown sample is very efficient because the Viterbi algorithm is implemented with a dynamic programming method, as demonstrated in Figure 2 for one of our test sentences. For each HMM, the probability parameters ∏, A and B are obtained from the training step. Given an unknown sequence of POS tags, we consider these tags as HMM output symbols in a time sequence; for each hidden state i and each stage t in the time sequence, the algorithm computes the highest probability of being in state i at time t, based on the state probabilities in the previous time stage and records the predecessor hidden state that generated the current highest probability. The algorithm ends when it reach the last symbol and the hidden state with highest probability at this time is considered the end of hidden state sequence. The hidden state sequence can then be generated by backtracking the predecessor from the end state, and the probability of this hidden state sequence is the probability at the end state.class could be different and the three models were trained separately, each on the corresponding data set.

C. Training

The training step was done with the Baum-Welch algorithm. For each model, given a set of output symbols (the tag set) and its size n_{symbol} and the number of hidden states n_{state}, we initialized the emission probability of each state to $1/n_{symbol}$, the initial probabilities of the hidden states and the transition probabilities to be all $1/n_{state}$. Thus all the output symbols and hidden states are equally probable at the beginning.

We briefly summarize here the standard HMM training procedure [12]. In the forward procedure, we denote the o_t as the observation symbol at time t, a_{ij} as the transition probability from state i to state j, and $b_j(o_t)$ as the emission probability of state j emitting symbol o_t. With the initial parameters, the conditional probability $\alpha_i(t)$ of being in state i at time t, given the observation sequence $(o_1,o_2,...,o_t)$, can be updated recursively using the following formula:

$$\alpha_i(t) = \{\begin{matrix} \pi_i b_i(y_i) & t = 1 \\ b_j(o_t)\sum_{i=1}^{n_{state}} \alpha_i(t-1)a_{ij} & t > 1 \end{matrix} \quad (1)$$

In the backward procedure, we denote the length of the observation sequence as T, the conditional probability $\beta_i(t)$ of the ending partial sequence after time t $(o_{t+1},o_{t+2},...,o_T)$ given starting state i at time t can be updated with the following formula:

$$\beta_i(t) = \{\begin{matrix} 1 & t = T \\ \sum_{j=1}^{n_{state}} \beta_j(t+1)a_{ij}b_j(y_{t+1}) & t < T \end{matrix} \quad (2)$$

With the two conditional probabilities for all the states, we denote the hidden state sequence as $(s_1,s_2,...,s_T)$ and the observation sequence as $O=(o_1,o_2,...,o_T)$, the initial probability of state i can be represented as $P(s_1=i \mid O,\alpha,\beta)$ and updated with:

$$\pi_i = P(s_1 = i|O,\alpha,\beta) = \frac{\alpha_i(1)\beta_i(1)}{\sum_{j=1}^{n_{state}} \alpha_j(1)\beta_j(1)} \quad (3)$$

The transition probability parameters $A=\{a_{ij}\}$ can be updated with the expected number of transitions from state i to state j compared to the total number of transitions origin at state i. Similarly, the emission probability parameters $B=\{b_i(o)\}$ can be updated with the expected number of times observation o is emitted from state i compared to the total number of observations emitted from state i. The formulas for updating these two probability parameters are:

$$a_{ij} = \frac{\sum_{t=1}^{T-1} P(s_t = i, s_{t+1} = j|O,\alpha,\beta)}{\sum_{t=1}^{T-1} P(s_t = i|O,\alpha,\beta)}, b_i(o) = \frac{\sum_t P(s_t = i|O,\alpha,\beta)\ \ (if o_t = o)}{\sum_{t=1}^{T} P(s_t = i|O,\alpha,\beta)} \quad (4)$$

For the training process of our system, the three classifiers (one for each class) were trained independently. Given training data of one class, we first initialized the parameters with the number of hidden states. Then, we performed an iteration over all the sample sequences in the data set, and we applied the above formulas to update the parameters. At the end of the training step, three HMMs with different output symbol sets and different probability parameters were trained. These models were saved for later prediction on test samples.

D. Testing

The testing of a new sentence is done by first converting the sentence into a POS tag sequence, and then feeding the tag sequence to the three trained HMMs. The Viterbi algorithm is used to predict the hidden states sequence of each class and the associated probability value. Lastly, a softmax function is applied to normalize these three values and get the probability for each class.

The testing of an unknown sample is very efficient because the Viterbi algorithm is implemented with a dynamic programming method, as demonstrated in Figure 2 for one of our test sentences. For each HMM, the probability parameters

∏, A and B are obtained from the training step. Given an unknown sequence of POS tags, we consider these tags as HMM output symbols in a time sequence; for each hidden state i and each stage t in the time sequence, the algorithm computes the highest probability of being in state i at time t, based on the state probabilities in the previous time stage and records the predecessor hidden state that generated the current highest probability. The algorithm ends when it reach the last symbol and the hidden state with highest probability at this time is considered the end of hidden state sequence. The hidden state sequence can then be generated by backtracking the predecessor from the end state, and the probability of this hidden state sequence is the probability at the end state.

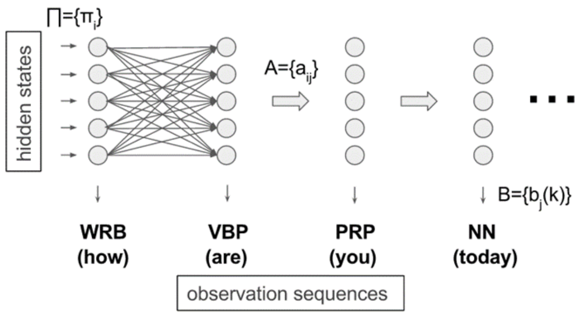

Fig. 2. Sentence Type Probability Calculation Using Viterbi Algorithm

V. Results and Evaluation

The evaluation for sentence type recognition is done with a ten-fold cross-validation method. The whole data set with 450 preprocessed sequences are divided into ten subsets, in each testing fold one of them is used as the test set while the other nine are combined to form a training set. Parameters of three HMMs are trained using the training set samples, then classes of test samples are predicted using the test method described in previous section.

The test results are compared to the ground truth labels of the test set, and the number of correct predictions for each class is counted. Since the data set is collected in a balanced way for each class, the precision rate can be used as the accuracy for each fold. The overall accuracy of the classifier is the average accuracy of the ten folds. The confusion matrix of ten-fold cross-validation with HMMs of five hidden states is as shown in Table I.

A demo video of our system is available at: https://drive.google.com/file/d/0B_zi9xXL_3Y5WThuWUpVQ1dMbnM/view?usp=sharing

We are also making the dataset and the code available on GitHub. https://github.com/raybrshen/robot_sds

TABLE I. Confusion Matrix for Cross-Validation

Class \ Prediction	declarative	imperative	interrogative
declarative	129	8	13
imperative	25	114	11
interrogative	30	22	98

In the dialog system, both declarative sentences and interrogative sentences are submitted to online services for speech response; only imperative sentences are considered as commands to a robot. Thus, in terms of speech intent recognition, this becomes a two class problem. The accuracy regarding speech intent recognition can be calculated by combining the data of declarative class and interrogative class. Using the data from the confusion matrix shown in Table 1, the accuracy of intent recognition is 83%, which is higher than 76%, the accuracy of the original three class sentence type classifier. The accuracy of the intent recognition for the three classes (76%) is quite high if we compared it with a baseline accuracy of a system that would randomly choose between the three classes (33% for our dataset).

The number of hidden states in this problem is a hyperparameter to be fit; numbers between two and ten are tested in this work and it is found that the accuracy does not tend to converge when the parameter goes larger. The accuracy of the classifiers (for 3 classes) with different number of hidden states (n_{hs}) is shown in Table 2 as well as in Figure 3.

TABLE II. Accuracy of Classifiers with Different Numbers of Hidden States

n_{hs}	2	3	4	5	6	7	8	9	10
Acc(%)	71.3	75.1	77.7	75.7	77.3	74.2	77.5	77.3	75.7

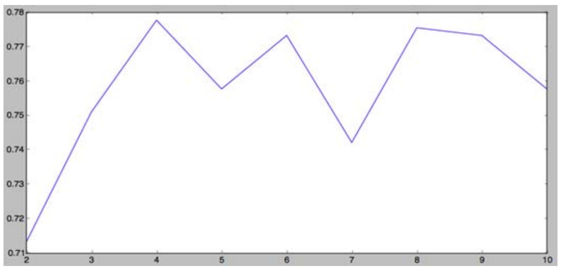

Fig. 3. Accuracy of Classifiers with Different Numbers of Hidden States

VI. Conclusions and Future Work

We presented a spoken dialog system for robots. The system has been deployed and tested on a robot arm which runs on a microcontroller and communicates with the dialog manager through ROS interface.

The dialog manager maintains a process pool of four sub-modules; they are independent from each other and this feature enhances the robustness of the system. All the sub-modules, as well as the dialog manager, can be dynamically suspended or resumed by a system controller program, which provides flexibility to the system.

In order to achieve speech intent (command/question) recognition, a sequence classifier based on hidden Markov models was implemented and evaluated. The classifier is composed of three HMMs corresponding to the three sentence types: declarative, imperative and interrogative. The number of hidden states for HMM was considered as a hyperparameter and was tested with different values. Based on the evaluation result, the classification accuracy is not significantly affected by the value of this hyperparameter.

In terms of classification performance, as shown in the evaluation section, declarative and imperative sentences are more accurately classified than interrogative sentences. Furthermore, the accuracy in terms of speech intent recognition is higher than that of sentence type classification.

The current implementation of spoken dialog system for robots is our first attempt; there is still room for improvements. Ongoing development will focus on the following aspects. We plan to develop a cache data structure based on least-frequently-used sorting algorithm to store user queries and outputs. By eliminating the time for processing repeated queries, the runtime efficiency of the SLU module can be improved. We plan to collect more training data for speech intent recognition by using unsupervised clustering algorithms rather than manual data collection. We hope to improve the classification accuracy for interrogative sentences by training classifiers based on audio features, such as the pitch (F0) at the end of utterances. We would also like to enhance the interactivity of the dialog system by utilizing the cameras on a robot to get image data, which may provide additional information for conversations.

Acknowledgment

We thank Professor Emil Petriu from the University of Ottawa, School of Electrical Engineering and Computer Science, for allowing us access to the robots in his lab. We also thank the Natural Science and Engineering Research Council (NSERC) of Canada for supporting our research.

References

[1] E. F. Lussier and H. K. J. Kuo, “Using semantic class information for rapid development of language models within ASR dialogue systems”, IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 1, pp. 553-556, May, 2001.

[2] M. Ana, B. Luka, P. Miran, M. I. Sanda and I. Ivo, “Overview of a Croatian weather domain spoken dialog system prototype”, 32nd International Conference on Information Technology Interfaces, pp. 103-108, June, 2010.

[3] C. Lee, Y. S. Cha, T. Y Kuc, “Implementation of dialogue system for intelligent service robots”, International Conference on Control, Automation and Systems, pp. 2038-2042, October, 2008.

[4] R. Jiang, T. Y. Kee, H. Li, C. Y. Wong, D. K. Limbu, “Development of Event-Driven Dialogue System for Social Mobile Robot”, 2009 WRI Global Congress on Intelligent Systems, vol. 2, pp. 117-121, May, 2009.

[5] P. P. Kuksa, “Biological Sequence Classification with Multivariate String Kernels”, IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 10, no. 5, pp. 1201-1210, March, 2013.

[6] P. P. Kuksa, V. Pavlovic, “Spatial Representation for Efficient Sequence Classification”, 20th International Conference on Pattern Recognition, pp. 3320-3323, August, 2010.

[7] A. Iosifidis, A. Tefas and I. Pita, “Multidimensional Sequence Classification Based on Fuzzy Distances and Discriminant Analysis”, IEEE Transactions on Knowledge and Data Engineering, vol. 25, no. 11, pp. 2564-2575, November, 2012.

[8] F. Raue, W. Byeon, T. M. Breuel, M. Liwicki, “Parallel sequence classification using recurrent neural networks and alignment”, 13th International Conference on Document Analysis and Recognition, pp. 581-585, August, 2015.

[9] G. A. Miller, “WordNet: A Lexical Database for English”, Communications of the ACM, vol. 38, no. 11, pp. 39-41, November, 1995.

[10] L. Rabiner and B. Juang, “An introduction to hidden Markov models”, IEEE ASSP Magazine, vol. 3, no. 1, pp. 4-16, January, 1986.

[11] M. Bicegoa, V. Murinoa, M. Figueiredob, “Similarity-based classification of sequences using hidden Markov models”, Pattern Recognition, vol. 37, no. 12, pp. 2281-2291, December, 2004.

[12] S. Blasiak and H. Rangwala, “A Hidden Markov Model Variant for Sequence Classification”, Proceedings of the Twenty-Second international joint conference on Artificial Intelligence, vol. 2, pp. 1192-1197, July, 2011.

[13] B. Esmael, A. Arnaout, R. K. Fruhwirth, G. Thonhause, “Improving Time Series Classification Using Hidden Markov Models”, 12th International Conference on Hybrid Intelligent Systems, pp. 502-507, December, 2012.

[14] S. R. Eddy, “Hidden Markov Models”, Current Opinion in Structural Biology, vol. 6, pp. 361-365, June, 1996.

[15] H. Pezeshk, S. Naghizadeh, S. A. Malekpour, C. Eslahchi, M. Sadeghi, “A modified bidirectional hidden Markov model and its application in protein secondary structure prediction”, 2nd International Conference on Advanced Computer Control, vol. 3, pp. 535-538, March, 2010.

[16] J. Dai, “Hybrid approach to speech recognition using hidden Markov models and Markov chains”, IEEE Proceedings of Vision, Image and Signal Processing, vol. 141, no. 5, pp. 273-279, October, 1994.

[17] M. Marcus, B. Santorini, M. A. Marcinkiewicz, “Building a Large Annotated Corpus of English: The Penn Treebank”, University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-93-87, October, 1993.